Tools and Techniques for Computational Economics

Thomas J. Sargent and John Stachurski

Jun 13, 2024

CONTENTS

This website presents a set of lectures on the tools and techniques required to study computational economics.

Part I

Linear Algebra

CHAPTER

ONE

LINEAR ALGEBRA

1.1 Overview

Linear algebra is one of the most useful branches of applied mathematics for economists to invest in.

For example, many applied problems in economics and finance require the solution of a linear system of equations, such as

$$\begin{aligned} y_1 &= ax_1 + bx_2 \\ y_2 &= cx_1 + dx_2 \end{aligned}$$

or, more generally,

$$\begin{aligned} y_1 &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1k}x_k \\ &\vdots \\ y_n &= a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nk}x_k \end{aligned} \tag{1.1}$$

The objective here is to solve for the "unknowns" $x_1, \ldots, x_k$ given $a_{11}, \ldots, a_{nk}$ and $y_1, \ldots, y_n$.

When considering such problems, it is essential that we first consider at least some of the following questions

- Does a solution actually exist?
- Are there in fact many solutions, and if so how should we interpret them?
- If no solution exists, is there a best "approximate" solution?
- If a solution exists, how should we compute it?

These are the kinds of topics addressed by linear algebra.

In this lecture we will cover the basics of linear and matrix algebra, treating both theory and computation.

We admit some overlap with this lecture, where operations on NumPy arrays were first explained.

Note that this lecture is more theoretical than most, and contains background material that will be used in applications as we go along.

Let's start with some imports:

```python
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import numpy as np
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from scipy.linalg import inv, solve, det, eig
```

1.2 Vectors

A *vector* of length n is just a sequence (or array, or tuple) of n numbers, which we write as $x = (x_1, \dots, x_n)$ or $x = [x_1, \dots, x_n]$.

We will write these sequences either horizontally or vertically as we please.

(Later, when we wish to perform certain matrix operations, it will become necessary to distinguish between the two)

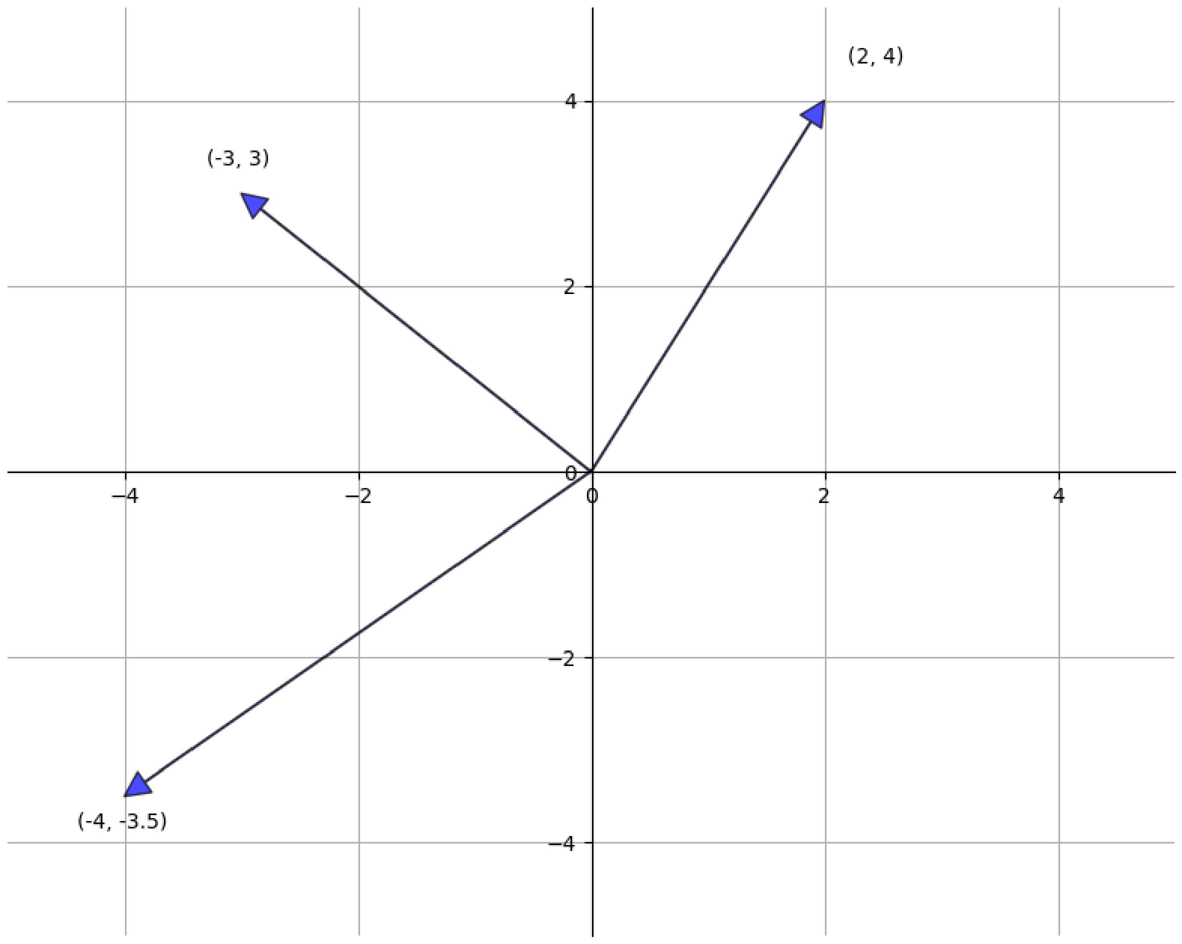
The set of all n-vectors is denoted by $\mathbb{R}^n$.

For example, $\mathbb{R}^2$ is the plane, and a vector in $\mathbb{R}^2$ is just a point in the plane.

Traditionally, vectors are represented visually as arrows from the origin to the point.

The following figure represents three vectors in this manner

```python
fig, ax = plt.subplots(figsize=(10, 8))
# Set the axes through the origin
for spine in ['left', 'bottom']:
    ax.spines[spine].set_position('zero')
for spine in ['right', 'top']:
    ax.spines[spine].set_color('none')

ax.set(xlim=(-5, 5), ylim=(-5, 5))
ax.grid()
vecs = ((2, 4), (-3, 3), (-4, -3.5))
for v in vecs:
    ax.annotate('', xy=v, xytext=(0, 0),
                arrowprops=dict(facecolor='blue',
                shrink=0,
                alpha=0.7,
                width=0.5))
    ax.text(1.1 * v[0], 1.1 * v[1], str(v))
plt.show()
```



1.2.1 Vector Operations

The two most common operators for vectors are addition and scalar multiplication, which we now describe.

As a matter of definition, when we add two vectors, we add them element-by-element

$$x + y = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} := \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix}$$

Scalar multiplication is an operation that takes a number γ and a vector x and produces

$$\gamma x := \begin{bmatrix} \gamma x_1 \\ \gamma x_2 \\ \vdots \\ \gamma x_n \end{bmatrix}$$

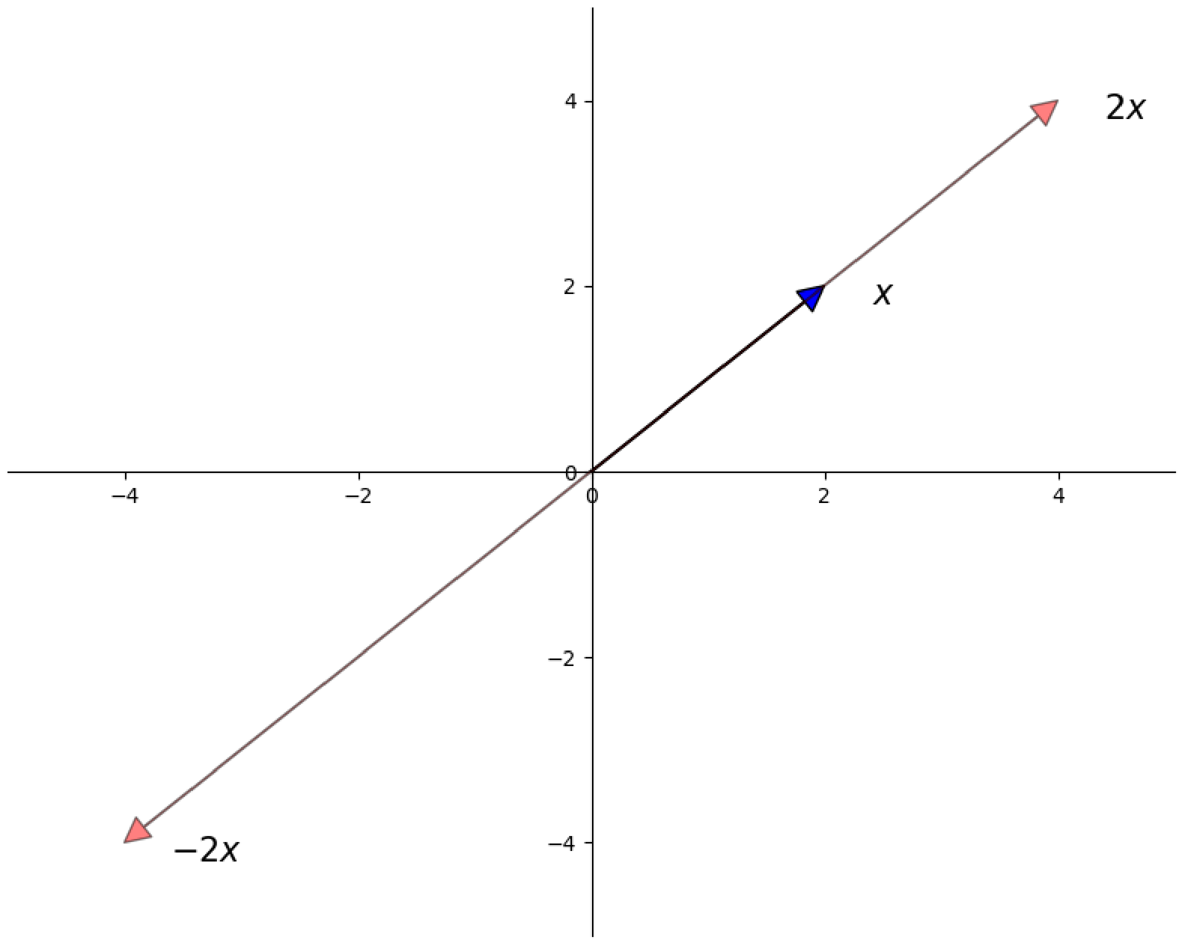
Scalar multiplication is illustrated in the next figure

```python
fig, ax = plt.subplots(figsize=(10, 8))
# Set the axes through the origin
for spine in ['left', 'bottom']:
```

(continues on next page)

(continued from previous page)

```
    ax.spines[spine].set_position('zero')
for spine in ['right', 'top']:
    ax.spines[spine].set_color('none')

ax.set(xlim=(-5, 5), ylim=(-5, 5))
x = (2, 2)
ax.annotate('', xy=x, xytext=(0, 0),
            arrowprops=dict(facecolor='blue',
            shrink=0,
            alpha=1,
            width=0.5))
ax.text(x[0] + 0.4, x[1] - 0.2, '$x$', fontsize='16')


scalars = (-2, 2)
x = np.array(x)

for s in scalars:
    v = s * x
    ax.annotate('', xy=v, xytext=(0, 0),
                arrowprops=dict(facecolor='red',
                shrink=0,
                alpha=0.5,
                width=0.5))
    ax.text(v[0] + 0.4, v[1] - 0.2, f'${s} x$', fontsize='16')
plt.show()
```



In Python, a vector can be represented as a list or tuple, such as `x = (2, 4, 6)`, but is more commonly represented as a NumPy array.

One advantage of NumPy arrays is that scalar multiplication and addition have very natural syntax

```
x = np.ones(3)            # Vector of three ones
y = np.array((2, 4, 6))   # Converts tuple (2, 4, 6) into array
x + y
```

```
array([3., 5., 7.])
```

```
4 * x
```

```
array([4., 4., 4.])
```

1.2.2 Inner Product and Norm

The *inner product* of vectors $x, y \in \mathbb{R}^n$ is defined as

$$x'y := \sum_{i=1}^{n} x_i y_i$$

Two vectors are called *orthogonal* if their inner product is zero.

The *norm* of a vector x represents its "length" (i.e., its distance from the zero vector) and is defined as

$$\|x\| := \sqrt{x'x} := \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}$$

The expression $\|x - y\|$ is thought of as the distance between x and y.

Continuing on from the previous example, the inner product and norm can be computed as follows

```
np.sum(x * y)          # Inner product of x and y
```

```
12.0
```

```
np.sqrt(np.sum(x**2))  # Norm of x, take one
```

```
1.7320508075688772
```

```
np.linalg.norm(x)      # Norm of x, take two
```

```
1.7320508075688772
```

1.2.3 Span

Given a set of vectors $A := \{a_1, \dots, a_k\}$ in $\mathbb{R}^n$, it's natural to think about the new vectors we can create by performing linear operations.

New vectors created in this manner are called *linear combinations* of A.

In particular, $y \in \mathbb{R}^n$ is a linear combination of $A := \{a_1, \dots, a_k\}$ if

$$y = \beta_1 a_1 + \cdots + \beta_k a_k \text{ for some scalars } \beta_1, \dots, \beta_k$$

In this context, the values $\beta_1, \dots, \beta_k$ are called the *coefficients* of the linear combination.

The set of linear combinations of A is called the *span* of A.

The next figure shows the span of $A = \{a_1, a_2\}$ in $\mathbb{R}^3$.

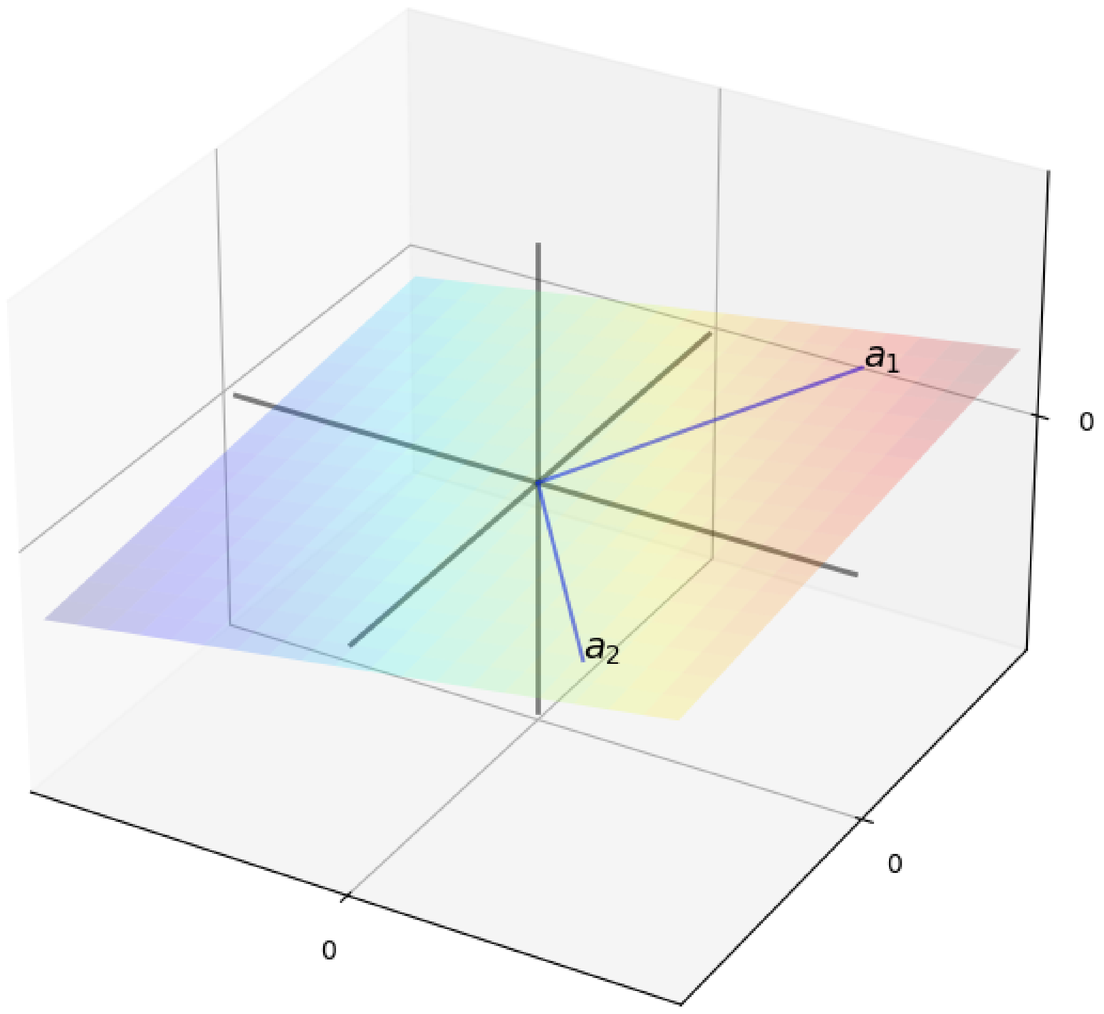
The span is a two-dimensional plane passing through these two points and the origin.

```
ax = plt.figure(figsize=(10, 8)).add_subplot(projection='3d')

x_min, x_max = -5, 5
y_min, y_max = -5, 5
```

(continues on next page)

(continued from previous page)

```
α, β = 0.2, 0.1

ax.set(xlim=(x_min, x_max), ylim=(x_min, x_max), zlim=(x_min, x_max),
       xticks=(0,), yticks=(0,), zticks=(0,))

gs = 3
z = np.linspace(x_min, x_max, gs)
x = np.zeros(gs)
y = np.zeros(gs)
ax.plot(x, y, z, 'k-', lw=2, alpha=0.5)
ax.plot(z, x, y, 'k-', lw=2, alpha=0.5)
ax.plot(y, z, x, 'k-', lw=2, alpha=0.5)


# Fixed linear function, to generate a plane
def f(x, y):
    return α * x + β * y

# Vector locations, by coordinate
x_coords = np.array((3, 3))
y_coords = np.array((4, -4))
z = f(x_coords, y_coords)
for i in (0, 1):
    ax.text(x_coords[i], y_coords[i], z[i], f'$a_{i+1}$', fontsize=14)

# Lines to vectors
for i in (0, 1):
    x = (0, x_coords[i])
    y = (0, y_coords[i])
    z = (0, f(x_coords[i], y_coords[i]))
    ax.plot(x, y, z, 'b-', lw=1.5, alpha=0.6)


# Draw the plane
grid_size = 20
xr2 = np.linspace(x_min, x_max, grid_size)
yr2 = np.linspace(y_min, y_max, grid_size)
x2, y2 = np.meshgrid(xr2, yr2)
z2 = f(x2, y2)
ax.plot_surface(x2, y2, z2, rstride=1, cstride=1, cmap=cm.jet,
                linewidth=0, antialiased=True, alpha=0.2)
plt.show()
```



Examples

If A contains only one vector $a_1 \in \mathbb{R}^2$, then its span is just the scalar multiples of a_1, which is the unique line passing through both a_1 and the origin.

If $A = \{e_1, e_2, e_3\}$ consists of the *canonical basis vectors* of $\mathbb{R}^3$, that is

$$e_1 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad e_2 := \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad e_3 := \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

then the span of A is all of $\mathbb{R}^3$, because, for any $x = (x_1, x_2, x_3) \in \mathbb{R}^3$, we can write

$$x = x_1 e_1 + x_2 e_2 + x_3 e_3$$

Now consider $A_0 = \{e_1, e_2, e_1 + e_2\}$.

If $y = (y_1, y_2, y_3)$ is any linear combination of these vectors, then $y_3 = 0$ (check it).

Hence A_0 fails to span all of $\mathbb{R}^3$.

1.2.4 Linear Independence

As we'll see, it's often desirable to find families of vectors with relatively large span, so that many vectors can be described by linear operators on a few vectors.

The condition we need for a set of vectors to have a large span is what's called linear independence.

In particular, a collection of vectors $A := \{a_1, \ldots, a_k\}$ in $\mathbb{R}^n$ is said to be

- *linearly dependent* if some strict subset of A has the same span as A.
- *linearly independent* if it is not linearly dependent.

Put differently, a set of vectors is linearly independent if no vector is redundant to the span and linearly dependent otherwise.

To illustrate the idea, recall *the figure* that showed the span of vectors $\{a_1, a_2\}$ in $\mathbb{R}^3$ as a plane through the origin.

If we take a third vector a_3 and form the set $\{a_1, a_2, a_3\}$, this set will be

- linearly dependent if a_3 lies in the plane
- linearly independent otherwise

As another illustration of the concept, since $\mathbb{R}^n$ can be spanned by n vectors (see the discussion of canonical basis vectors above), any collection of $m > n$ vectors in $\mathbb{R}^n$ must be linearly dependent.

The following statements are equivalent to linear independence of $A := \{a_1, \ldots, a_k\} \subset \mathbb{R}^n$

1. No vector in A can be formed as a linear combination of the other elements.
2. If $\beta_1 a_1 + \cdots \beta_k a_k = 0$ for scalars $\beta_1, \ldots, \beta_k$, then $\beta_1 = \cdots = \beta_k = 0$.

(The zero in the first expression is the origin of $\mathbb{R}^n$)

1.2.5 Unique Representations

Another nice thing about sets of linearly independent vectors is that each element in the span has a unique representation as a linear combination of these vectors.

In other words, if $A := \{a_1, \ldots, a_k\} \subset \mathbb{R}^n$ is linearly independent and

$$y = \beta_1 a_1 + \cdots \beta_k a_k$$

then no other coefficient sequence $\gamma_1, \ldots, \gamma_k$ will produce the same vector y.

Indeed, if we also have $y = \gamma_1 a_1 + \cdots \gamma_k a_k$, then

$$(\beta_1 - \gamma_1) a_1 + \cdots + (\beta_k - \gamma_k) a_k = 0$$

Linear independence now implies $\gamma_i = \beta_i$ for all i.

1.3 Matrices

Matrices are a neat way of organizing data for use in linear operations.

An $n \times k$ matrix is a rectangular array A of numbers with n rows and k columns:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix}$$

Often, the numbers in the matrix represent coefficients in a system of linear equations, as discussed at the start of this lecture.

For obvious reasons, the matrix A is also called a vector if either $n = 1$ or $k = 1$.

In the former case, A is called a *row vector*, while in the latter it is called a *column vector*.

If $n = k$, then A is called *square*.

The matrix formed by replacing a_{ij} by a_{ji} for every i and j is called the *transpose* of A and denoted A' or $A^\top$.

If $A = A'$, then A is called *symmetric*.

For a square matrix A, the i elements of the form a_{ii} for $i = 1, \ldots, n$ are called the *principal diagonal*.

A is called *diagonal* if the only nonzero entries are on the principal diagonal.

If, in addition to being diagonal, each element along the principal diagonal is equal to 1, then A is called the *identity matrix* and denoted by I.

1.3.1 Matrix Operations

Just as was the case for vectors, a number of algebraic operations are defined for matrices.

Scalar multiplication and addition are immediate generalizations of the vector case:

$$\gamma A = \gamma \begin{bmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nk} \end{bmatrix} := \begin{bmatrix} \gamma a_{11} & \cdots & \gamma a_{1k} \\ \vdots & \vdots & \vdots \\ \gamma a_{n1} & \cdots & \gamma a_{nk} \end{bmatrix}$$

and

$$A + B = \begin{bmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nk} \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \vdots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{bmatrix} := \begin{bmatrix} a_{11} + b_{11} & \cdots & a_{1k} + b_{1k} \\ \vdots & \vdots & \vdots \\ a_{n1} + b_{n1} & \cdots & a_{nk} + b_{nk} \end{bmatrix}$$

In the latter case, the matrices must have the same shape in order for the definition to make sense.

We also have a convention for *multiplying* two matrices.

The rule for matrix multiplication generalizes the idea of inner products discussed above and is designed to make multiplication play well with basic linear operations.

If A and B are two matrices, then their product AB is formed by taking as its i, j-th element the inner product of the i-th row of A and the j-th column of B.

There are many tutorials to help you visualize this operation, such as this one, or the discussion on the Wikipedia page.

If A is $n \times k$ and B is $j \times m$, then to multiply A and B we require $k = j$, and the resulting matrix AB is $n \times m$.

As perhaps the most important special case, consider multiplying $n \times k$ matrix A and $k \times 1$ column vector x.

According to the preceding rule, this gives us an $n \times 1$ column vector

$$Ax = \begin{bmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nk} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix} := \begin{bmatrix} a_{11} x_1 + \cdots + a_{1k} x_k \\ \vdots \\ a_{n1} x_1 + \cdots + a_{nk} x_k \end{bmatrix} \tag{1.2}$$

Note: AB and BA are not generally the same thing.

Another important special case is the identity matrix.

You should check that if A is $n \times k$ and I is the $k \times k$ identity matrix, then $AI = A$.

If I is the $n \times n$ identity matrix, then $IA = A$.

1.3.2 Matrices in NumPy

NumPy arrays are also used as matrices, and have fast, efficient functions and methods for all the standard matrix operations[1].

You can create them manually from tuples of tuples (or lists of lists) as follows

```
A = ((1, 2),
     (3, 4))

type(A)
```

```
tuple
```

```
A = np.array(A)

type(A)
```

```
numpy.ndarray
```

```
A.shape
```

```
(2, 2)
```

The `shape` attribute is a tuple giving the number of rows and columns — see here for more discussion.

To get the transpose of `A`, use `A.transpose()` or, more simply, `A.T`.

There are many convenient functions for creating common matrices (matrices of zeros, ones, etc.) — see here.

Since operations are performed elementwise by default, scalar multiplication and addition have very natural syntax

```
A = np.identity(3)
B = np.ones((3, 3))
2 * A
```

[1] Although there is a specialized matrix data type defined in NumPy, it's more standard to work with ordinary NumPy arrays. See this discussion.

```
array([[2., 0., 0.],
       [0., 2., 0.],
       [0., 0., 2.]])
```

```
A + B
```

```
array([[2., 1., 1.],
       [1., 2., 1.],
       [1., 1., 2.]])
```

To multiply matrices we use the `@` symbol.

In particular, `A @ B` is matrix multiplication, whereas `A * B` is element-by-element multiplication.

See here for more discussion.

1.3.3 Matrices as Maps

Each $n \times k$ matrix A can be identified with a function $f(x) = Ax$ that maps $x \in \mathbb{R}^k$ into $y = Ax \in \mathbb{R}^n$.

These kinds of functions have a special property: they are *linear*.

A function $f : \mathbb{R}^k \to \mathbb{R}^n$ is called *linear* if, for all $x, y \in \mathbb{R}^k$ and all scalars α, β, we have

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

You can check that this holds for the function $f(x) = Ax + b$ when b is the zero vector and fails when b is nonzero.

In fact, it's known that f is linear if and *only if* there exists a matrix A such that $f(x) = Ax$ for all x.

1.4 Solving Systems of Equations

Recall again the system of equations (1.1).

If we compare (1.1) and (1.2), we see that (1.1) can now be written more conveniently as

$$y = Ax \tag{1.3}$$

The problem we face is to determine a vector $x \in \mathbb{R}^k$ that solves (1.3), taking y and A as given.

This is a special case of a more general problem: Find an x such that $y = f(x)$.

Given an arbitrary function f and a y, is there always an x such that $y = f(x)$?

If so, is it always unique?

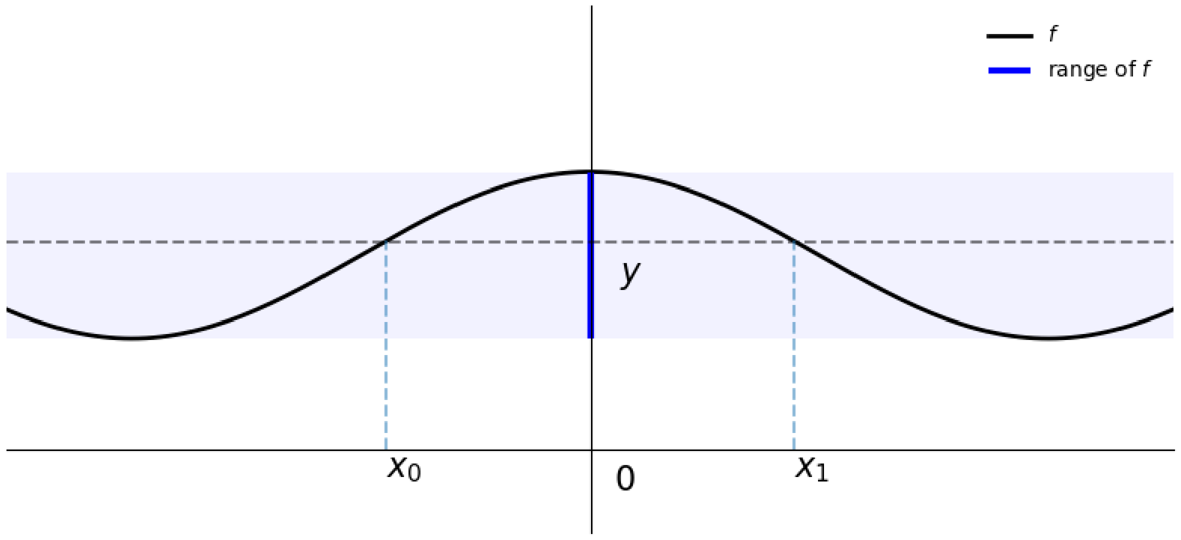
The answer to both these questions is negative, as the next figure shows

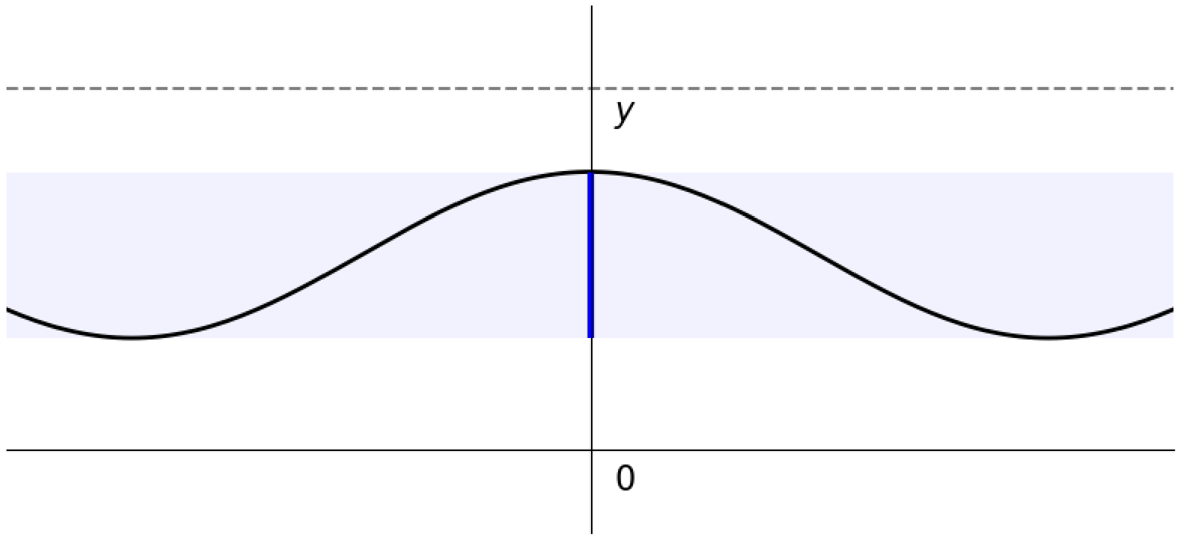
```
def f(x):
    return 0.6 * np.cos(4 * x) + 1.4


xmin, xmax = -1, 1
x = np.linspace(xmin, xmax, 160)
```

(continues on next page)

(continued from previous page)

```python
y = f(x)
ya, yb = np.min(y), np.max(y)

fig, axes = plt.subplots(2, 1, figsize=(10, 10))

for ax in axes:
    # Set the axes through the origin
    for spine in ['left', 'bottom']:
        ax.spines[spine].set_position('zero')
    for spine in ['right', 'top']:
        ax.spines[spine].set_color('none')

    ax.set(ylim=(-0.6, 3.2), xlim=(xmin, xmax),
           yticks=(), xticks=())

    ax.plot(x, y, 'k-', lw=2, label='$f$')
    ax.fill_between(x, ya, yb, facecolor='blue', alpha=0.05)
    ax.vlines([0], ya, yb, lw=3, color='blue', label='range of $f$')
    ax.text(0.04, -0.3, '$0$', fontsize=16)

ax = axes[0]

ax.legend(loc='upper right', frameon=False)
ybar = 1.5
ax.plot(x, x * 0 + ybar, 'k--', alpha=0.5)
ax.text(0.05, 0.8 * ybar, '$y$', fontsize=16)
for i, z in enumerate((-0.35, 0.35)):
    ax.vlines(z, 0, f(z), linestyle='--', alpha=0.5)
    ax.text(z, -0.2, f'$x_{i}$', fontsize=16)

ax = axes[1]

ybar = 2.6
ax.plot(x, x * 0 + ybar, 'k--', alpha=0.5)
ax.text(0.04, 0.91 * ybar, '$y$', fontsize=16)

plt.show()
```





In the first plot, there are multiple solutions, as the function is not one-to-one, while in the second there are no solutions, since y lies outside the range of f.

Can we impose conditions on A in (1.3) that rule out these problems?

In this context, the most important thing to recognize about the expression Ax is that it corresponds to a linear combination of the columns of A.

In particular, if $a_1, \dots, a_k$ are the columns of A, then

$$Ax = x_1 a_1 + \cdots + x_k a_k$$

Hence the range of $f(x) = Ax$ is exactly the span of the columns of A.

We want the range to be large so that it contains arbitrary y.

As you might recall, the condition that we want for the span to be large is *linear independence*.

A happy fact is that linear independence of the columns of A also gives us uniqueness.

Indeed, it follows from our *earlier discussion* that if $\{a_1, \ldots, a_k\}$ are linearly independent and $y = Ax = x_1 a_1 + \cdots + x_k a_k$, then no $z \neq x$ satisfies $y = Az$.

1.4.1 The Square Matrix Case

Let's discuss some more details, starting with the case where A is $n \times n$.

This is the familiar case where the number of unknowns equals the number of equations.

For arbitrary $y \in \mathbb{R}^n$, we hope to find a unique $x \in \mathbb{R}^n$ such that $y = Ax$.

In view of the observations immediately above, if the columns of A are linearly independent, then their span, and hence the range of $f(x) = Ax$, is all of $\mathbb{R}^n$.

Hence there always exists an x such that $y = Ax$.

Moreover, the solution is unique.

In particular, the following are equivalent

1. The columns of A are linearly independent.
2. For any $y \in \mathbb{R}^n$, the equation $y = Ax$ has a unique solution.

The property of having linearly independent columns is sometimes expressed as having *full column rank*.

Inverse Matrices

Can we give some sort of expression for the solution?

If y and A are scalar with $A \neq 0$, then the solution is $x = A^{-1} y$.

A similar expression is available in the matrix case.

In particular, if square matrix A has full column rank, then it possesses a multiplicative *inverse matrix* A^{-1}, with the property that $AA^{-1} = A^{-1}A = I$.

As a consequence, if we pre-multiply both sides of $y = Ax$ by A^{-1}, we get $x = A^{-1} y$.

This is the solution that we're looking for.

Determinants

Another quick comment about square matrices is that to every such matrix we assign a unique number called the *determinant* of the matrix — you can find the expression for it here.

If the determinant of A is not zero, then we say that A is *nonsingular*.

Perhaps the most important fact about determinants is that A is nonsingular if and only if A is of full column rank.

This gives us a useful one-number summary of whether or not a square matrix can be inverted.

1.4.2 More Rows than Columns

This is the $n \times k$ case with $n > k$.

This case is very important in many settings, not least in the setting of linear regression (where n is the number of observations, and k is the number of explanatory variables).

Given arbitrary $y \in \mathbb{R}^n$, we seek an $x \in \mathbb{R}^k$ such that $y = Ax$.

In this setting, the existence of a solution is highly unlikely.

Without much loss of generality, let's go over the intuition focusing on the case where the columns of A are linearly independent.

It follows that the span of the columns of A is a k-dimensional subspace of $\mathbb{R}^n$.

This span is very "unlikely" to contain arbitrary $y \in \mathbb{R}^n$.

To see why, recall the *figure above*, where $k = 2$ and $n = 3$.

Imagine an arbitrarily chosen $y \in \mathbb{R}^3$, located somewhere in that three-dimensional space.

What's the likelihood that y lies in the span of $\{a_1, a_2\}$ (i.e., the two dimensional plane through these points)?

In a sense, it must be very small, since this plane has zero "thickness".

As a result, in the $n > k$ case we usually give up on existence.

However, we can still seek the best approximation, for example, an x that makes the distance $\|y - Ax\|$ as small as possible.

To solve this problem, one can use either calculus or the theory of orthogonal projections.

The solution is known to be $\hat{x} = (A'A)^{-1}A'y$ — see for example chapter 3 of these notes.

1.4.3 More Columns than Rows

This is the $n \times k$ case with $n < k$, so there are fewer equations than unknowns.

In this case there are either no solutions or infinitely many — in other words, uniqueness never holds.

For example, consider the case where $k = 3$ and $n = 2$.

Thus, the columns of A consists of 3 vectors in $\mathbb{R}^2$.

This set can never be linearly independent, since it is possible to find two vectors that span $\mathbb{R}^2$.

(For example, use the canonical basis vectors)

It follows that one column is a linear combination of the other two.

For example, let's say that $a_1 = \alpha a_2 + \beta a_3$.

Then if $y = Ax = x_1 a_1 + x_2 a_2 + x_3 a_3$, we can also write

$$y = x_1(\alpha a_2 + \beta a_3) + x_2 a_2 + x_3 a_3 = (x_1 \alpha + x_2) a_2 + (x_1 \beta + x_3) a_3$$

In other words, uniqueness fails.

1.4.4 Linear Equations with SciPy

Here's an illustration of how to solve linear equations with SciPy's `linalg` submodule.

All of these routines are Python front ends to time-tested and highly optimized FORTRAN code

```
A = ((1, 2), (3, 4))
A = np.array(A)
y = np.ones((2, 1))  # Column vector
det(A)  # Check that A is nonsingular, and hence invertible
```

```
-2.0
```

```
A_inv = inv(A)  # Compute the inverse
A_inv
```

```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

```
x = A_inv @ y  # Solution
A @ x          # Should equal y
```

```
array([[1.],
       [1.]])
```

```
solve(A, y)  # Produces the same solution
```

```
array([[-1.],
       [ 1.]])
```

Observe how we can solve for $x = A^{-1}y$ by either via `inv(A) @ y`, or using `solve(A, y)`.

The latter method uses a different algorithm (LU decomposition) that is numerically more stable, and hence should almost always be preferred.

To obtain the least-squares solution $\hat{x} = (A'A)^{-1}A'y$, use `scipy.linalg.lstsq(A, y)`.

1.5 Eigenvalues and Eigenvectors

Let A be an $n \times n$ square matrix.

If λ is scalar and v is a non-zero vector in $\mathbb{R}^n$ such that

$$Av = \lambda v$$

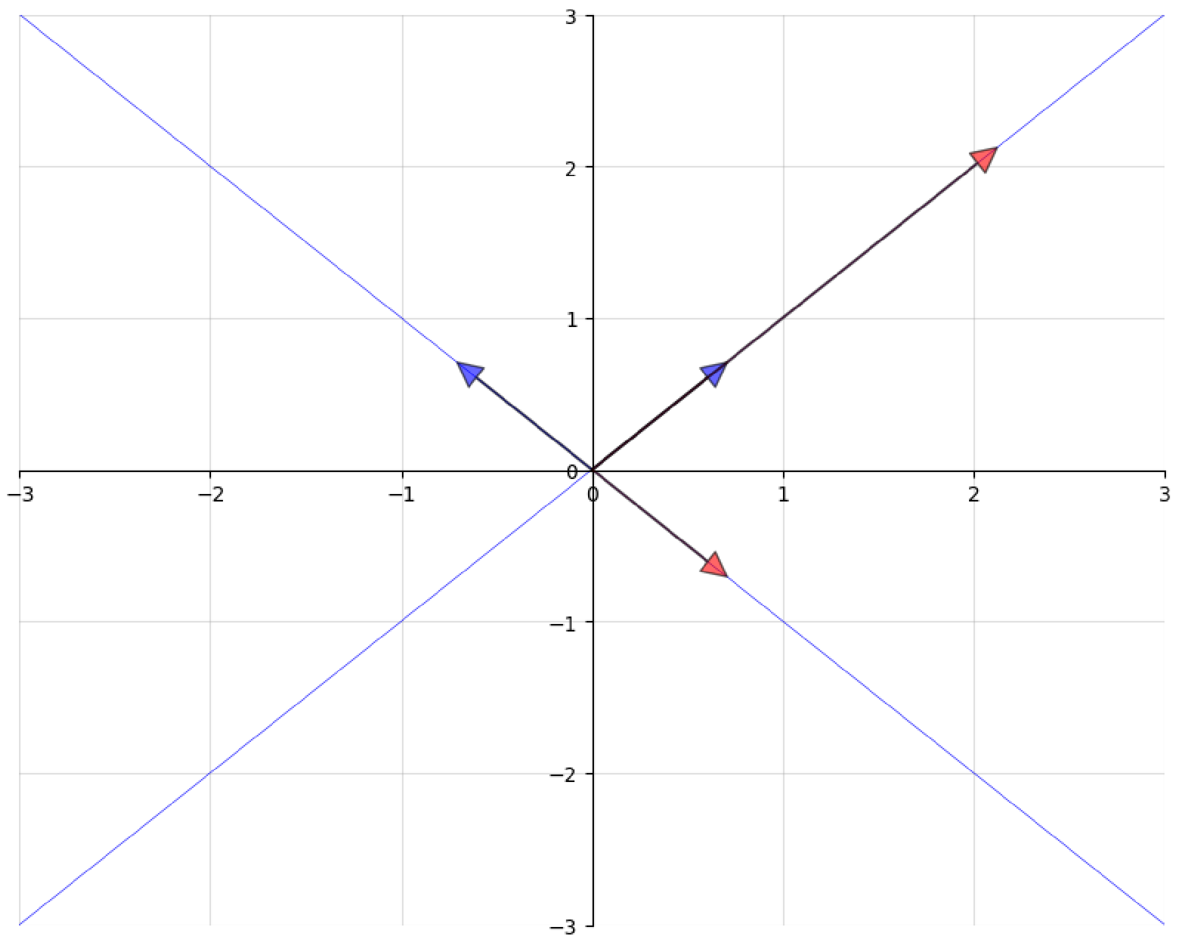
then we say that λ is an *eigenvalue* of A, and v is an *eigenvector*.

Thus, an eigenvector of A is a vector such that when the map $f(x) = Ax$ is applied, v is merely scaled.

The next figure shows two eigenvectors (blue arrows) and their images under A (red arrows).

As expected, the image Av of each v is just a scaled version of the original

```python
A = ((1, 2),
     (2, 1))
A = np.array(A)
evals, evecs = eig(A)
evecs = evecs[:, 0], evecs[:, 1]

fig, ax = plt.subplots(figsize=(10, 8))
# Set the axes through the origin
for spine in ['left', 'bottom']:
    ax.spines[spine].set_position('zero')
for spine in ['right', 'top']:
    ax.spines[spine].set_color('none')
ax.grid(alpha=0.4)

xmin, xmax = -3, 3
ymin, ymax = -3, 3
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))

# Plot each eigenvector
for v in evecs:
    ax.annotate('', xy=v, xytext=(0, 0),
                arrowprops=dict(facecolor='blue',
                shrink=0,
                alpha=0.6,
                width=0.5))

# Plot the image of each eigenvector
for v in evecs:
    v = A @ v
    ax.annotate('', xy=v, xytext=(0, 0),
                arrowprops=dict(facecolor='red',
                shrink=0,
                alpha=0.6,
                width=0.5))

# Plot the lines they run through
x = np.linspace(xmin, xmax, 3)
for v in evecs:
    a = v[1] / v[0]
    ax.plot(x, a * x, 'b-', lw=0.4)

plt.show()
```



The eigenvalue equation is equivalent to $(A - \lambda I)v = 0$, and this has a nonzero solution v only when the columns of $A - \lambda I$ are linearly dependent.

This in turn is equivalent to stating that the determinant is zero.

Hence to find all eigenvalues, we can look for λ such that the determinant of $A - \lambda I$ is zero.

This problem can be expressed as one of solving for the roots of a polynomial in λ of degree n.

This in turn implies the existence of n solutions in the complex plane, although some might be repeated.

Some nice facts about the eigenvalues of a square matrix A are as follows

1. The determinant of A equals the product of the eigenvalues.
2. The trace of A (the sum of the elements on the principal diagonal) equals the sum of the eigenvalues.
3. If A is symmetric, then all of its eigenvalues are real.
4. If A is invertible and $\lambda_1, \ldots, \lambda_n$ are its eigenvalues, then the eigenvalues of A^{-1} are $1/\lambda_1, \ldots, 1/\lambda_n$.

A corollary of the first statement is that a matrix is invertible if and only if all its eigenvalues are nonzero.

Using SciPy, we can solve for the eigenvalues and eigenvectors of a matrix as follows

```
A = ((1, 2),
     (2, 1))
```

(continues on next page)

(continued from previous page)

```
A = np.array(A)
evals, evecs = eig(A)
evals
```

```
array([ 3.+0.j, -1.+0.j])
```

```
evecs
```

```
array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]])
```

Note that the *columns* of `evecs` are the eigenvectors.

Since any scalar multiple of an eigenvector is an eigenvector with the same eigenvalue (check it), the eig routine normalizes the length of each eigenvector to one.

1.5.1 Generalized Eigenvalues

It is sometimes useful to consider the *generalized eigenvalue problem*, which, for given matrices A and B, seeks generalized eigenvalues λ and eigenvectors v such that

$$Av = \lambda Bv$$

This can be solved in SciPy via `scipy.linalg.eig(A, B)`.

Of course, if B is square and invertible, then we can treat the generalized eigenvalue problem as an ordinary eigenvalue problem $B^{-1}Av = \lambda v$, but this is not always the case.

1.6 Further Topics

We round out our discussion by briefly mentioning several other important topics.

1.6.1 Series Expansions

Recall the usual summation formula for a geometric progression, which states that if $|a| < 1$, then $\sum_{k=0}^{\infty} a^k = (1-a)^{-1}$.

A generalization of this idea exists in the matrix setting.

Matrix Norms

Let A be a square matrix, and let

$$\|A\| := \max_{\|x\|=1} \|Ax\|$$

The norms on the right-hand side are ordinary vector norms, while the norm on the left-hand side is a *matrix norm* — in this case, the so-called *spectral norm*.

For example, for a square matrix S, the condition $\|S\| < 1$ means that S is *contractive*, in the sense that it pulls all vectors towards the origin[2].

Neumann's Theorem

Let A be a square matrix and let $A^k := AA^{k-1}$ with $A^1 := A$.

In other words, A^k is the k-th power of A.

Neumann's theorem states the following: If $\|A^k\| < 1$ for some $k \in \mathbb{N}$, then $I - A$ is invertible, and

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k \tag{1.4}$$

Spectral Radius

A result known as Gelfand's formula tells us that, for any square matrix A,

$$\rho(A) = \lim_{k \to \infty} \|A^k\|^{1/k}$$

Here $\rho(A)$ is the *spectral radius*, defined as $\max_i |\lambda_i|$, where $\{\lambda_i\}_i$ is the set of eigenvalues of A.

As a consequence of Gelfand's formula, if all eigenvalues are strictly less than one in modulus, there exists a k with $\|A^k\| < 1$.

In which case (1.4) is valid.

1.6.2 Positive Definite Matrices

Let A be a symmetric $n \times n$ matrix.

We say that A is

1. *positive definite* if $x' A x > 0$ for every $x \in \mathbb{R}^n \quad \{0\}$
2. *positive semi-definite* or *nonnegative definite* if $x' A x \geq 0$ for every $x \in \mathbb{R}^n$

Analogous definitions exist for negative definite and negative semi-definite matrices.

It is notable that if A is positive definite, then all of its eigenvalues are strictly positive, and hence A is invertible (with positive definite inverse).

1.6.3 Differentiating Linear and Quadratic Forms

The following formulas are useful in many economic contexts. Let

- z, x and a all be $n \times 1$ vectors
- A be an $n \times n$ matrix
- B be an $m \times n$ matrix and y be an $m \times 1$ vector

Then

1. $\frac{\partial a' x}{\partial x} = a$

[2] Suppose that $\|S\| < 1$. Take any nonzero vector x, and let $r := \|x\|$. We have $\|Sx\| = r\|S(x/r)\| \leq r\|S\| < r = \|x\|$. Hence every point is pulled towards the origin.

2. $\frac{\partial Ax}{\partial x} = A'$
3. $\frac{\partial x'Ax}{\partial x} = (A + A')x$
4. $\frac{\partial y'Bz}{\partial y} = Bz$
5. $\frac{\partial y'Bz}{\partial B} = yz'$

Exercise 1.7.1 below asks you to apply these formulas.

1.6.4 Further Reading

The documentation of the `scipy.linalg` submodule can be found here.

Chapters 2 and 3 of the Econometric Theory contains a discussion of linear algebra along the same lines as above, with solved exercises.

If you don't mind a slightly abstract approach, a nice intermediate-level text on linear algebra is [Jänich, 1994].

1.7 Exercises

Exercise 1.7.1

Let x be a given $n \times 1$ vector and consider the problem

$$v(x) = \max_{y,u} \{-y'Py - u'Qu\}$$

subject to the linear constraint

$$y = Ax + Bu$$

Here

- P is an $n \times n$ matrix and Q is an $m \times m$ matrix
- A is an $n \times n$ matrix and B is an $n \times m$ matrix
- both P and Q are symmetric and positive semidefinite

(What must the dimensions of y and u be to make this a well-posed problem?)

One way to solve the problem is to form the Lagrangian

$$\mathcal{L} = -y'Py - u'Qu + \lambda' [Ax + Bu - y]$$

where λ is an $n \times 1$ vector of Lagrange multipliers.

Try applying the formulas given above for differentiating quadratic and linear forms to obtain the first-order conditions for maximizing $\mathcal{L}$ with respect to y, u and minimizing it with respect to λ.

Show that these conditions imply that

1. $\lambda = -2Py$.
2. The optimizing choice of u satisfies $u = -(Q + B'PB)^{-1}B'PAx$.
3. The function v satisfies $v(x) = -x'\tilde{P}x$ where $\tilde{P} = A'PA - A'PB(Q + B'PB)^{-1}B'PA$.

As we will see, in economic contexts Lagrange multipliers often are shadow prices.

Note: If we don't care about the Lagrange multipliers, we can substitute the constraint into the objective function, and then just maximize $-(Ax + Bu)'P(Ax + Bu) - u'Qu$ with respect to u. You can verify that this leads to the same maximizer.

Solution to Exercise 1.7.1

We have an optimization problem:

$$v(x) = \max_{y,u}\{-y'Py - u'Qu\}$$

s.t.

$$y = Ax + Bu$$

with primitives

- P be a symmetric and positive semidefinite $n \times n$ matrix
- Q be a symmetric and positive semidefinite $m \times m$ matrix
- A an $n \times n$ matrix
- B an $n \times m$ matrix

The associated Lagrangian is:

$$L = -y'Py - u'Qu + \lambda'[Ax + Bu - y]$$

Step 1.

Differentiating Lagrangian equation w.r.t y and setting its derivative equal to zero yields

$$\frac{\partial L}{\partial y} = -(P + P')y - \lambda = -2Py - \lambda = 0\,,$$

since P is symmetric.

Accordingly, the first-order condition for maximizing L w.r.t. y implies

$$\lambda = -2Py$$

Step 2.

Differentiating Lagrangian equation w.r.t. u and setting its derivative equal to zero yields

$$\frac{\partial L}{\partial u} = -(Q + Q')u - B'\lambda = -2Qu + B'\lambda = 0$$

Substituting $\lambda = -2Py$ gives

$$Qu + B'Py = 0$$

Substituting the linear constraint $y = Ax + Bu$ into above equation gives

$$Qu + B'P(Ax + Bu) = 0$$

$$(Q + B'PB)u + B'PAx = 0$$

which is the first-order condition for maximizing L w.r.t. u.

Thus, the optimal choice of u must satisfy

$$u = -(Q + B'PB)^{-1}B'PAx \ ,$$

which follows from the definition of the first-order conditions for Lagrangian equation.

Step 3.

Rewriting our problem by substituting the constraint into the objective function, we get

$$v(x) = \max_u \{-(Ax + Bu)'P(Ax + Bu) - u'Qu\}$$

Since we know the optimal choice of u satisfies $u = -(Q + B'PB)^{-1}B'PAx$, then

$$v(x) = -(Ax + Bu)'P(Ax + Bu) - u'Qu \ \ with \ \ u = -(Q + B'PB)^{-1}B'PAx$$

To evaluate the function

$$\begin{aligned} v(x) &= -(Ax + Bu)'P(Ax + Bu) - u'Qu \\ &= -(x'A' + u'B')P(Ax + Bu) - u'Qu \\ &= -x'A'PAx - u'B'PAx - x'A'PBu - u'B'PBu - u'Qu \\ &= -x'A'PAx - 2u'B'PAx - u'(Q + B'PB)u \end{aligned}$$

For simplicity, denote by $S := (Q + B'PB)^{-1}B'PA$, then $u = -Sx$.

Regarding the second term $-2u'B'PAx$,

$$\begin{aligned} -2u'B'PAx &= -2x'S'B'PAx \\ &= 2x'A'PB(Q + B'PB)^{-1}B'PAx \end{aligned}$$

Notice that the term $(Q + B'PB)^{-1}$ is symmetric as both P and Q are symmetric.

Regarding the third term $-u'(Q + B'PB)u$,

$$\begin{aligned} -u'(Q + B'PB)u &= -x'S'(Q + B'PB)Sx \\ &= -x'A'PB(Q + B'PB)^{-1}B'PAx \end{aligned}$$

Hence, the summation of second and third terms is $x'A'PB(Q + B'PB)^{-1}B'PAx$.

This implies that

$$\begin{aligned} v(x) &= -x'A'PAx - 2u'B'PAx - u'(Q + B'PB)u \\ &= -x'A'PAx + x'A'PB(Q + B'PB)^{-1}B'PAx \\ &= -x'[A'PA - A'PB(Q + B'PB)^{-1}B'PA]x \end{aligned}$$

Therefore, the solution to the optimization problem $v(x) = -x'\tilde{P}x$ follows the above result by denoting $\tilde{P} := A'PA - A'PB(Q + B'PB)^{-1}B'PA$

CHAPTER

TWO

QR DECOMPOSITION

2.1 Overview

This lecture describes the QR decomposition and how it relates to

- Orthogonal projection and least squares
- A Gram-Schmidt process
- Eigenvalues and eigenvectors

We'll write some Python code to help consolidate our understandings.

2.2 Matrix Factorization

The QR decomposition (also called the QR factorization) of a matrix is a decomposition of a matrix into the product of an orthogonal matrix and a triangular matrix.

A QR decomposition of a real matrix A takes the form

$$A = QR$$

where

- Q is an orthogonal matrix (so that $Q^T Q = I$)
- R is an upper triangular matrix

We'll use a **Gram-Schmidt process** to compute a QR decomposition

Because doing so is so educational, we'll write our own Python code to do the job

2.3 Gram-Schmidt process

We'll start with a **square** matrix A.

If a square matrix A is nonsingular, then a QR factorization is unique.

We'll deal with a rectangular matrix A later.

Actually, our algorithm will work with a rectangular A that is not square.

2.3.1 Gram-Schmidt process for square A

Here we apply a Gram-Schmidt process to the **columns** of matrix A.

In particular, let

$$A = \left[\ a_1 \mid a_2 \mid \cdots \mid a_n\ \right]$$

Let $||\cdot||$ denote the L2 norm.

The Gram-Schmidt algorithm repeatedly combines the following two steps in a particular order

- **normalize** a vector to have unit norm
- **orthogonalize** the next vector

To begin, we set $u_1 = a_1$ and then **normalize**:

$$u_1 = a_1, \quad e_1 = \frac{u_1}{||u_1||}$$

We **orgonalize** first to compute u_2 and then **normalize** to create e_2:

$$u_2 = a_2 - (a_2 \cdot e_1)e_1, \quad e_2 = \frac{u_2}{||u_2||}$$

We invite the reader to verify that e_1 is orthogonal to e_2 by checking that $e_1 \cdot e_2 = 0$.

The Gram-Schmidt procedure continues iterating.

Thus, for $k = 2, \ldots, n-1$ we construct

$$u_{k+1} = a_{k+1} - (a_{k+1} \cdot e_1)e_1 - \cdots - (a_{k+1} \cdot e_k)e_k, \quad e_{k+1} = \frac{u_{k+1}}{||u_{k+1}||}$$

Here $(a_j \cdot e_i)$ can be interpreted as the linear least squares **regression coefficient** of a_j on e_i

- it is the inner product of a_j and e_i divided by the inner product of e_i where $e_i \cdot e_i = 1$, as *normalization* has assured us.
- this regression coefficient has an interpretation as being a **covariance** divided by a **variance**

It can be verified that

$$A = \left[\ a_1 \mid a_2 \mid \cdots \mid a_n\ \right] = \left[\ e_1 \mid e_2 \mid \cdots \mid e_n\ \right] \begin{bmatrix} a_1 \cdot e_1 & a_2 \cdot e_1 & \cdots & a_n \cdot e_1 \\ 0 & a_2 \cdot e_2 & \cdots & a_n \cdot e_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \cdot e_n \end{bmatrix}$$

Thus, we have constructed the decomposision

$$A = QR$$

where

$$Q = \left[\ a_1 \mid a_2 \mid \cdots \mid a_n\ \right] = \left[\ e_1 \mid e_2 \mid \cdots \mid e_n\ \right]$$

and

$$R = \begin{bmatrix} a_1 \cdot e_1 & a_2 \cdot e_1 & \cdots & a_n \cdot e_1 \\ 0 & a_2 \cdot e_2 & \cdots & a_n \cdot e_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \cdot e_n \end{bmatrix}$$

2.3.2 A not square

Now suppose that A is an $n \times m$ matrix where $m > n$.

Then a QR decomposition is

$$A = \left[\ a_1 \mid a_2 \mid \cdots \mid a_m\ \right] = \left[\ e_1 \mid e_2 \mid \cdots \mid e_n\ \right] \begin{bmatrix} a_1 \cdot e_1 & a_2 \cdot e_1 & \cdots & a_n \cdot e_1 & a_{n+1} \cdot e_1 & \cdots & a_m \cdot e_1 \\ 0 & a_2 \cdot e_2 & \cdots & a_n \cdot e_2 & a_{n+1} \cdot e_2 & \cdots & a_m \cdot e_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \cdot e_n & a_{n+1} \cdot e_n & \cdots & a_m \cdot e_n \end{bmatrix}$$

which implies that

$$\begin{aligned} a_1 &= (a_1 \cdot e_1) e_1 \\ a_2 &= (a_2 \cdot e_1) e_1 + (a_2 \cdot e_2) e_2 \\ \vdots &\quad \vdots \\ a_n &= (a_n \cdot e_1) e_1 + (a_n \cdot e_2) e_2 + \cdots + (a_n \cdot e_n) e_n \\ a_{n+1} &= (a_{n+1} \cdot e_1) e_1 + (a_{n+1} \cdot e_2) e_2 + \cdots + (a_{n+1} \cdot e_n) e_n \\ \vdots &\quad \vdots \\ a_m &= (a_m \cdot e_1) e_1 + (a_m \cdot e_2) e_2 + \cdots + (a_m \cdot e_n) e_n \end{aligned}$$

2.4 Some Code

Now let's write some homemade Python code to implement a QR decomposition by deploying the Gram-Schmidt process described above.

```python
import numpy as np
from scipy.linalg import qr
```

```python
def QR_Decomposition(A):
    n, m = A.shape # get the shape of A

    Q = np.empty((n, n)) # initialize matrix Q
    u = np.empty((n, n)) # initialize matrix u

    u[:, 0] = A[:, 0]
    Q[:, 0] = u[:, 0] / np.linalg.norm(u[:, 0])

    for i in range(1, n):

        u[:, i] = A[:, i]
        for j in range(i):
            u[:, i] -= (A[:, i] @ Q[:, j]) * Q[:, j] # get each u vector

        Q[:, i] = u[:, i] / np.linalg.norm(u[:, i]) # compute each e vetor

    R = np.zeros((n, m))
    for i in range(n):
        for j in range(i, m):
            R[i, j] = A[:, j] @ Q[:, i]

    return Q, R
```

The preceding code is fine but can benefit from some further housekeeping.

We want to do this because later in this notebook we want to compare results from using our homemade code above with the code for a QR that the Python `scipy` package delivers.

There can be be sign differences between the Q and R matrices produced by different numerical algorithms.

All of these are valid QR decompositions because of how the sign differences cancel out when we compute QR.

However, to make the results from our homemade function and the QR module in `scipy` comparable, let's require that Q have positive diagonal entries.

We do this by adjusting the signs of the columns in Q and the rows in R appropriately.

To accomplish this we'll define a pair of functions.

```
def diag_sign(A):
    "Compute the signs of the diagonal of matrix A"

    D = np.diag(np.sign(np.diag(A)))

    return D

def adjust_sign(Q, R):
    """
    Adjust the signs of the columns in Q and rows in R to
    impose positive diagonal of Q
    """

    D = diag_sign(Q)

    Q[:, :] = Q @ D
    R[:, :] = D @ R

    return Q, R
```

2.5 Example

Now let's do an example.

```
A = np.array([[1.0, 1.0, 0.0], [1.0, 0.0, 1.0], [0.0, 1.0, 1.0]])
# A = np.array([[1.0, 0.5, 0.2], [0.5, 0.5, 1.0], [0.0, 1.0, 1.0]])
# A = np.array([[1.0, 0.5, 0.2], [0.5, 0.5, 1.0]])

A
```

```
array([[1., 1., 0.],
       [1., 0., 1.],
       [0., 1., 1.]])
```

```
Q, R = adjust_sign(*QR_Decomposition(A))
```

```
Q
```

```
array([[ 0.70710678, -0.40824829, -0.57735027],
       [ 0.70710678,  0.40824829,  0.57735027],
       [ 0.        , -0.81649658,  0.57735027]])
```

```
R
```

```
array([[ 1.41421356,  0.70710678,  0.70710678],
       [ 0.        , -1.22474487, -0.40824829],
       [ 0.        ,  0.        ,  1.15470054]])
```

Let's compare outcomes with what the `scipy` package produces

```
Q_scipy, R_scipy = adjust_sign(*qr(A))
```

```
print('Our Q: \n', Q)
print('\n')
print('Scipy Q: \n', Q_scipy)
```

```
Our Q:
 [[ 0.70710678 -0.40824829 -0.57735027]
 [ 0.70710678  0.40824829  0.57735027]
 [ 0.         -0.81649658  0.57735027]]


Scipy Q:
 [[ 0.70710678 -0.40824829 -0.57735027]
 [ 0.70710678  0.40824829  0.57735027]
 [ 0.         -0.81649658  0.57735027]]
```

```
print('Our R: \n', R)
print('\n')
print('Scipy R: \n', R_scipy)
```

```
Our R:
 [[ 1.41421356  0.70710678  0.70710678]
 [ 0.         -1.22474487 -0.40824829]
 [ 0.          0.          1.15470054]]


Scipy R:
 [[ 1.41421356  0.70710678  0.70710678]
 [ 0.         -1.22474487 -0.40824829]
 [ 0.          0.          1.15470054]]
```

The above outcomes give us the good news that our homemade function agrees with what scipy produces.

Now let's do a QR decomposition for a rectangular matrix A that is $n \times m$ with $m > n$.

```
A = np.array([[1, 3, 4], [2, 0, 9]])
```

```
Q, R = adjust_sign(*QR_Decomposition(A))
Q, R
```

```
(array([[ 0.4472136 , -0.89442719],
        [ 0.89442719,  0.4472136 ]]),
 array([[ 2.23606798,  1.34164079,  9.8386991 ],
        [ 0.        , -2.68328157,  0.4472136 ]]))
```

```
Q_scipy, R_scipy = adjust_sign(*qr(A))
Q_scipy, R_scipy
```

```
(array([[ 0.4472136 , -0.89442719],
        [ 0.89442719,  0.4472136 ]]),
 array([[ 2.23606798,  1.34164079,  9.8386991 ],
        [ 0.        , -2.68328157,  0.4472136 ]]))
```

2.6 Using QR Decomposition to Compute Eigenvalues

Now for a useful fact about the QR algorithm.

The following iterations on the QR decomposition can be used to compute **eigenvalues** of a **square** matrix A.

Here is the algorithm:

1. Set $A_0 = A$ and form $A_0 = Q_0 R_0$
2. Form $A_1 = R_0 Q_0$. Note that A_1 is similar to A_0 (easy to verify) and so has the same eigenvalues.
3. Form $A_1 = Q_1 R_1$ (i.e., form the QR decomposition of A_1).
4. Form $A_2 = R_1 Q_1$ and then $A_2 = Q_2 R_2$.
5. Iterate to convergence.
6. Compute eigenvalues of A and compare them to the diagonal values of the limiting A_n found from this process.

Remark: this algorithm is close to one of the most efficient ways of computing eigenvalues!

Let's write some Python code to try out the algorithm

```
def QR_eigvals(A, tol=1e-12, maxiter=1000):
    "Find the eigenvalues of A using QR decomposition."

    A_old = np.copy(A)
    A_new = np.copy(A)

    diff = np.inf
    i = 0
    while (diff > tol) and (i < maxiter):
        A_old[:, :] = A_new
        Q, R = QR_Decomposition(A_old)

        A_new[:, :] = R @ Q

        diff = np.abs(A_new - A_old).max()
        i += 1

    eigvals = np.diag(A_new)

    return eigvals
```

Now let's try the code and compare the results with what `scipy.linalg.eigvals` gives us

Here goes

```
# experiment this with one random A matrix
A = np.random.random((3, 3))
```

```
sorted(QR_eigvals(A))
```

```
[-0.4297694064697409, -0.26958335762838337, 1.5689115427669107]
```

Compare with the `scipy` package.

```
sorted(np.linalg.eigvals(A))
```

```
[-0.4297694064697404, -0.2695833576283847, 1.568911542766912]
```

2.7 QR and PCA

There are interesting connections between the QR decomposition and principal components analysis (PCA).

Here are some.

1. Let X' be a $k \times n$ random matrix where the jth column is a random draw from $\mathcal{N}(\mu, \Sigma)$ where μ is $k \times 1$ vector of means and Σ is a $k \times k$ covariance matrix. We want $n >> k$ – this is an "econometrics example".
2. Form $X' = QR$ where Q is $k \times k$ and R is $k \times n$.
3. Form the eigenvalues of RR', i.e., we'll compute $RR' = \tilde{P}\Lambda\tilde{P}'$.
4. Form $X'X = Q\tilde{P}\Lambda\tilde{P}'Q'$ and compare it with the eigen decomposition $X'X = P\hat{\Lambda}P'$.
5. It will turn out that that $\Lambda = \hat{\Lambda}$ and that $P = Q\tilde{P}$.

Let's verify conjecture 5 with some Python code.

Start by simulating a random (n, k) matrix X.

```
k = 5
n = 1000

# generate some random moments
□ = np.random.random(size=k)
C = np.random.random((k, k))
Σ = C.T @ C
```

```
# X is random matrix where each column follows multivariate normal dist.
X = np.random.multivariate_normal(□, Σ, size=n)
```

```
X.shape
```

```
(1000, 5)
```

Let's apply the QR decomposition to X'.

```
Q, R = adjust_sign(*QR_Decomposition(X.T))
```

Check the shapes of Q and R.

```
Q.shape, R.shape
```

```
((5, 5), (5, 1000))
```

Now we can construct $RR' = \tilde{P}\Lambda\tilde{P}'$ and form an eigen decomposition.

```
RR = R @ R.T

⍰, P_tilde = np.linalg.eigh(RR)
Λ = np.diag(⍰)
```

We can also apply the decomposition to $X'X = P\hat{\Lambda}P'$.

```
XX = X.T @ X

⍰_hat, P = np.linalg.eigh(XX)
Λ_hat = np.diag(⍰_hat)
```

Compare the eigenvalues that are on the diagonals of Λ and $\hat{\Lambda}$.

```
⍰, ⍰_hat
```

```
(array([  13.69522485,  453.32701014,  576.93233248,  754.61505888,
        8872.61109916]),
 array([  13.69522485,  453.32701014,  576.93233248,  754.61505888,
        8872.61109916]))
```

Let's compare P and $Q\tilde{P}$.

Again we need to be careful about sign differences between the columns of P and $Q\tilde{P}$.

```
QP_tilde = Q @ P_tilde

np.abs(P @ diag_sign(P) - QP_tilde @ diag_sign(QP_tilde)).max()
```

```
6.8833827526759706e-15
```

Let's verify that $X'X$ can be decomposed as $Q\tilde{P}\Lambda\tilde{P}'Q'$.

```
QPΛPQ = Q @ P_tilde @ Λ @ P_tilde.T @ Q.T
```

```
np.abs(QPΛPQ - XX).max()
```

```
5.6843418860808015e-12
```

CHAPTER

THREE

CIRCULANT MATRICES

3.1 Overview

This lecture describes circulant matrices and some of their properties.

Circulant matrices have a special structure that connects them to useful concepts including

- convolution
- Fourier transforms
- permutation matrices

Because of these connections, circulant matrices are widely used in machine learning, for example, in image processing.

We begin by importing some Python packages

```python
import numpy as np
from numba import njit
import matplotlib.pyplot as plt
```

```python
np.set_printoptions(precision=3, suppress=True)
```

3.2 Constructing a Circulant Matrix

To construct an $N \times N$ circulant matrix, we need only the first row, say,

$$\begin{bmatrix} c_0 & c_1 & c_2 & c_3 & c_4 & \cdots & c_{N-1} \end{bmatrix}.$$

After setting entries in the first row, the remaining rows of a circulant matrix are determined as follows:

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 & c_4 & \cdots & c_{N-1} \\ c_{N-1} & c_0 & c_1 & c_2 & c_3 & \cdots & c_{N-2} \\ c_{N-2} & c_{N-1} & c_0 & c_1 & c_2 & \cdots & c_{N-3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_3 & c_4 & c_5 & c_6 & c_7 & \cdots & c_2 \\ c_2 & c_3 & c_4 & c_5 & c_6 & \cdots & c_1 \\ c_1 & c_2 & c_3 & c_4 & c_5 & \cdots & c_0 \end{bmatrix} \tag{3.1}$$

It is also possible to construct a circulant matrix by creating the transpose of the above matrix, in which case only the first column needs to be specified.

Let's write some Python code to generate a circulant matrix.

```
@njit
def construct_cirlulant(row):

    N = row.size

    C = np.empty((N, N))

    for i in range(N):

        C[i, i:] = row[:N-i]
        C[i, :i] = row[N-i:]

    return C
```

```
# a simple case when N = 3
construct_cirlulant(np.array([1., 2., 3.]))
```

```
array([[1., 2., 3.],
       [3., 1., 2.],
       [2., 3., 1.]])
```

3.2.1 Some Properties of Circulant Matrices

Here are some useful properties:

Suppose that A and B are both circulant matrices. Then it can be verified that

- The transpose of a circulant matrix is a circulant matrix.
- $A + B$ is a circulant matrix
- AB is a circulant matrix
- $AB = BA$

Now consider a circulant matrix with first row

$$c = \begin{bmatrix} c_0 & c_1 & \cdots & c_{N-1} \end{bmatrix}$$

and consider a vector

$$a = \begin{bmatrix} a_0 & a_1 & \cdots & a_{N-1} \end{bmatrix}$$

The **convolution** of vectors c and a is defined as the vector $b = c * a$ with components

$$b_k = \sum_{i=0}^{n-1} c_{k-i} a_i \tag{3.2}$$

We use $*$ to denote **convolution** via the calculation described in equation (3.2).

It can be verified that the vector b satisfies

$$b = C^T a$$

where C^T is the transpose of the circulant matrix defined in equation (3.1).

3.3 Connection to Permutation Matrix

A good way to construct a circulant matrix is to use a **permutation matrix**.

Before defining a permutation **matrix**, we'll define a **permutation**.

A **permutation** of a set of the set of non-negative integers $\{0, 1, 2, \ldots\}$ is a one-to-one mapping of the set into itself.

A permutation of a set $\{1, 2, \ldots, n\}$ rearranges the n integers in the set.

A permutation matrix is obtained by permuting the rows of an $n \times n$ identity matrix according to a permutation of the numbers 1 to n.

Thus, every row and every column contain precisely a single 1 with 0 everywhere else.

Every permutation corresponds to a unique permutation matrix.

For example, the $N \times N$ matrix

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.3}$$

serves as a **cyclic shift** operator that, when applied to an $N \times 1$ vector h, shifts entries in rows 2 through N up one row and shifts the entry in row 1 to row N.

Eigenvalues of the cyclic shift permutation matrix P defined in equation (3.3) can be computed by constructing

$$P - \lambda I = \begin{bmatrix} -\lambda & 1 & 0 & 0 & \cdots & 0 \\ 0 & -\lambda & 1 & 0 & \cdots & 0 \\ 0 & 0 & -\lambda & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & 0 & \cdots & -\lambda \end{bmatrix}$$

and solving

$$\det(P - \lambda I) = (-1)^N \lambda^N - 1 = 0$$

Eigenvalues λ_i can be complex.

Magnitudes $\mid \lambda_i \mid$ of these eigenvalues λ_i all equal 1.

Thus, **singular values** of the permutation matrix P defined in equation (3.3) all equal 1.

It can be verified that permutation matrices are orthogonal matrices:

$$PP' = I$$

3.4 Examples with Python

Let's write some Python code to illustrate these ideas.

```
@njit
def construct_P(N):

    P = np.zeros((N, N))

    for i in range(N-1):
        P[i, i+1] = 1
    P[-1, 0] = 1

    return P
```

```
P4 = construct_P(4)
P4
```

```
array([[0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [1., 0., 0., 0.]])
```

```
# compute the eigenvalues and eigenvectors
□, Q = np.linalg.eig(P4)
```

```
for i in range(4):
    print(f'□{i} = {□[i]:.1f} \nvec{i} = {Q[i, :]}\n')
```

```
□0 = -1.0+0.0j
vec0 = [-0.5+0.j  0.5+0.j  0.5-0.j -0.5+0.j]

□1 = -0.0+1.0j
vec1 = [ 0.5+0.j  -0. +0.5j -0. -0.5j -0.5+0.j ]

□2 = -0.0-1.0j
vec2 = [-0.5+0.j -0.5-0.j -0.5+0.j -0.5+0.j]

□3 = 1.0+0.0j
vec3 = [ 0.5+0.j   0. -0.5j  0. +0.5j -0.5+0.j ]
```

In graphs below, we shall portray eigenvalues of a shift permutation matrix in the complex plane.

These eigenvalues are uniformly distributed along the unit circle.

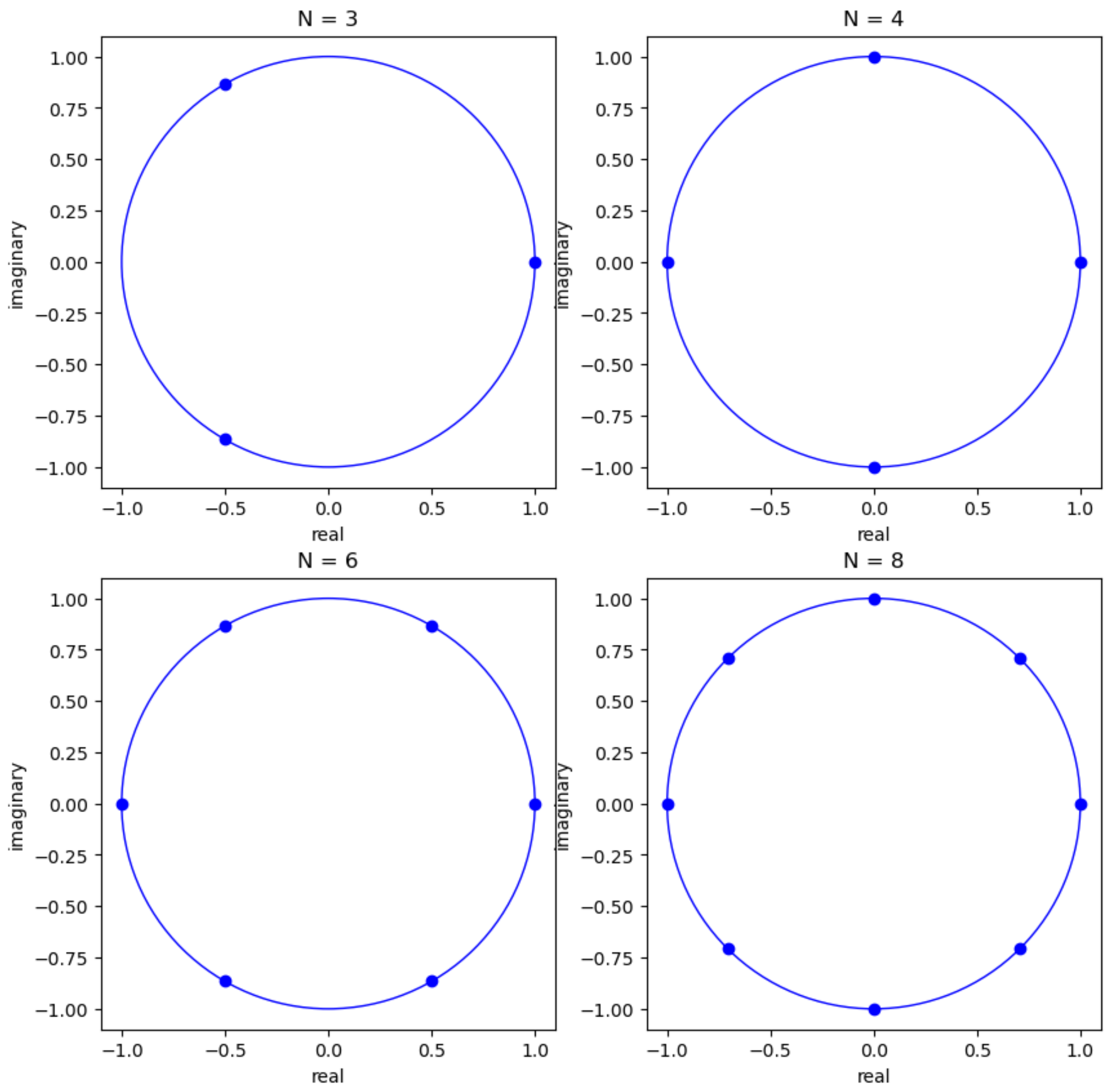
They are the n **roots of unity**, meaning they are the n numbers z that solve $z^n = 1$, where z is a complex number.

In particular, the n roots of unity are

$$z = \exp\left(\frac{2\pi jk}{N}\right), \quad k = 0, \ldots, N-1$$

where j denotes the purely imaginary unit number.

```python
fig, ax = plt.subplots(2, 2, figsize=(10, 10))

for i, N in enumerate([3, 4, 6, 8]):

    row_i = i // 2
    col_i = i % 2

    P = construct_P(N)
    □, Q = np.linalg.eig(P)

    circ = plt.Circle((0, 0), radius=1, edgecolor='b', facecolor='None')
    ax[row_i, col_i].add_patch(circ)

    for j in range(N):
        ax[row_i, col_i].scatter(□[j].real, □[j].imag, c='b')

    ax[row_i, col_i].set_title(f'N = {N}')
    ax[row_i, col_i].set_xlabel('real')
    ax[row_i, col_i].set_ylabel('imaginary')

plt.show()
```



For a vector of coefficients $\{c_i\}_{i=0}^{n-1}$, eigenvectors of P are also eigenvectors of

$$C = c_0 I + c_1 P + c_2 P^2 + \cdots + c_{N-1} P^{N-1}.$$

Consider an example in which $N = 8$ and let $w = e^{-2\pi j/N}$.

It can be verified that the matrix F_8 of eigenvectors of P_8 is

$$F_8 = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^7 \\ 1 & w^2 & w^4 & \cdots & w^{14} \\ 1 & w^3 & w^6 & \cdots & w^{21} \\ 1 & w^4 & w^8 & \cdots & w^{28} \\ 1 & w^5 & w^{10} & \cdots & w^{35} \\ 1 & w^6 & w^{12} & \cdots & w^{42} \\ 1 & w^7 & w^{14} & \cdots & w^{49} \end{bmatrix}$$

The matrix F_8 defines a Discete Fourier Transform.

To convert it into an orthogonal eigenvector matrix, we can simply normalize it by dividing every entry by $\sqrt{8}$.

- stare at the first column of F_8 above to convince yourself of this fact

The eigenvalues corresponding to each eigenvector are $\{w^j\}_{j=0}^7$ in order.

```
def construct_F(N):

    w = np.e ** (-complex(0, 2*np.pi/N))

    F = np.ones((N, N), dtype=complex)
    for i in range(1, N):
        F[i, 1:] = w ** (i * np.arange(1, N))

    return F, w
```

```
F8, w = construct_F(8)
```

```
w
```

```
(0.7071067811865476-0.7071067811865475j)
```

```
F8
```

```
array([[ 1.   +0.j   ,  1.   +0.j   ,  1.   +0.j   ,  1.   +0.j   ,
         1.   +0.j   ,  1.   +0.j   ,  1.   +0.j   ,  1.   +0.j   ],
       [ 1.   +0.j   ,  0.707-0.707j,  0.   -1.j   , -0.707-0.707j,
        -1.   -0.j   , -0.707+0.707j, -0.   +1.j   ,  0.707+0.707j],
       [ 1.   +0.j   ,  0.   -1.j   , -1.   -0.j   , -0.   +1.j   ,
         1.   +0.j   ,  0.   -1.j   , -1.   -0.j   , -0.   +1.j   ],
       [ 1.   +0.j   , -0.707-0.707j, -0.   +1.j   ,  0.707-0.707j,
        -1.   -0.j   ,  0.707+0.707j,  0.   -1.j   , -0.707+0.707j],
       [ 1.   +0.j   , -1.   -0.j   ,  1.   +0.j   , -1.   -0.j   ,
         1.   +0.j   , -1.   -0.j   ,  1.   +0.j   , -1.   -0.j   ],
       [ 1.   +0.j   , -0.707+0.707j,  0.   -1.j   ,  0.707+0.707j,
        -1.   -0.j   ,  0.707-0.707j, -0.   +1.j   , -0.707-0.707j],
       [ 1.   +0.j   , -0.   +1.j   , -1.   -0.j   ,  0.   -1.j   ,
         1.   +0.j   , -0.   +1.j   , -1.   -0.j   ,  0.   -1.j   ],
       [ 1.   +0.j   ,  0.707+0.707j, -0.   +1.j   , -0.707+0.707j,
        -1.   -0.j   , -0.707-0.707j,  0.   -1.j   ,  0.707-0.707j]])
```

```
# normalize
Q8 = F8 / np.sqrt(8)
```

```
# verify the orthogonality (unitarity)
Q8 @ np.conjugate(Q8)
```

```
array([[ 1.+0.j, -0.+0.j, -0.+0.j, -0.+0.j,  0.+0.j,  0.+0.j,  0.+0.j,
         0.+0.j],
       [-0.-0.j,  1.+0.j, -0.+0.j, -0.+0.j, -0.+0.j,  0.+0.j,  0.+0.j,
         0.+0.j],
       [-0.-0.j, -0.-0.j,  1.+0.j, -0.+0.j, -0.+0.j, -0.+0.j,  0.+0.j,
```

(continues on next page)

(continued from previous page)

```
          0.+0.j],
        [-0.-0.j, -0.-0.j, -0.-0.j,  1.+0.j, -0.+0.j, -0.+0.j, -0.+0.j,
         -0.+0.j],
        [ 0.-0.j, -0.-0.j, -0.-0.j, -0.+0.j,  1.+0.j, -0.+0.j, -0.+0.j,
         -0.+0.j],
        [ 0.-0.j,  0.-0.j, -0.-0.j, -0.-0.j, -0.-0.j,  1.+0.j, -0.+0.j,
         -0.+0.j],
        [ 0.-0.j,  0.-0.j,  0.-0.j, -0.-0.j, -0.-0.j, -0.-0.j,  1.+0.j,
         -0.+0.j],
        [ 0.-0.j,  0.-0.j,  0.-0.j, -0.-0.j, -0.-0.j, -0.-0.j, -0.-0.j,
          1.+0.j]])
```

Let's verify that kth column of Q_8 is an eigenvector of P_8 with an eigenvalue w^k.

```
P8 = construct_P(8)
```

```
diff_arr = np.empty(8, dtype=complex)
for j in range(8):
    diff = P8 @ Q8[:, j] - w ** j * Q8[:, j]
    diff_arr[j] = diff @ diff.T
```

```
diff_arr
```

```
array([ 0.+0.j, -0.+0.j, -0.+0.j, -0.+0.j, -0.+0.j, -0.+0.j, -0.+0.j,
       -0.+0.j])
```

3.5 Associated Permutation Matrix

Next, we execute calculations to verify that the circulant matrix C defined in equation (3.1) can be written as

$$C = c_0 I + c_1 P + \cdots + c_{n-1} P^{n-1}$$

and that every eigenvector of P is also an eigenvector of C.

We illustrate this for $N = 8$ case.

```
c = np.random.random(8)
```

```
c
```

```
array([0.985, 0.504, 0.387, 0.779, 0.839, 0.394, 0.461, 0.776])
```

```
C8 = construct_cirlulant(c)
```

Compute $c_0 I + c_1 P + \cdots + c_{n-1} P^{n-1}$.

```
N = 8

C = np.zeros((N, N))
P = np.eye(N)

for i in range(N):
    C += c[i] * P
    P = P8 @ P
```

```
C
```

```
array([[0.985, 0.504, 0.387, 0.779, 0.839, 0.394, 0.461, 0.776],
       [0.776, 0.985, 0.504, 0.387, 0.779, 0.839, 0.394, 0.461],
       [0.461, 0.776, 0.985, 0.504, 0.387, 0.779, 0.839, 0.394],
       [0.394, 0.461, 0.776, 0.985, 0.504, 0.387, 0.779, 0.839],
       [0.839, 0.394, 0.461, 0.776, 0.985, 0.504, 0.387, 0.779],
       [0.779, 0.839, 0.394, 0.461, 0.776, 0.985, 0.504, 0.387],
       [0.387, 0.779, 0.839, 0.394, 0.461, 0.776, 0.985, 0.504],
       [0.504, 0.387, 0.779, 0.839, 0.394, 0.461, 0.776, 0.985]])
```

```
C8
```

```
array([[0.985, 0.504, 0.387, 0.779, 0.839, 0.394, 0.461, 0.776],
       [0.776, 0.985, 0.504, 0.387, 0.779, 0.839, 0.394, 0.461],
       [0.461, 0.776, 0.985, 0.504, 0.387, 0.779, 0.839, 0.394],
       [0.394, 0.461, 0.776, 0.985, 0.504, 0.387, 0.779, 0.839],
       [0.839, 0.394, 0.461, 0.776, 0.985, 0.504, 0.387, 0.779],
       [0.779, 0.839, 0.394, 0.461, 0.776, 0.985, 0.504, 0.387],
       [0.387, 0.779, 0.839, 0.394, 0.461, 0.776, 0.985, 0.504],
       [0.504, 0.387, 0.779, 0.839, 0.394, 0.461, 0.776, 0.985]])
```

Now let's compute the difference between two circulant matrices that we have constructed in two different ways.

```
np.abs(C - C8).max()
```

```
0.0
```

The kth column of P_8 associated with eigenvalue w^{k-1} is an eigenvector of C_8 associated with an eigenvalue $\sum_{h=0}^{7} c_j w^{hk}$.

```
⍰_C8 = np.zeros(8, dtype=complex)

for j in range(8):
    for k in range(8):
        ⍰_C8[j] += c[k] * w ** (j * k)
```

```
⍰_C8
```

```
array([5.125+0.j   , 0.222-0.006j, 0.976+0.656j, 0.07 -0.155j,
       0.22 -0.j   , 0.07 +0.155j, 0.976-0.656j, 0.222+0.006j])
```

We can verify this by comparing `C8 @ Q8[:, j]` with `⍰_C8[j] * Q8[:, j]`.

```
# verify
for j in range(8):
    diff = C8 @ Q8[:, j] - ⍰_C8[j] * Q8[:, j]
    print(diff)
```

```
[0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]
[ 0.+0.j  0.-0.j -0.-0.j -0.-0.j -0.-0.j -0.-0.j -0.+0.j -0.+0.j]
[ 0.-0.j -0.-0.j -0.-0.j -0.+0.j  0.-0.j -0.-0.j -0.+0.j  0.+0.j]
[-0.+0.j -0.-0.j -0.+0.j  0.-0.j -0.-0.j -0.+0.j  0.-0.j -0.-0.j]
[ 0.+0.j -0.-0.j  0.+0.j -0.-0.j  0.-0.j -0.-0.j  0.+0.j -0.-0.j]
[ 0.+0.j -0.-0.j  0.+0.j -0.-0.j  0.-0.j  0.+0.j -0.-0.j  0.-0.j]
[ 0.+0.j -0.-0.j  0.-0.j  0.+0.j -0.-0.j  0.-0.j  0.-0.j  0.+0.j]
[-0.+0.j  0.-0.j  0.-0.j  0.-0.j  0.-0.j  0.-0.j  0.+0.j  0.+0.j]
```

3.6 Discrete Fourier Transform

The **Discrete Fourier Transform** (DFT) allows us to represent a discrete time sequence as a weighted sum of complex sinusoids.

Consider a sequence of N real number $\{x_j\}_{j=0}^{N-1}$.

The **Discrete Fourier Transform** maps $\{x_j\}_{j=0}^{N-1}$ into a sequence of complex numbers $\{X_k\}_{k=0}^{N-1}$

where

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi \frac{kn}{N} i}$$

```
def DFT(x):
    "The discrete Fourier transform."

    N = len(x)
    w = np.e ** (-complex(0, 2*np.pi/N))

    X = np.zeros(N, dtype=complex)
    for k in range(N):
        for n in range(N):
            X[k] += x[n] * w ** (k * n)

    return X
```

Consider the following example.

$$x_n = \begin{cases} 1/2 & n = 0, 1 \\ 0 & \text{otherwise} \end{cases}$$

```
x = np.zeros(10)
x[0:2] = 1/2
```

```
x
```

```
array([0.5, 0.5, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ])
```

Apply a discrete Fourier transform.

```
X = DFT(x)
```

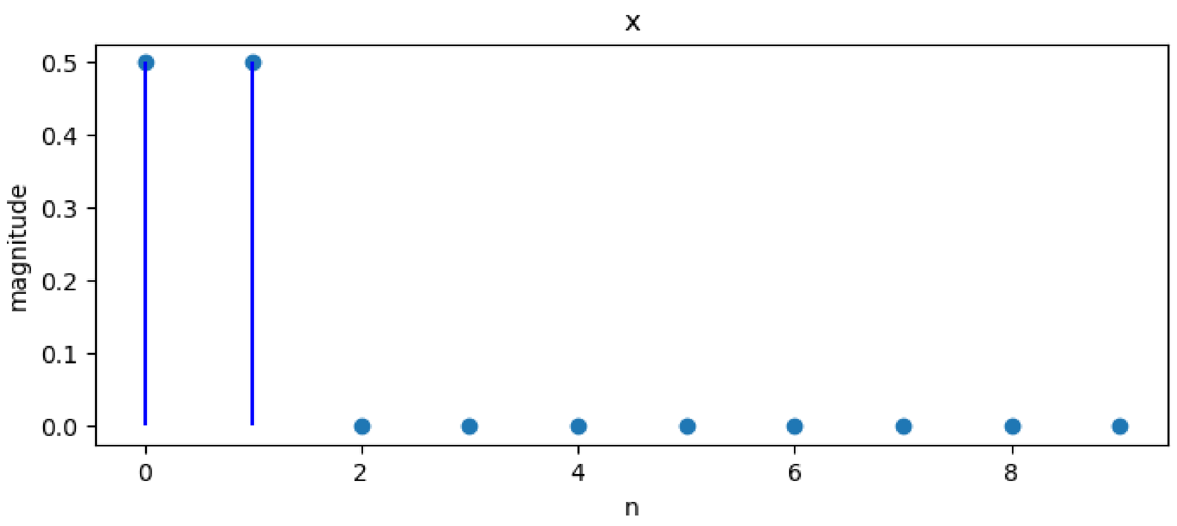
```
X
```

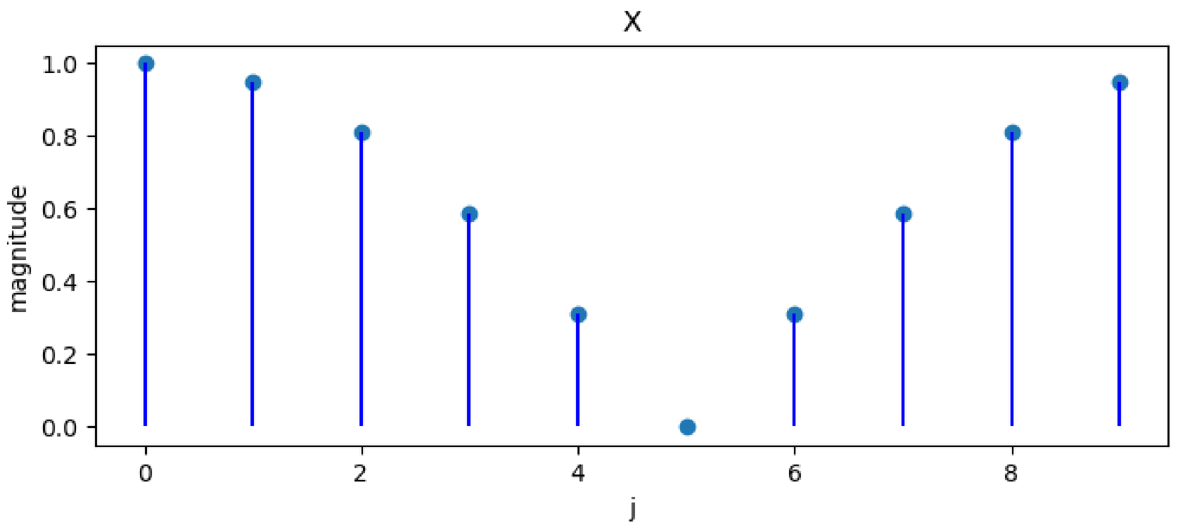
```
array([ 1.   +0.j   ,  0.905-0.294j,  0.655-0.476j,  0.345-0.476j,
        0.095-0.294j, -0.   +0.j   ,  0.095+0.294j,  0.345+0.476j,
        0.655+0.476j,  0.905+0.294j])
```

We can plot magnitudes of a sequence of numbers and the associated discrete Fourier transform.

```
def plot_magnitude(x=None, X=None):

    data = []
    names = []
    xs = []
    if (x is not None):
        data.append(x)
        names.append('x')
        xs.append('n')
    if (X is not None):
        data.append(X)
        names.append('X')
        xs.append('j')

    num = len(data)
    for i in range(num):
        n = data[i].size
        plt.figure(figsize=(8, 3))
        plt.scatter(range(n), np.abs(data[i]))
        plt.vlines(range(n), 0, np.abs(data[i]), color='b')

        plt.xlabel(xs[i])
        plt.ylabel('magnitude')
        plt.title(names[i])
        plt.show()
```

```
plot_magnitude(x=x, X=X)
```





The **inverse Fourier transform** transforms a Fourier transform X of x back to x.

The inverse Fourier transform is defined as

$$x_n = \sum_{k=0}^{N-1} \frac{1}{N} X_k e^{2\pi(\frac{kn}{N})i}, \quad n = 0, 1, \ldots, N-1$$

```
def inverse_transform(X):

    N = len(X)
    w = np.e ** (complex(0, 2*np.pi/N))

    x = np.zeros(N, dtype=complex)
    for n in range(N):
        for k in range(N):
            x[n] += X[k] * w ** (k * n) / N

    return x
```

```
inverse_transform(X)
```

```
array([ 0.5+0.j,  0.5-0.j, -0. -0.j, -0. -0.j, -0. -0.j, -0. -0.j,
       -0. +0.j, -0. +0.j, -0. +0.j, -0. +0.j])
```

Another example is

$$x_n = 2\cos\left(2\pi\frac{11}{40}n\right),\ n = 0, 1, 2, \cdots 19$$

Since $N = 20$, we cannot use an integer multiple of $\frac{1}{20}$ to represent a frequency $\frac{11}{40}$.

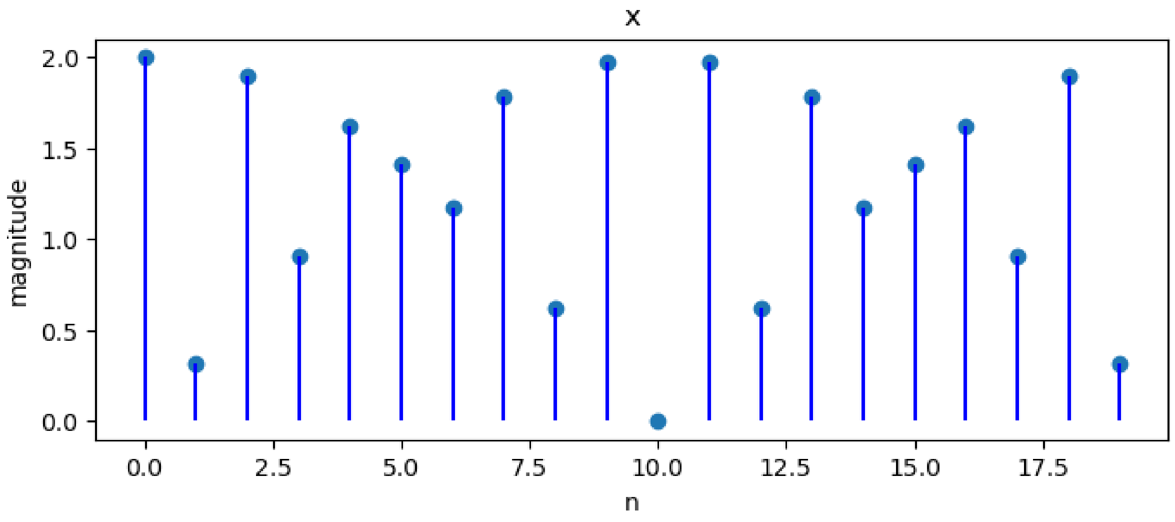
To handle this, we shall end up using all N of the availble frequencies in the DFT.

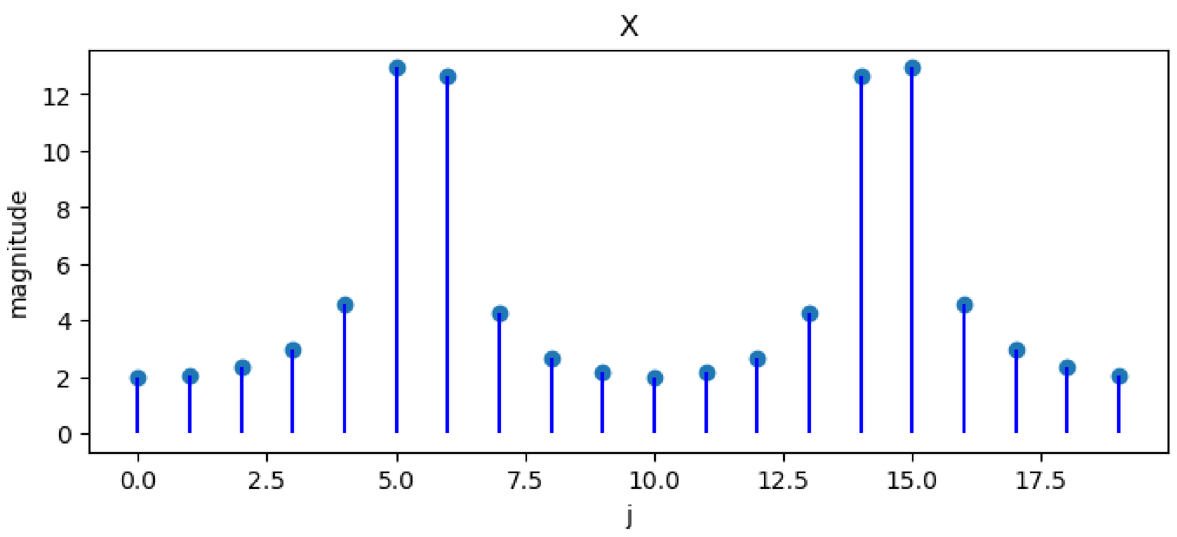
Since $\frac{11}{40}$ is in between $\frac{10}{40}$ and $\frac{12}{40}$ (each of which is an integer multiple of $\frac{1}{20}$), the complex coefficients in the DFT have their largest magnitudes at $k = 5, 6, 15, 16$, not just at a single frequency.

```
N = 20
x = np.empty(N)

for j in range(N):
    x[j] = 2 * np.cos(2 * np.pi * 11 * j / 40)
```

```
X = DFT(x)
```

```
plot_magnitude(x=x, X=X)
```





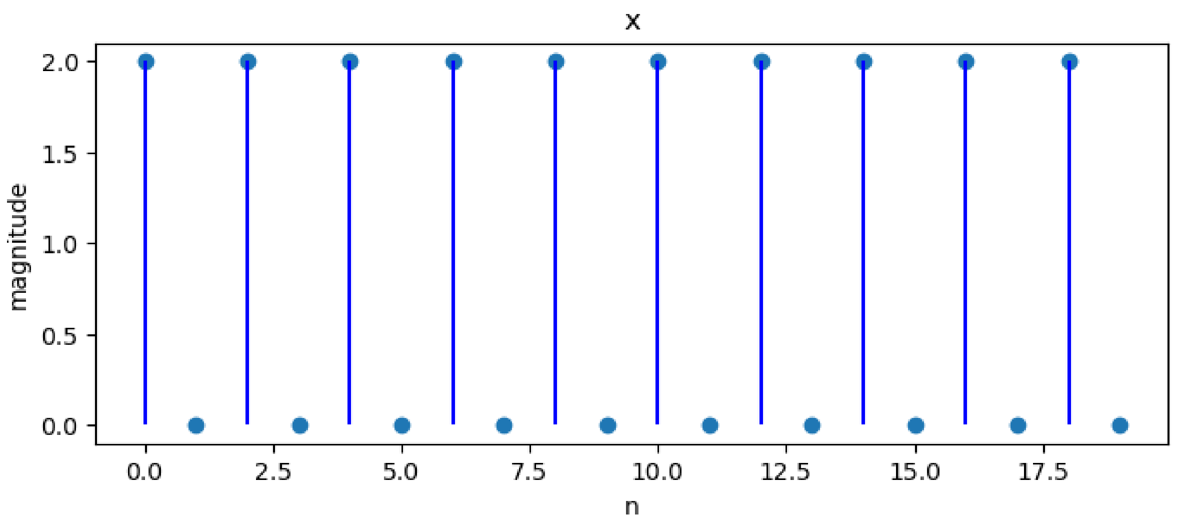
What happens if we change the last example to $x_n = 2\cos\left(2\pi\frac{10}{40}n\right)$?

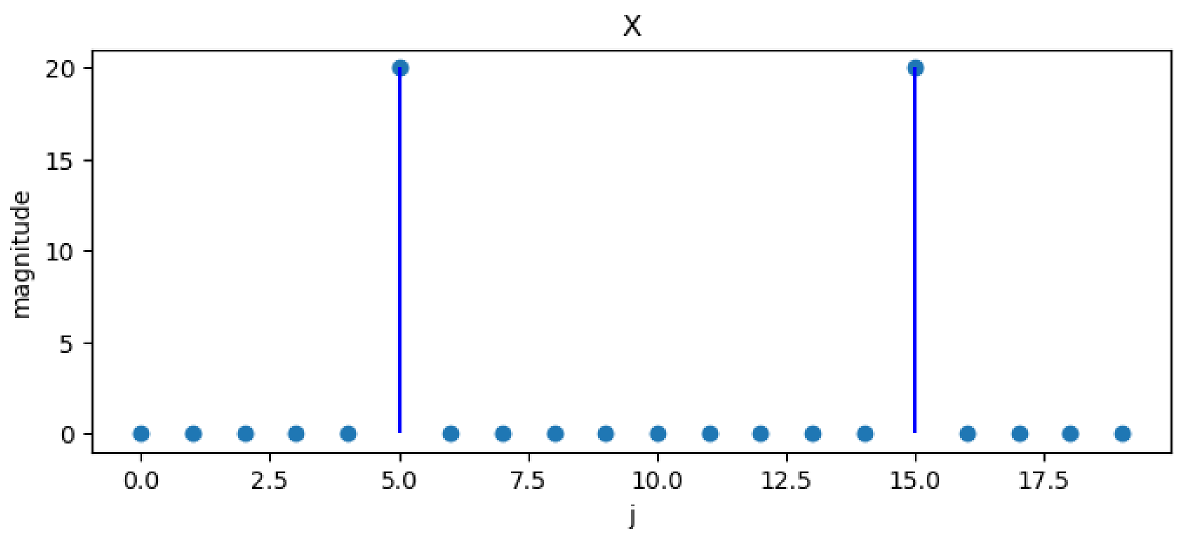
Note that $\frac{10}{40}$ is an integer multiple of $\frac{1}{20}$.

```
N = 20
x = np.empty(N)

for j in range(N):
    x[j] = 2 * np.cos(2 * np.pi * 10 * j / 40)
```

```
X = DFT(x)
```

```
plot_magnitude(x=x, X=X)
```





If we represent the discrete Fourier transform as a matrix, we discover that it equals the matrix F_N of eigenvectors of the permutation matrix P_N.

We can use the example where $x_n = 2\cos\left(2\pi\frac{11}{40}n\right),\ n = 0, 1, 2, \cdots 19$ to illustrate this.

```
N = 20
x = np.empty(N)

for j in range(N):
    x[j] = 2 * np.cos(2 * np.pi * 11 * j / 40)
```

```
x
```

```
array([ 2.   , -0.313, -1.902,  0.908,  1.618, -1.414, -1.176,  1.782,
        0.618, -1.975, -0.   ,  1.975, -0.618, -1.782,  1.176,  1.414,
       -1.618, -0.908,  1.902,  0.313])
```

First use the summation formula to transform x to X.

```
X = DFT(x)
X
```

```
array([2. +0.j   , 2. +0.558j, 2. +1.218j, 2. +2.174j, 2. +4.087j,
       2.+12.785j, 2.-12.466j, 2. -3.751j, 2. -1.801j, 2. -0.778j,
       2. -0.j   , 2. +0.778j, 2. +1.801j, 2. +3.751j, 2.+12.466j,
       2.-12.785j, 2. -4.087j, 2. -2.174j, 2. -1.218j, 2. -0.558j])
```

Now let's evaluate the outcome of postmultiplying the eigenvector matrix F_{20} by the vector x, a product that we claim should equal the Fourier tranform of the sequence $\{x_n\}_{n=0}^{N-1}$.

```
F20, _ = construct_F(20)
```

```
F20 @ x
```

```
array([2. +0.j   , 2. +0.558j, 2. +1.218j, 2. +2.174j, 2. +4.087j,
       2.+12.785j, 2.-12.466j, 2. -3.751j, 2. -1.801j, 2. -0.778j,
       2. -0.j   , 2. +0.778j, 2. +1.801j, 2. +3.751j, 2.+12.466j,
       2.-12.785j, 2. -4.087j, 2. -2.174j, 2. -1.218j, 2. -0.558j])
```

Similarly, the inverse DFT can be expressed as a inverse DFT matrix F_{20}^{-1}.

```
F20_inv = np.linalg.inv(F20)
F20_inv @ X
```

```
array([ 2.   -0.j, -0.313+0.j, -1.902-0.j,  0.908-0.j,  1.618-0.j,
       -1.414-0.j, -1.176+0.j,  1.782-0.j,  0.618-0.j, -1.975-0.j,
       -0.   +0.j,  1.975-0.j, -0.618-0.j, -1.782+0.j,  1.176+0.j,
        1.414-0.j, -1.618-0.j, -0.908+0.j,  1.902+0.j,  0.313-0.j])
```

CHAPTER

FOUR

SINGULAR VALUE DECOMPOSITION (SVD)

4.1 Overview

The **singular value decomposition** (SVD) is a work-horse in applications of least squares projection that form foundations for many statistical and machine learning methods.

After defining the SVD, we'll describe how it connects to

- **four fundamental spaces** of linear algebra
- under-determined and over-determined **least squares regressions**
- **principal components analysis** (PCA)

Like principal components analysis (PCA), DMD can be thought of as a data-reduction procedure that represents salient patterns by projecting data onto a limited set of factors.

In a sequel to this lecture about *Dynamic Mode Decompositions*, we'll describe how SVD's provide ways rapidly to compute reduced-order approximations to first-order Vector Autoregressions (VARs).

4.2 The Setting

Let X be an $m \times n$ matrix of rank p.

Necessarily, $p \leq \min(m, n)$.

In much of this lecture, we'll think of X as a matrix of **data** in which

- each column is an **individual** – a time period or person, depending on the application
- each row is a **random variable** describing an attribute of a time period or a person, depending on the application

We'll be interested in two situations

- A **short and fat** case in which $m << n$, so that there are many more columns (individuals) than rows (attributes).
- A **tall and skinny** case in which $m >> n$, so that there are many more rows (attributes) than columns (individuals).

We'll apply a **singular value decomposition** of X in both situations.

In the $m << n$ case in which there are many more individuals n than attributes m, we can calculate sample moments of a joint distribution by taking averages across observations of functions of the observations.

In this $m << n$ case, we'll look for **patterns** by using a **singular value decomposition** to do a **principal components analysis** (PCA).

In the $m >> n$ case in which there are many more attributes m than individuals n and when we are in a time-series setting in which n equals the number of time periods covered in the data set X, we'll proceed in a different way.

We'll again use a **singular value decomposition**, but now to construct a **dynamic mode decomposition** (DMD)

4.3 Singular Value Decomposition

A **singular value decomposition** of an $m \times n$ matrix X of rank $p \leq \min(m, n)$ is

$$X = U\Sigma V^\top \tag{4.1}$$

where

$$\begin{aligned} UU^\top &= I \qquad & U^\top U &= I \\ VV^\top &= I \qquad & V^\top V &= I \end{aligned}$$

and

- U is an $m \times m$ orthogonal matrix of **left singular vectors** of X
- Columns of U are eigenvectors of $XX^\top$
- V is an $n \times n$ orthogonal matrix of **right singular vectors** of X
- Columns of V are eigenvectors of $X^\top X$
- Σ is an $m \times n$ matrix in which the first p places on its main diagonal are positive numbers $\sigma_1, \sigma_2, \dots, \sigma_p$ called **singular values**; remaining entries of Σ are all zero
- The p singular values are positive square roots of the eigenvalues of the $m \times m$ matrix $XX^\top$ and also of the $n \times n$ matrix $X^\top X$
- We adopt a convention that when U is a complex valued matrix, $U^\top$ denotes the **conjugate-transpose** or **Hermitian-transpose** of U, meaning that $U_{ij}^\top$ is the complex conjugate of U_{ji}.
- Similarly, when V is a complex valued matrix, $V^\top$ denotes the **conjugate-transpose** or **Hermitian-transpose** of V

The matrices U, Σ, V entail linear transformations that reshape in vectors in the following ways:

- multiplying vectors by the unitary matrices U and V **rotates** them, but leaves **angles between vectors** and **lengths of vectors** unchanged.
- multiplying vectors by the diagonal matrix Σ leaves **angles between vectors** unchanged but **rescales** vectors.

Thus, representation (4.1) asserts that multiplying an $n \times 1$ vector y by the $m \times n$ matrix X amounts to performing the following three multiplications of y sequentially:

- **rotating** y by computing $V^\top y$
- **rescaling** $V^\top y$ by multiplying it by Σ
- **rotating** $\Sigma V^\top y$ by multiplying it by U

This structure of the $m \times n$ matrix X opens the door to constructing systems of data **encoders** and **decoders**.

Thus,

- $V^\top y$ is an encoder
- Σ is an operator to be applied to the encoded data
- U is a decoder to be applied to the output from applying operator Σ to the encoded data

We'll apply this circle of ideas later in this lecture when we study Dynamic Mode Decomposition.

Road Ahead

What we have described above is called a **full** SVD.

In a **full** SVD, the shapes of U, Σ, and V are (m, m), (m, n), (n, n), respectively.

Later we'll also describe an **economy** or **reduced** SVD.

Before we study a **reduced** SVD we'll say a little more about properties of a **full** SVD.

4.4 Four Fundamental Subspaces

Let $\mathcal{C}$ denote a column space, $\mathcal{N}$ denote a null space, and $\mathcal{R}$ denote a row space.

Let's start by recalling the four fundamental subspaces of an $m \times n$ matrix X of rank p.

- The **column space** of X, denoted $\mathcal{C}(X)$, is the span of the columns of X, i.e., all vectors y that can be written as linear combinations of columns of X. Its dimension is p.
- The **null space** of X, denoted $\mathcal{N}(X)$ consists of all vectors y that satisfy $Xy = 0$. Its dimension is $n - p$.
- The **row space** of X, denoted $\mathcal{R}(X)$ is the column space of $X^\top$. It consists of all vectors z that can be written as linear combinations of rows of X. Its dimension is p.
- The **left null space** of X, denoted $\mathcal{N}(X^\top)$, consist of all vectors z such that $X^\top z = 0$. Its dimension is $m - p$.

For a full SVD of a matrix X, the matrix U of left singular vectors and the matrix V of right singular vectors contain orthogonal bases for all four subspaces.

They form two pairs of orthogonal subspaces that we'll describe now.

Let $u_i, i = 1, \ldots, m$ be the m column vectors of U and let $v_i, i = 1, \ldots, n$ be the n column vectors of V.

Let's write the full SVD of X as

$$X = \begin{bmatrix} U_L & U_R \end{bmatrix} \begin{bmatrix} \Sigma_p & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_L & V_R \end{bmatrix}^\top \tag{4.2}$$

where Σ_p is a $p \times p$ diagonal matrix with the p singular values on the diagonal and

$$\begin{aligned} U_L &= \begin{bmatrix} u_1 & \cdots & u_p \end{bmatrix}, \quad U_R = \begin{bmatrix} u_{p+1} & \cdots u_m \end{bmatrix} \\ V_L &= \begin{bmatrix} v_1 & \cdots & v_p \end{bmatrix}, \quad U_R = \begin{bmatrix} v_{p+1} & \cdots u_n \end{bmatrix} \end{aligned}$$

Representation (4.2) implies that

$$X \begin{bmatrix} V_L & V_R \end{bmatrix} = \begin{bmatrix} U_L & U_R \end{bmatrix} \begin{bmatrix} \Sigma_p & 0 \\ 0 & 0 \end{bmatrix}$$

or

$$\begin{aligned} XV_L &= U_L \Sigma_p \\ XV_R &= 0 \end{aligned} \tag{4.3}$$

or

$$\begin{aligned} Xv_i &= \sigma_i u_i, \quad i = 1, \ldots, p \\ Xv_i &= 0, \quad i = p + 1, \ldots, n \end{aligned} \tag{4.4}$$

Equations (4.4) tell how the transformation X maps a pair of orthonormal vectors v_i, v_j for i and j both less than or equal to the rank p of X into a pair of orthonormal vectors u_i, u_j.

Equations (4.3) assert that

$$\begin{aligned} \mathcal{C}(X) &= \mathcal{C}(U_L) \\ \mathcal{N}(X) &= \mathcal{C}(V_R) \end{aligned}$$

Taking transposes on both sides of representation (4.2) implies

$$X^\top \begin{bmatrix} U_L & U_R \end{bmatrix} = \begin{bmatrix} V_L & V_R \end{bmatrix} \begin{bmatrix} \Sigma_p & 0 \\ 0 & 0 \end{bmatrix}$$

or

$$\begin{aligned} X^\top U_L &= V_L \Sigma_p \\ X^\top U_R &= 0 \end{aligned} \tag{4.5}$$

or

$$\begin{aligned} X^\top u_i &= \sigma_i v_i, \quad i = 1, \dots, p \\ X^\top u_i &= 0 \quad i = p+1, \dots, m \end{aligned} \tag{4.6}$$

Notice how equations (4.6) assert that the transformation $X^\top$ maps a pair of distinct orthonormal vectors u_i, u_j for i and j both less than or equal to the rank p of X into a pair of distinct orthonormal vectors v_i, v_j .

Equations (4.5) assert that

$$\begin{aligned} \mathcal{R}(X) \equiv \mathcal{C}(X^\top) &= \mathcal{C}(V_L) \\ \mathcal{N}(X^\top) &= \mathcal{C}(U_R) \end{aligned}$$

Thus, taken together, the systems of equations (4.3) and (4.5) describe the four fundamental subspaces of X in the following ways:

$$\begin{aligned} \mathcal{C}(X) &= \mathcal{C}(U_L) \\ \mathcal{N}(X^\top) &= \mathcal{C}(U_R) \\ \mathcal{R}(X) \equiv \mathcal{C}(X^\top) &= \mathcal{C}(V_L) \\ \mathcal{N}(X) &= \mathcal{C}(V_R) \end{aligned} \tag{4.7}$$

Since U and V are both orthonormal matrices, collection (4.7) asserts that

- U_L is an orthonormal basis for the column space of X
- U_R is an orthonormal basis for the null space of $X^\top$
- V_L is an orthonormal basis for the row space of X
- V_R is an orthonormal basis for the null space of X

We have verified the four claims in (4.7) simply by performing the multiplications called for by the right side of (4.2) and reading them.

The claims in (4.7) and the fact that U and V are both unitary (i.e, orthonormal) matrices imply that

- the column space of X is orthogonal to the null space of $X^\top$
- the null space of X is orthogonal to the row space of X

Sometimes these properties are described with the following two pairs of orthogonal complement subspaces:

- $\mathcal{C}(X)$ is the orthogonal complement of $\mathcal{N}(X^\top)$
- $\mathcal{R}(X)$ is the orthogonal complement $\mathcal{N}(X)$

Let's do an example.

```
import numpy as np
import numpy.linalg as LA
import matplotlib.pyplot as plt
```

Having imported these modules, let’s do the example.

```
np.set_printoptions(precision=2)

# Define the matrix
A = np.array([[1, 2, 3, 4, 5],
              [2, 3, 4, 5, 6],
              [3, 4, 5, 6, 7],
              [4, 5, 6, 7, 8],
              [5, 6, 7, 8, 9]])

# Compute the SVD of the matrix
U, S, V = np.linalg.svd(A,full_matrices=True)

# Compute the rank of the matrix
rank = np.linalg.matrix_rank(A)

# Print the rank of the matrix
print("Rank of matrix:\n", rank)
print("S: \n", S)

# Compute the four fundamental subspaces
row_space = U[:, :rank]
col_space = V[:, :rank]
null_space = V[:, rank:]
left_null_space = U[:, rank:]


print("U:\n", U)
print("Column space:\n", col_space)
print("Left null space:\n", left_null_space)
print("V.T:\n", V.T)
print("Row space:\n", row_space.T)
print("Right null space:\n", null_space.T)
```

```
Rank of matrix:
 2
S:
 [2.69e+01 1.86e+00 8.62e-16 5.26e-16 2.77e-17]
U:
 [[-0.27 -0.73 -0.53 -0.34  0.03]
 [-0.35 -0.42  0.24  0.8  -0.07]
 [-0.43 -0.11  0.67 -0.41  0.43]
 [-0.51  0.19  0.09 -0.22 -0.8 ]
 [-0.59  0.5  -0.46  0.17  0.4 ]]
Column space:
 [[-0.27 -0.35]
 [ 0.73  0.42]
 [-0.03  0.16]
 [-0.51  0.82]
 [ 0.37  0.06]]
Left null space:
```

(continues on next page)

(continued from previous page)

```
 [[-0.53 -0.34  0.03]
 [ 0.24  0.8  -0.07]
 [ 0.67 -0.41  0.43]
 [ 0.09 -0.22 -0.8 ]
 [-0.46  0.17  0.4 ]]
V.T:
 [[-0.27  0.73 -0.03 -0.51  0.37]
 [-0.35  0.42  0.16  0.82  0.06]
 [-0.43  0.11  0.25 -0.23 -0.83]
 [-0.51 -0.19 -0.84  0.04 -0.02]
 [-0.59 -0.5   0.46 -0.12  0.41]]
Row space:
 [[-0.27 -0.35 -0.43 -0.51 -0.59]
 [-0.73 -0.42 -0.11  0.19  0.5 ]]
Right null space:
 [[-0.43  0.11  0.25 -0.23 -0.83]
 [-0.51 -0.19 -0.84  0.04 -0.02]
 [-0.59 -0.5   0.46 -0.12  0.41]]
```

4.5 Eckart-Young Theorem

Suppose that we want to construct the best rank r approximation of an $m \times n$ matrix X.

By best, we mean a matrix X_r of rank $r < p$ that, among all rank r matrices, minimizes

$$||X - X_r||$$

where $|| \cdot ||$ denotes a norm of a matrix X and where X_r belongs to the space of all rank r matrices of dimension $m \times n$.

Three popular **matrix norms** of an $m \times n$ matrix X can be expressed in terms of the singular values of X

- the **spectral** or l^2 norm $||X||_2 = \max_{||y|| \neq 0} \frac{||Xy||}{||y||} = \sigma_1$
- the **Frobenius** norm $||X||_F = \sqrt{\sigma_1^2 + \cdots + \sigma_p^2}$
- the **nuclear** norm $||X||_N = \sigma_1 + \cdots + \sigma_p$

The Eckart-Young theorem states that for each of these three norms, same rank r matrix is best and that it equals

$$\hat{X}_r = \sigma_1 U_1 V_1^\top + \sigma_2 U_2 V_2^\top + \cdots + \sigma_r U_r V_r^\top \tag{4.8}$$

This is a very powerful theorem that says that we can take our $m \times n$ matrix X that in not full rank, and we can best approximate it by a full rank $p \times p$ matrix through the SVD.

Moreover, if some of these p singular values carry more information than others, and if we want to have the most amount of information with the least amount of data, we can take r leading singular values ordered by magnitude.

We'll say more about this later when we present Principal Component Analysis.

You can read about the Eckart-Young theorem and some of its uses here.

We'll make use of this theorem when we discuss principal components analysis (PCA) and also dynamic mode decomposition (DMD).

4.6 Full and Reduced SVD's

Up to now we have described properties of a **full** SVD in which shapes of U, Σ, and V are (m, m), (m, n), (n, n), respectively.

There is an alternative bookkeeping convention called an **economy** or **reduced** SVD in which the shapes of U, Σ and V are different from what they are in a full SVD.

Thus, note that because we assume that X has rank p, there are only p nonzero singular values, where $p = \text{rank}(X) \leq \min(m, n)$.

A **reduced** SVD uses this fact to express U, Σ, and V as matrices with shapes (m, p), (p, p), (n, p).

You can read about reduced and full SVD here https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html

For a full SVD,

$$\begin{aligned} UU^\top &= I \qquad & U^\top U &= I \\ VV^\top &= I \qquad & V^\top V &= I \end{aligned}$$

But not all these properties hold for a **reduced** SVD.

Which properties hold depend on whether we are in a **tall-skinny** case or a **short-fat** case.

- In a **tall-skinny** case in which $m >> n$, for a **reduced** SVD

$$\begin{aligned} UU^\top &\neq I \qquad & U^\top U &= I \\ VV^\top &= I \qquad & V^\top V &= I \end{aligned}$$

- In a **short-fat** case in which $m << n$, for a **reduced** SVD

$$\begin{aligned} UU^\top &= I \qquad & U^\top U &= I \\ VV^\top &= I \qquad & V^\top V &\neq I \end{aligned}$$

When we study Dynamic Mode Decomposition below, we shall want to remember these properties when we use a reduced SVD to compute some DMD representations.

Let's do an exercise to compare **full** and **reduced** SVD's.

To review,

- in a **full** SVD
  - U is $m \times m$
  - Σ is $m \times n$
  - V is $n \times n$
- in a **reduced** SVD
  - U is $m \times p$
  - Σ is $p \times p$
  - V is $n \times p$

First, let's study a case in which $m = 5 > n = 2$.

(This is a small example of the **tall-skinny** case that will concern us when we study **Dynamic Mode Decompositions** below.)

```
import numpy as np
X = np.random.rand(5,2)
U, S, V = np.linalg.svd(X,full_matrices=True)  # full SVD
Uhat, Shat, Vhat = np.linalg.svd(X,full_matrices=False) # economy SVD
print('U, S, V =')
U, S, V
```

```
U, S, V =
```

```
(array([[-0.31,  0.95,  0.01,  0.05,  0.08],
        [-0.34, -0.03, -0.4 , -0.47, -0.7 ],
        [-0.41, -0.11,  0.87, -0.14, -0.21],
        [-0.44, -0.17, -0.15,  0.83, -0.25],
        [-0.66, -0.25, -0.23, -0.25,  0.63]]),
 array([1.62, 0.41]),
 array([[-0.89, -0.46],
        [-0.46,  0.89]]))
```

```
print('Uhat, Shat, Vhat = ')
Uhat, Shat, Vhat
```

```
Uhat, Shat, Vhat =
```

```
(array([[-0.31,  0.95],
        [-0.34, -0.03],
        [-0.41, -0.11],
        [-0.44, -0.17],
        [-0.66, -0.25]]),
 array([1.62, 0.41]),
 array([[-0.89, -0.46],
        [-0.46,  0.89]]))
```

```
rr = np.linalg.matrix_rank(X)
print(f'rank of X = {rr}')
```

```
rank of X = 2
```

Properties:

- Where U is constructed via a full SVD, $U^\top U = I_{m\times m}$ and $UU^\top = I_{m\times m}$
- Where $\hat{U}$ is constructed via a reduced SVD, although $\hat{U}^\top \hat{U} = I_{p\times p}$, it happens that $\hat{U}\hat{U}^\top \neq I_{m\times m}$

We illustrate these properties for our example with the following code cells.

```
UTU = U.T@U
UUT = U@U.T
print('UUT, UTU = ')
UUT, UTU
```

```
UUT, UTU =
(array([[ 1.00e+00,  1.67e-16, -6.94e-17, -1.32e-16, -1.18e-16],
        [ 1.67e-16,  1.00e+00,  2.78e-17,  5.55e-17,  5.55e-17],
        [-6.94e-17,  2.78e-17,  1.00e+00,  1.39e-17,  5.55e-17],
        [-1.32e-16,  5.55e-17,  1.39e-17,  1.00e+00,  1.11e-16],
        [-1.18e-16,  5.55e-17,  5.55e-17,  1.11e-16,  1.00e+00]]),
 array([[ 1.00e+00,  2.50e-16,  0.00e+00,  0.00e+00,  5.55e-17],
        [ 2.50e-16,  1.00e+00, -8.33e-17, -1.73e-16, -1.94e-16],
        [ 0.00e+00, -8.33e-17,  1.00e+00, -9.02e-17, -5.55e-17],
        [ 0.00e+00, -1.73e-16, -9.02e-17,  1.00e+00, -5.55e-17],
        [ 5.55e-17, -1.94e-16, -5.55e-17, -5.55e-17,  1.00e+00]]))
```

```
UhatUhatT = Uhat@Uhat.T
UhatTUhat = Uhat.T@Uhat
print('UhatUhatT, UhatTUhat= ')
UhatUhatT, UhatTUhat
```

```
UhatUhatT, UhatTUhat=
```

```
(array([[ 0.99,  0.08,  0.02, -0.02, -0.03],
        [ 0.08,  0.12,  0.14,  0.15,  0.23],
        [ 0.02,  0.14,  0.18,  0.2 ,  0.3 ],
        [-0.02,  0.15,  0.2 ,  0.22,  0.33],
        [-0.03,  0.23,  0.3 ,  0.33,  0.49]]),
 array([[1.0e+00, 2.5e-16],
        [2.5e-16, 1.0e+00]]))
```

Remarks:

The cells above illustrate the application of the `full_matrices=True` and `full_matrices=False` options. Using `full_matrices=False` returns a reduced singular value decomposition.

The **full** and **reduced** SVD's both accurately decompose an $m \times n$ matrix X

When we study Dynamic Mode Decompositions below, it will be important for us to remember the preceding properties of full and reduced SVD's in such tall-skinny cases.

Now let's turn to a short-fat case.

To illustrate this case, we'll set $m = 2 < 5 = n$ and compute both full and reduced SVD's.

```
import numpy as np
X = np.random.rand(2,5)
U, S, V = np.linalg.svd(X,full_matrices=True)  # full SVD
Uhat, Shat, Vhat = np.linalg.svd(X,full_matrices=False) # economy SVD
print('U, S, V = ')
U, S, V
```

```
U, S, V =
```

```
(array([[ 0.74, -0.67],
        [ 0.67,  0.74]]),
 array([1.69, 0.57]),
 array([[ 0.73,  0.47,  0.16,  0.43,  0.17],
        [-0.19,  0.68,  0.45, -0.53, -0.13],
```

(continues on next page)

(continued from previous page)

```
         [ 0.04, -0.47,  0.87,  0.12,  0.03],
         [-0.62,  0.3 ,  0.1 ,  0.72, -0.09],
         [-0.21,  0.05,  0.02, -0.09,  0.97]]))
```

```
print('Uhat, Shat, Vhat = ')
Uhat, Shat, Vhat
```

```
Uhat, Shat, Vhat =
```

```
(array([[ 0.74, -0.67],
        [ 0.67,  0.74]]),
 array([1.69, 0.57]),
 array([[ 0.73,  0.47,  0.16,  0.43,  0.17],
        [-0.19,  0.68,  0.45, -0.53, -0.13]]))
```

Let's verify that our reduced SVD accurately represents X

```
SShat=np.diag(Shat)
np.allclose(X, Uhat@SShat@Vhat)
```

```
True
```

4.7 Polar Decomposition

A **reduced** singular value decomposition (SVD) of X is related to a **polar decomposition** of X

$$X = SQ$$

where

$$\begin{aligned} S &= U\Sigma U^\top \\ Q &= UV^\top \end{aligned}$$

Here

- S is an $m \times m$ **symmetric** matrix
- Q is an $m \times n$ **orthogonal** matrix

and in our reduced SVD

- U is an $m \times p$ orthonormal matrix
- Σ is a $p \times p$ diagonal matrix
- V is an $n \times p$ orthonormal

4.8 Application: Principal Components Analysis (PCA)

Let's begin with a case in which $n >> m$, so that we have many more individuals n than attributes m.

The matrix X is **short and fat** in an $n >> m$ case as opposed to a **tall and skinny** case with $m >> n$ to be discussed later.

We regard X as an $m \times n$ matrix of **data**:

$$X = [X_1 \mid X_2 \mid \cdots \mid X_n]$$

where for $j = 1, \dots, n$ the column vector $X_j = \begin{bmatrix} X_{1j} \\ X_{2j} \\ \vdots \\ X_{mj} \end{bmatrix}$ is a vector of observations on variables $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$.

In a **time series** setting, we would think of columns j as indexing different **times** at which random variables are observed, while rows index different random variables.

In a **cross-section** setting, we would think of columns j as indexing different **individuals** for which random variables are observed, while rows index different **attributes**.

As we have seen before, the SVD is a way to decompose a matrix into useful components, just like polar decomposition, eigendecomposition, and many others.

PCA, on the other hand, is a method that builds on the SVD to analyze data. The goal is to apply certain steps, to help better visualize patterns in data, using statistical tools to capture the most important patterns in data.

Step 1: Standardize the data:

Because our data matrix may hold variables of different units and scales, we first need to standardize the data.

First by computing the average of each row of X.

$$\bar{X}_j = \frac{1}{m} \sum_{i=1}^{m} x_{i,j}$$

We then create an average matrix out of these means:

$$\bar{X} = \begin{bmatrix} 1 \\ 1 \\ \cdots \\ 1 \end{bmatrix} [\bar{X}_1 \mid \bar{X}_2 \mid \cdots \mid \bar{X}_n]$$

And subtract out of the original matrix to create a mean centered matrix:

$$B = X - \bar{X}$$

Step 2: Compute the covariance matrix:

Then because we want to extract the relationships between variables rather than just their magnitude, in other words, we want to know how they can explain each other, we compute the covariance matrix of B.

$$C = \frac{1}{n} B^\top B$$

Step 3: Decompose the covariance matrix and arrange the singular values:

If the matrix C is diagonalizable, we can eigendecompose it, find its eigenvalues and rearrange the eigenvalue and eigenvector matrices in a decreasing other.

If C is not diagonalizable, we can perform an SVD of C:

$$\begin{aligned} B^T B &= V\Sigma^\top U^\top U \Sigma V^\top \\ &= V\Sigma^\top \Sigma V^\top \end{aligned}$$

$$C = \frac{1}{n} V\Sigma^\top \Sigma V^\top$$

We can then rearrange the columns in the matrices V and Σ so that the singular values are in decreasing order.

Step 4: Select singular values, (optional) truncate the rest:

We can now decide how many singular values to pick, based on how much variance you want to retain. (e.g., retaining 95% of the total variance).

We can obtain the percentage by calculating the variance contained in the leading r factors divided by the variance in total:

$$\frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{p} \sigma_i^2}$$

Step 5: Create the Score Matrix:

$$\begin{aligned} T &= BV \\ &= U\Sigma V^\top \\ &= U\Sigma \end{aligned}$$

4.9 Relationship of PCA to SVD

To relate an SVD to a PCA of data set X, first construct the SVD of the data matrix X:

Let's assume that sample means of all variables are zero, so we don't need to standardize our matrix.

$$X = U\Sigma V^\top = \sigma_1 U_1 V_1^\top + \sigma_2 U_2 V_2^\top + \cdots + \sigma_p U_p V_p^\top \tag{4.9}$$

where

$$U = [U_1 | U_2 | \ldots | U_m]$$

$$V^\top = \begin{bmatrix} V_1^\top \\ V_2^\top \\ \ldots \\ V_n^\top \end{bmatrix}$$

In equation (4.9), each of the $m \times n$ matrices $U_j V_j^\top$ is evidently of rank 1.

Thus, we have

$$X = \sigma_1 \begin{pmatrix} U_{11} V_1^\top \\ U_{21} V_1^\top \\ \cdots \\ U_{m1} V_1^\top \end{pmatrix} + \sigma_2 \begin{pmatrix} U_{12} V_2^\top \\ U_{22} V_2^\top \\ \cdots \\ U_{m2} V_2^\top \end{pmatrix} + \ldots + \sigma_p \begin{pmatrix} U_{1p} V_p^\top \\ U_{2p} V_p^\top \\ \cdots \\ U_{mp} V_p^\top \end{pmatrix} \tag{4.10}$$

Here is how we would interpret the objects in the matrix equation (4.10) in a time series context:

- for each $k = 1, \ldots, n$, the object $\{V_{kj}\}_{j=1}^{n}$ is a time series for the kth **principal component**

- $U_j = \begin{bmatrix} U_{1k} \\ U_{2k} \\ \ldots \\ U_{mk} \end{bmatrix}$ $k = 1, \ldots, m$ is a vector of **loadings** of variables X_i on the kth principal component, $i = 1, \ldots, m$
- σ_k for each $k = 1, \ldots, p$ is the strength of kth **principal component**, where strength means contribution to the overall covariance of X.

4.10 PCA with Eigenvalues and Eigenvectors

We now use an eigen decomposition of a sample covariance matrix to do PCA.

Let $X_{m\times n}$ be our $m \times n$ data matrix.

Let's assume that sample means of all variables are zero.

We can assure this by **pre-processing** the data by subtracting sample means.

Define a sample covariance matrix Ω as

$$\Omega = XX^\top$$

Then use an eigen decomposition to represent Ω as follows:

$$\Omega = P\Lambda P^\top$$

Here

- P is $m \times m$ matrix of eigenvectors of Ω
- Λ is a diagonal matrix of eigenvalues of Ω

We can then represent X as

$$X = P\epsilon$$

where

$$\epsilon = P^{-1}X$$

and

$$\epsilon\epsilon^\top = \Lambda.$$

We can verify that

$$XX^\top = P\Lambda P^\top. \tag{4.11}$$

It follows that we can represent the data matrix X as

$$X = [X_1|X_2|\ldots|X_m] = [P_1|P_2|\ldots|P_m] \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \ldots \\ \epsilon_m \end{bmatrix} = P_1\epsilon_1 + P_2\epsilon_2 + \ldots + P_m\epsilon_m$$

To reconcile the preceding representation with the PCA that we had obtained earlier through the SVD, we first note that $\epsilon_j^2 = \lambda_j \equiv \sigma_j^2$.

Now define $\tilde{\epsilon_j} = \frac{\epsilon_j}{\sqrt{\lambda_j}}$, which implies that $\tilde{\epsilon}_j \tilde{\epsilon}_j^\top = 1$.

Therefore

$$\begin{aligned} X &= \sqrt{\lambda_1} P_1 \tilde{\epsilon_1} + \sqrt{\lambda_2} P_2 \tilde{\epsilon_2} + \ldots + \sqrt{\lambda_m} P_m \tilde{\epsilon_m} \\ &= \sigma_1 P_1 \tilde{\epsilon_2} + \sigma_2 P_2 \tilde{\epsilon_2} + \ldots + \sigma_m P_m \tilde{\epsilon_m}, \end{aligned}$$

which agrees with

$$X = \sigma_1 U_1 {V_1}^T + \sigma_2 U_2 {V_2}^T + \ldots + \sigma_r U_r {V_r}^T$$

provided that we set

- $U_j = P_j$ (a vector of loadings of variables on principal component j)
- ${V_k}^T = \tilde{\epsilon_k}$ (the kth principal component)

Because there are alternative algorithms for computing P and U for given a data matrix X, depending on algorithms used, we might have sign differences or different orders of eigenvectors.

We can resolve such ambiguities about U and P by

1. sorting eigenvalues and singular values in descending order
2. imposing positive diagonals on P and U and adjusting signs in $V^\top$ accordingly

4.11 Connections

To pull things together, it is useful to assemble and compare some formulas presented above.

First, consider an SVD of an $m \times n$ matrix:

$$X = U \Sigma V^\top$$

Compute:

$$\begin{aligned} XX^\top &= U\Sigma V^\top V \Sigma^\top U^\top \\ &\equiv U \Sigma \Sigma^\top U^\top \\ &\equiv U \Lambda U^\top \end{aligned} \tag{4.12}$$

Compare representation (4.12) with equation (4.11) above.

Evidently, U in the SVD is the matrix P of eigenvectors of $XX^\top$ and $\Sigma\Sigma^\top$ is the matrix Λ of eigenvalues.

Second, let's compute

$$\begin{aligned} X^\top X &= V \Sigma^\top U^\top U \Sigma V^\top \\ &= V \Sigma^\top \Sigma V^\top \end{aligned}$$

Thus, the matrix V in the SVD is the matrix of eigenvectors of $X^\top X$

Summarizing and fitting things together, we have the eigen decomposition of the sample covariance matrix

$$XX^\top = P \Lambda P^\top$$

where P is an orthogonal matrix.

Further, from the SVD of X, we know that

$$XX^\top = U \Sigma \Sigma^\top U^\top$$

where U is an orthogonal matrix.

Thus, $P = U$ and we have the representation of X

$$X = P\epsilon = U\Sigma V^\top$$

It follows that

$$U^\top X = \Sigma V^\top = \epsilon$$

Note that the preceding implies that

$$\epsilon\epsilon^\top = \Sigma V^\top V \Sigma^\top = \Sigma\Sigma^\top = \Lambda,$$

so that everything fits together.

Below we define a class `DecomAnalysis` that wraps PCA and SVD for a given a data matrix `X`.

```
class DecomAnalysis:
    """
    A class for conducting PCA and SVD.
    X: data matrix
    r_component: chosen rank for best approximation
    """

    def __init__(self, X, r_component=None):

        self.X = X

        self.Ω = (X @ X.T)

        self.m, self.n = X.shape
        self.r = LA.matrix_rank(X)

        if r_component:
            self.r_component = r_component
        else:
            self.r_component = self.m

    def pca(self):

        ⍰, P = LA.eigh(self.Ω)    # columns of P are eigenvectors

        ind = sorted(range(⍰.size), key=lambda x: ⍰[x], reverse=True)

        # sort by eigenvalues
        self.⍰ = ⍰[ind]
        P = P[:, ind]
        self.P = P @ diag_sign(P)

        self.Λ = np.diag(self.⍰)

        self.explained_ratio_pca = np.cumsum(self.⍰) / self.⍰.sum()

        # compute the N by T matrix of principal components
        self.⍰ = self.P.T @ self.X

        P = self.P[:, :self.r_component]
```

(continues on next page)

(continued from previous page)

```
        □ = self.□[:self.r_component, :]

        # transform data
        self.X_pca = P @ □

    def svd(self):

        U, □, VT = LA.svd(self.X)

        ind = sorted(range(□.size), key=lambda x: □[x], reverse=True)

        # sort by eigenvalues
        d = min(self.m, self.n)

        self.□ = □[ind]
        U = U[:, ind]
        D = diag_sign(U)
        self.U = U @ D
        VT[:d, :] = D @ VT[ind, :]
        self.VT = VT

        self.Σ = np.zeros((self.m, self.n))
        self.Σ[:d, :d] = np.diag(self.□)

        □_sq = self.□ ** 2
        self.explained_ratio_svd = np.cumsum(□_sq) / □_sq.sum()

        # slicing matrices by the number of components to use
        U = self.U[:, :self.r_component]
        Σ = self.Σ[:self.r_component, :self.r_component]
        VT = self.VT[:self.r_component, :]

        # transform data
        self.X_svd = U @ Σ @ VT

    def fit(self, r_component):

        # pca
        P = self.P[:, :r_component]
        □ = self.□[:r_component, :]

        # transform data
        self.X_pca = P @ □

        # svd
        U = self.U[:, :r_component]
        Σ = self.Σ[:r_component, :r_component]
        VT = self.VT[:r_component, :]

        # transform data
        self.X_svd = U @ Σ @ VT

def diag_sign(A):
    "Compute the signs of the diagonal of matrix A"

    D = np.diag(np.sign(np.diag(A)))
```

(continues on next page)

(continued from previous page)

```
    return D
```

We also define a function that prints out information so that we can compare decompositions obtained by different algorithms.

```
def compare_pca_svd(da):
    """
    Compare the outcomes of PCA and SVD.
    """

    da.pca()
    da.svd()

    print('Eigenvalues and Singular values\n')
    print(f'λ = {da.λ}\n')
    print(f'σ^2 = {da.σ**2}\n')
    print('\n')

    # loading matrices
    fig, axs = plt.subplots(1, 2, figsize=(14, 5))
    plt.suptitle('loadings')
    axs[0].plot(da.P.T)
    axs[0].set_title('P')
    axs[0].set_xlabel('m')
    axs[1].plot(da.U.T)
    axs[1].set_title('U')
    axs[1].set_xlabel('m')
    plt.show()

    # principal components
    fig, axs = plt.subplots(1, 2, figsize=(14, 5))
    plt.suptitle('principal components')
    axs[0].plot(da.ε.T)
    axs[0].set_title('ε')
    axs[0].set_xlabel('n')
    axs[1].plot(da.VT[:da.r, :].T * np.sqrt(da.λ))
    axs[1].set_title('$V^\top *\sqrt{\lambda}$')
    axs[1].set_xlabel('n')
    plt.show()
```

4.12 Exercises

Exercise 4.12.1

In Ordinary Least Squares (OLS), we learn to compute $\hat{\beta} = (X^\top X)^{-1} X^\top y$, but there are cases such as when we have colinearity or an underdetermined system: **short fat** matrix.

In these cases, the $(X^\top X)$ matrix is not not invertible (its determinant is zero) or ill-conditioned (its determinant is very close to zero).

What we can do instead is to create what is called a pseudoinverse, a full rank approximation of the inverted matrix so we can compute $\hat{\beta}$ with it.

Thinking in terms of the Eckart-Young theorem, build the pseudoinverse matrix X^+ and use it to compute $\hat{\beta}$.

Solution to Exercise 4.12.1

We can use SVD to compute the pseudoinverse:

$$X = U\Sigma V^\top$$

inverting X, we have:

$$X^+ = V\Sigma^+U^\top$$

where:

$$\Sigma^+ = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{\sigma_2} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_p} & 0 \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

and finally:

$$\hat{\beta} = X^+y = V\Sigma^+U^\top y$$

For an example PCA applied to analyzing the structure of intelligence tests see this lecture Multivariable Normal Distribution.

Look at parts of that lecture that describe and illustrate the classic factor analysis model.

As mentioned earlier, in a sequel to this lecture about *Dynamic Mode Decompositions*, we'll describe how SVD's provide ways rapidly to compute reduced-order approximations to first-order Vector Autoregressions (VARs).

CHAPTER

FIVE

CLASSICAL CONTROL WITH LINEAR ALGEBRA

5.1 Overview

In an earlier lecture Linear Quadratic Dynamic Programming Problems, we have studied how to solve a special class of dynamic optimization and prediction problems by applying the method of dynamic programming. In this class of problems

- the objective function is **quadratic** in **states** and **controls**.
- the one-step transition function is **linear**.
- shocks are IID Gaussian or martingale differences.

In this lecture and a companion lecture *Classical Filtering with Linear Algebra*, we study the classical theory of linear-quadratic (LQ) optimal control problems.

The classical approach does not use the two closely related methods – dynamic programming and Kalman filtering – that we describe in other lectures, namely, Linear Quadratic Dynamic Programming Problems and A First Look at the Kalman Filter.

Instead, they use either

- z-transform and lag operator methods, or
- matrix decompositions applied to linear systems of first-order conditions for optimum problems.

In this lecture and the sequel *Classical Filtering with Linear Algebra*, we mostly rely on elementary linear algebra.

The main tool from linear algebra we'll put to work here is LU decomposition.

We'll begin with discrete horizon problems.

Then we'll view infinite horizon problems as appropriate limits of these finite horizon problems.

Later, we will examine the close connection between LQ control and least-squares prediction and filtering problems.

These classes of problems are connected in the sense that to solve each, essentially the same mathematics is used.

Let's start with some standard imports:

```
import numpy as np
import matplotlib.pyplot as plt
```

5.1.1 References

Useful references include [Whittle, 1963], [Hansen and Sargent, 1980], [Orfanidis, 1988], [Athanasios and Pillai, 1991], and [Muth, 1960].

5.2 A Control Problem

Let L be the **lag operator**, so that, for sequence $\{x_t\}$ we have $Lx_t = x_{t-1}$.

More generally, let $L^k x_t = x_{t-k}$ with $L^0 x_t = x_t$ and

$$d(L) = d_0 + d_1 L + \ldots + d_m L^m$$

where $d_0, d_1, \ldots, d_m$ is a given scalar sequence.

Consider the discrete-time control problem

$$\max_{\{y_t\}} \lim_{N\to\infty} \sum_{t=0}^{N} \beta^t \left\{ a_t y_t - \frac{1}{2} h y_t^2 - \frac{1}{2} \left[d(L) y_t\right]^2 \right\}, \tag{5.1}$$

where

- h is a positive parameter and $\beta \in (0,1)$ is a discount factor.
- $\{a_t\}_{t\geq 0}$ is a sequence of exponential order less than $\beta^{-1/2}$, by which we mean $\lim_{t\to\infty} \beta^{\frac{t}{2}} a_t = 0$.

Maximization in (5.1) is subject to initial conditions for $y_{-1}, y_{-2} \ldots, y_{-m}$.

Maximization is over infinite sequences $\{y_t\}_{t\geq 0}$.

5.2.1 Example

The formulation of the LQ problem given above is broad enough to encompass many useful models.

As a simple illustration, recall that in LQ Control: Foundations we consider a monopolist facing stochastic demand shocks and adjustment costs.

Let's consider a deterministic version of this problem, where the monopolist maximizes the discounted sum

$$\sum_{t=0}^{\infty} \beta^t \pi_t$$

and

$$\pi_t = p_t q_t - c q_t - \gamma (q_{t+1} - q_t)^2 \quad \text{with} \quad p_t = \alpha_0 - \alpha_1 q_t + d_t$$

In this expression, q_t is output, c is average cost of production, and d_t is a demand shock.

The term $\gamma(q_{t+1} - q_t)^2$ represents adjustment costs.

You will be able to confirm that the objective function can be rewritten as (5.1) when

- $a_t := \alpha_0 + d_t - c$
- $h := 2\alpha_1$
- $d(L) := \sqrt{2\gamma}(I - L)$

Further examples of this problem for factor demand, economic growth, and government policy problems are given in ch. IX of [Sargent, 1987].

5.3 Finite Horizon Theory

We first study a finite N version of the problem.

Later we will study an infinite horizon problem solution as a limiting version of a finite horizon problem.

(This will require being careful because the limits as $N \to \infty$ of the necessary and sufficient conditions for maximizing finite N versions of (5.1) are not sufficient for maximizing (5.1))

We begin by

1. fixing $N > m$,
2. differentiating the finite version of (5.1) with respect to $y_0, y_1, \dots, y_N$, and
3. setting these derivatives to zero.

For $t = 0, \dots, N - m$ these first-order necessary conditions are the *Euler equations*.

For $t = N - m + 1, \dots, N$, the first-order conditions are a set of *terminal conditions*.

Consider the term

$$\begin{aligned} J &= \sum_{t=0}^{N} \beta^t [d(L) y_t][d(L) y_t] \\ &= \sum_{t=0}^{N} \beta^t \, (d_0 \, y_t + d_1 \, y_{t-1} + \cdots + d_m \, y_{t-m}) \, (d_0 \, y_t + d_1 \, y_{t-1} + \cdots + d_m \, y_{t-m}) \end{aligned}$$

Differentiating J with respect to y_t for $t = 0, \ 1, \ \dots, \ N - m$ gives

$$\begin{aligned} \frac{\partial J}{\partial y_t} &= 2\beta^t \, d_0 \, d(L) y_t + 2\beta^{t+1} \, d_1 \, d(L) y_{t+1} + \cdots + 2\beta^{t+m} \, d_m \, d(L) y_{t+m} \\ &= 2\beta^t \left(d_0 + d_1 \, \beta L^{-1} + d_2 \, \beta^2 \, L^{-2} + \cdots + d_m \, \beta^m \, L^{-m} \right) d(L) y_t \end{aligned}$$

We can write this more succinctly as

$$\frac{\partial J}{\partial y_t} = 2\beta^t \, d(\beta L^{-1}) \, d(L) y_t \tag{5.2}$$

Differentiating J with respect to y_t for $t = N - m + 1, \dots, N$ gives

$$\begin{aligned} \frac{\partial J}{\partial y_N} &= 2\beta^N \, d_0 \, d(L) y_N \\ \frac{\partial J}{\partial y_{N-1}} &= 2\beta^{N-1} \left[d_0 + \beta \, d_1 \, L^{-1} \right] d(L) y_{N-1} \\ \vdots \quad & \quad \vdots \\ \frac{\partial J}{\partial y_{N-m+1}} &= 2\beta^{N-m+1} \left[d_0 + \beta L^{-1} \, d_1 + \cdots + \beta^{m-1} \, L^{-m+1} \, d_{m-1} \right] d(L) y_{N-m+1} \end{aligned} \tag{5.3}$$

With these preliminaries under our belts, we are ready to differentiate (5.1).

Differentiating (5.1) with respect to y_t for $t = 0, \dots, N - m$ gives the Euler equations

$$[h + d \, (\beta L^{-1}) \, d(L)] y_t = a_t, \quad t = 0, \, 1, \, \dots, N - m \tag{5.4}$$

The system of equations (5.4) forms a $2 \times m$ order linear *difference equation* that must hold for the values of t indicated.

Differentiating (5.1) with respect to y_t for $t = N - m + 1, \dots, N$ gives the terminal conditions

$$\begin{aligned} \beta^N (a_N - hy_N - d_0\, d(L) y_N) &= 0 \\ \beta^{N-1} \left(a_{N-1} - hy_{N-1} - \left(d_0 + \beta\, d_1\, L^{-1}\right) d(L)\, y_{N-1}\right) &= 0 \\ \vdots \qquad \vdots \qquad &= 0 \\ \beta^{N-m+1} \left(a_{N-m+1} - hy_{N-m+1} - (d_0 + \beta L^{-1} d_1 + \cdots + \beta^{m-1} L^{-m+1} d_{m-1}) d(L) y_{N-m+1}\right) &= 0 \end{aligned} \tag{5.5}$$

In the finite N problem, we want simultaneously to solve (5.4) subject to the m initial conditions $y_{-1}, \dots, y_{-m}$ and the m terminal conditions (5.5).

These conditions uniquely pin down the solution of the finite N problem.

That is, for the finite N problem, conditions (5.4) and (5.5) are necessary and sufficient for a maximum, by concavity of the objective function.

Next, we describe how to obtain the solution using matrix methods.

5.3.1 Matrix Methods

Let's look at how linear algebra can be used to tackle and shed light on the finite horizon LQ control problem.

A Single Lag Term

Let's begin with the special case in which $m = 1$.

We want to solve the system of $N + 1$ linear equations

$$\begin{aligned} [h + d\,(\beta L^{-1})\, d\,(L)] y_t &= a_t, \quad t = 0,\ 1,\ \dots,\ N-1 \\ \beta^N [a_N - h\, y_N - d_0\, d\,(L) y_N] &= 0 \end{aligned} \tag{5.6}$$

where $d(L) = d_0 + d_1 L$.

These equations are to be solved for $y_0, y_1, \dots, y_N$ as functions of $a_0, a_1, \dots, a_N$ and y_{-1}.

Let

$$\phi(L) = \phi_0 + \phi_1 L + \beta\phi_1 L^{-1} = h + d(\beta L^{-1}) d(L) = (h + d_0^2 + d_1^2) + d_1 d_0 L + d_1 d_0 \beta L^{-1}$$

Then we can represent (5.6) as the matrix equation

$$\begin{bmatrix} (\phi_0 - d_1^2) & \phi_1 & 0 & 0 & \dots & \dots & 0 \\ \beta\phi_1 & \phi_0 & \phi_1 & 0 & \dots & \dots & 0 \\ 0 & \beta\phi_1 & \phi_0 & \phi_1 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & \beta\phi_1 & \phi_0 & \phi_1 \\ 0 & \dots & \dots & \dots & 0 & \beta\phi_1 & \phi_0 \end{bmatrix} \begin{bmatrix} y_N \\ y_{N-1} \\ y_{N-2} \\ \vdots \\ y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} a_N \\ a_{N-1} \\ a_{N-2} \\ \vdots \\ a_1 \\ a_0 - \phi_1 y_{-1} \end{bmatrix} \tag{5.7}$$

or

$$W\bar{y} = \bar{a} \tag{5.8}$$

Notice how we have chosen to arrange the y_t's in reverse time order.

The matrix W on the left side of (5.7) is "almost" a Toeplitz matrix (where each descending diagonal is constant).

There are two sources of deviation from the form of a Toeplitz matrix

1. The first element differs from the remaining diagonal elements, reflecting the terminal condition.
2. The sub-diagonal elements equal β time the super-diagonal elements.

The solution of (5.8) can be expressed in the form

$$\bar{y} = W^{-1} \bar{a} \tag{5.9}$$

which represents each element y_t of $\bar{y}$ as a function of the entire vector $\bar{a}$.

That is, y_t is a function of past, present, and future values of a's, as well as of the initial condition y_{-1}.

An Alternative Representation

An alternative way to express the solution to (5.7) or (5.8) is in so-called **feedback-feedforward** form.

The idea here is to find a solution expressing y_t as a function of *past* y's and *current* and *future* a's.

To achieve this solution, one can use an LU decomposition of W.

There always exists a decomposition of W of the form $W = LU$ where

- L is an $(N+1) \times (N+1)$ lower triangular matrix.
- U is an $(N+1) \times (N+1)$ upper triangular matrix.

The factorization can be normalized so that the diagonal elements of U are unity.

Using the LU representation in (5.9), we obtain

$$U \bar{y} = L^{-1} \bar{a} \tag{5.10}$$

Since L^{-1} is lower triangular, this representation expresses y_t as a function of

- lagged y's (via the term $U \bar{y}$), and
- current and future a's (via the term $L^{-1} \bar{a}$)

Because there are zeros everywhere in the matrix on the left of (5.7) except on the diagonal, super-diagonal, and sub-diagonal, the LU decomposition takes

- L to be zero except in the diagonal and the leading sub-diagonal.
- U to be zero except on the diagonal and the super-diagonal.

Thus, (5.10) has the form

$$\begin{bmatrix} 1 & U_{12} & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & U_{23} & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & U_{34} & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & U_{N,N+1} \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_N \\ y_{N-1} \\ y_{N-2} \\ y_{N-3} \\ \vdots \\ y_1 \\ y_0 \end{bmatrix} =$$

$$\begin{bmatrix} L^{-1}_{11} & 0 & 0 & \dots & 0 \\ L^{-1}_{21} & L^{-1}_{22} & 0 & \dots & 0 \\ L^{-1}_{31} & L^{-1}_{32} & L^{-1}_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L^{-1}_{N,1} & L^{-1}_{N,2} & L^{-1}_{N,3} & \dots & 0 \\ L^{-1}_{N+1,1} & L^{-1}_{N+1,2} & L^{-1}_{N+1,3} & \dots & L^{-1}_{N+1\, N+1} \end{bmatrix} \begin{bmatrix} a_N \\ a_{N-1} \\ a_{N-2} \\ \vdots \\ a_1 \\ a_0 - \phi_1 y_{-1} \end{bmatrix}$$

where L^{-1}_{ij} is the (i,j) element of L^{-1} and U_{ij} is the (i,j) element of U.

Note how the left side for a given t involves y_t and one lagged value y_{t-1} while the right side involves all future values of the forcing process $a_t, a_{t+1}, \dots, a_N$.

Additional Lag Terms

We briefly indicate how this approach extends to the problem with $m > 1$.

Assume that $\beta = 1$ and let D_{m+1} be the $(m+1) \times (m+1)$ symmetric matrix whose elements are determined from the following formula:

$$D_{jk} = d_0 d_{k-j} + d_1 d_{k-j+1} + \dots + d_{j-1} d_{k-1}, \qquad k \geq j$$

Let I_{m+1} be the $(m+1) \times (m+1)$ identity matrix.

Let ϕ_j be the coefficients in the expansion $\phi(L) = h + d(L^{-1})d(L)$.

Then the first order conditions (5.4) and (5.5) can be expressed as:

$$(D_{m+1} + hI_{m+1}) \begin{bmatrix} y_N \\ y_{N-1} \\ \vdots \\ y_{N-m} \end{bmatrix} = \begin{bmatrix} a_N \\ a_{N-1} \\ \vdots \\ a_{N-m} \end{bmatrix} + M \begin{bmatrix} y_{N-m+1} \\ y_{N-m-2} \\ \vdots \\ y_{N-2m} \end{bmatrix}$$

where M is $(m+1) \times m$ and

$$M_{ij} = \begin{cases} D_{i-j,\, m+1} \text{ for } i > j \\ 0 \text{ for } i \leq j \end{cases}$$

$$\begin{aligned}
\phi_m y_{N-1} + \phi_{m-1} y_{N-2} + \dots + \phi_0 y_{N-m-1} + \phi_1 y_{N-m-2} + \\
\dots + \phi_m y_{N-2m-1} = a_{N-m-1} \\
\phi_m y_{N-2} + \phi_{m-1} y_{N-3} + \dots + \phi_0 y_{N-m-2} + \phi_1 y_{N-m-3} + \\
\dots + \phi_m y_{N-2m-2} = a_{N-m-2} \\
\vdots \\
\phi_m y_{m+1} + \phi_{m-1} y_m + + \dots + \phi_0 y_1 + \phi_1 y_0 + \phi_m y_{-m+1} = a_1 \\
\phi_m y_m + \phi_{m-1} y_{m-1} + \phi_{m-2} + \dots + \phi_0 y_0 + \phi_1 y_{-1} + \dots + \phi_m y_{-m} = a_0
\end{aligned}$$

As before, we can express this equation as $W\bar{y} = \bar{a}$.

The matrix on the left of this equation is "almost" Toeplitz, the exception being the leading $m \times m$ submatrix in the upper left-hand corner.

We can represent the solution in feedback-feedforward form by obtaining a decomposition $LU = W$, and obtain

$$U\bar{y} = L^{-1}\bar{a} \tag{5.11}$$

$$\sum_{j=0}^{t} U_{-t+N+1,\,-t+N+j+1}\, y_{t-j} = \sum_{j=0}^{N-t} L_{-t+N+1,\,-t+N+1-j}\, \bar{a}_{t+j}\ , \qquad t = 0, 1, \dots, N$$

where $L^{-1}_{t,s}$ is the element in the (t, s) position of L, and similarly for U.

The left side of equation (5.11) is the "feedback" part of the optimal control law for y_t, while the right-hand side is the "feedforward" part.

We note that there is a different control law for each t.

Thus, in the finite horizon case, the optimal control law is time-dependent.

It is natural to suspect that as $N \to \infty$, (5.11) becomes equivalent to the solution of our infinite horizon problem, which below we shall show can be expressed as

$$c(L)y_t = c(\beta L^{-1})^{-1} a_t \ ,$$

so that as $N \to \infty$ we expect that for each fixed $t, U^{-1}_{t,t-j} \to c_j$ and $L_{t,t+j}$ approaches the coefficient on L^{-j} in the expansion of $c(\beta L^{-1})^{-1}$.

This suspicion is true under general conditions that we shall study later.

For now, we note that by creating the matrix W for large N and factoring it into the LU form, good approximations to $c(L)$ and $c(\beta L^{-1})^{-1}$ can be obtained.

5.4 Infinite Horizon Limit

For the infinite horizon problem, we propose to discover first-order necessary conditions by taking the limits of (5.4) and (5.5) as $N \to \infty$.

This approach is valid, and the limits of (5.4) and (5.5) as N approaches infinity are first-order necessary conditions for a maximum.

However, for the infinite horizon problem with $\beta < 1$, the limits of (5.4) and (5.5) are, in general, not sufficient for a maximum.

That is, the limits of (5.5) do not provide enough information uniquely to determine the solution of the Euler equation (5.4) that maximizes (5.1).

As we shall see below, a side condition on the path of y_t that together with (5.4) is sufficient for an optimum is

$$\sum_{t=0}^{\infty} \beta^t \, h y_t^2 < \infty \tag{5.12}$$

All paths that satisfy the Euler equations, except the one that we shall select below, violate this condition and, therefore, evidently lead to (much) lower values of (5.1) than does the optimal path selected by the solution procedure below.

Consider the *characteristic equation* associated with the Euler equation

$$h + d\left(\beta z^{-1}\right) d\left(z\right) = 0 \tag{5.13}$$

Notice that if $\tilde z$ is a root of equation (5.13), then so is $\beta \tilde z^{-1}$.

Thus, the roots of (5.13) come in "β-reciprocal" pairs.

Assume that the roots of (5.13) are distinct.

Let the roots be, in descending order according to their moduli, $z_1, z_2, \dots, z_{2m}$.

From the reciprocal pairs property and the assumption of distinct roots, it follows that $|z_j| > \sqrt{\beta}$ for $j \leq m$ and $|z_j| < \sqrt{\beta}$ for $j > m$.

It also follows that $z_{2m-j} = \beta z^{-1}_{j+1}, j = 0, 1, \dots, m-1$.

Therefore, the characteristic polynomial on the left side of (5.13) can be expressed as

$$\begin{aligned} h + d(\beta z^{-1})d(z) &= z^{-m} z_0 (z - z_1) \cdots (z - z_m)(z - z_{m+1}) \cdots (z - z_{2m}) \\ &= z^{-m} z_0 (z - z_1)(z - z_2) \cdots (z - z_m)(z - \beta z_m^{-1}) \cdots (z - \beta z_2^{-1})(z - \beta z_1^{-1}) \end{aligned} \tag{5.14}$$

where z_0 is a constant.

In (5.14), we substitute $(z - z_j) = -z_j(1 - \frac{1}{z_j} z)$ and $(z - \beta z_j^{-1}) = z(1 - \frac{\beta}{z_j} z^{-1})$ for $j = 1, \ldots, m$ to get

$$h + d(\beta z^{-1})d(z) = (-1)^m (z_0 z_1 \cdots z_m)(1 - \frac{1}{z_1} z) \cdots (1 - \frac{1}{z_m} z)(1 - \frac{1}{z_1} \beta z^{-1}) \cdots (1 - \frac{1}{z_m} \beta z^{-1})$$

Now define $c(z) = \sum_{j=0}^{m} c_j \, z^j$ as

$$c\,(z) = \Big[(-1)^m z_0 \, z_1 \cdots z_m\Big]^{1/2} (1 - \frac{z}{z_1}) \, (1 - \frac{z}{z_2}) \cdots (1 - \frac{z}{z_m}) \tag{5.15}$$

Notice that (5.14) can be written

$$h + d \, (\beta z^{-1}) \, d \, (z) = c \, (\beta z^{-1}) \, c \, (z) \tag{5.16}$$

It is useful to write (5.15) as

$$c(z) = c_0 (1 - \lambda_1 \, z) \ldots (1 - \lambda_m z) \tag{5.17}$$

where

$$c_0 = \left[(-1)^m \, z_0 \, z_1 \cdots z_m\right]^{1/2}; \quad \lambda_j = \frac{1}{z_j}, \; j = 1, \ldots, m$$

Since $|z_j| > \sqrt{\beta}$ for $j = 1, \ldots, m$ it follows that $|\lambda_j| < 1/\sqrt{\beta}$ for $j = 1, \ldots, m$.

Using (5.17), we can express the factorization (5.16) as

$$h + d(\beta z^{-1})d(z) = c_0^2 (1 - \lambda_1 z) \cdots (1 - \lambda_m z)(1 - \lambda_1 \beta z^{-1}) \cdots (1 - \lambda_m \beta z^{-1})$$

In sum, we have constructed a factorization (5.16) of the characteristic polynomial for the Euler equation in which the zeros of $c(z)$ exceed $\beta^{1/2}$ in modulus, and the zeros of $c\,(\beta z^{-1})$ are less than $\beta^{1/2}$ in modulus.

Using (5.16), we now write the Euler equation as

$$c(\beta L^{-1}) \, c \, (L) \, y_t = a_t$$

The unique solution of the Euler equation that satisfies condition (5.12) is

$$c(L) \, y_t = c \, (\beta L^{-1})^{-1} a_t \tag{5.18}$$

This can be established by using an argument paralleling that in chapter IX of [Sargent, 1987].

To exhibit the solution in a form paralleling that of [Sargent, 1987], we use (5.17) to write (5.18) as

$$(1 - \lambda_1 L) \cdots (1 - \lambda_m L) y_t = \frac{c_0^{-2} a_t}{(1 - \beta \lambda_1 L^{-1}) \cdots (1 - \beta \lambda_m L^{-1})} \tag{5.19}$$

Using partial fractions, we can write the characteristic polynomial on the right side of (5.19) as

$$\sum_{j=1}^{m} \frac{A_j}{1 - \lambda_j \, \beta L^{-1}} \quad \text{where} \quad A_j := \frac{c_0^{-2}}{\prod_{i \neq j} (1 - \frac{\lambda_i}{\lambda_j})}$$

Then (5.19) can be written

$$(1 - \lambda_1 L) \cdots (1 - \lambda_m L) y_t = \sum_{j=1}^{m} \frac{A_j}{1 - \lambda_j \, \beta L^{-1}} a_t$$

or

$$(1-\lambda_1 L)\cdots(1-\lambda_m L)y_t = \sum_{j=1}^{m} A_j \sum_{k=0}^{\infty} (\lambda_j \beta)^k \, a_{t+k} \tag{5.20}$$

Equation (5.20) expresses the optimum sequence for y_t in terms of m lagged y's, and m weighted infinite geometric sums of future a_t's.

Furthermore, (5.20) is the unique solution of the Euler equation that satisfies the initial conditions and condition (5.12).

In effect, condition (5.12) compels us to solve the "unstable" roots of $h + d(\beta z^{-1})d(z)$ forward (see [Sargent, 1987]).

The step of factoring the polynomial $h + d(\beta z^{-1})\, d(z)$ into $c\,(\beta z^{-1})c\,(z)$, where the zeros of $c\,(z)$ all have modulus exceeding $\sqrt{\beta}$, is central to solving the problem.

We note two features of the solution (5.20)

- Since $|\lambda_j| < 1/\sqrt{\beta}$ for all j, it follows that $(\lambda_j\ \beta) < \sqrt{\beta}$.
- The assumption that $\{a_t\}$ is of exponential order less than $1/\sqrt{\beta}$ is sufficient to guarantee that the geometric sums of future a_t's on the right side of (5.20) converge.

We immediately see that those sums will converge under the weaker condition that $\{a_t\}$ is of exponential order less than ϕ^{-1} where $\phi = \max\,\{\beta\lambda_i, i = 1, \ldots, m\}$.

Note that with a_t identically zero, (5.20) implies that in general $|y_t|$ eventually grows exponentially at a rate given by $\max_i |\lambda_i|$.

The condition $\max_i |\lambda_i| < 1/\sqrt{\beta}$ guarantees that condition (5.12) is satisfied.

In fact, $\max_i |\lambda_i| < 1/\sqrt{\beta}$ is a necessary condition for (5.12) to hold.

Were (5.12) not satisfied, the objective function would diverge to $-\infty$, implying that the y_t path could not be optimal.

For example, with $a_t = 0$, for all t, it is easy to describe a naive (nonoptimal) policy for $\{y_t, t \geq 0\}$ that gives a finite value of (5.17).

We can simply let $y_t = 0$ for $t \geq 0$.

This policy involves at most m nonzero values of hy_t^2 and $[d(L)y_t]^2$, and so yields a finite value of (5.1).

Therefore it is easy to dominate a path that violates (5.12).

5.5 Undiscounted Problems

It is worthwhile focusing on a special case of the LQ problems above: the undiscounted problem that emerges when $\beta = 1$.

In this case, the Euler equation is

$$\left(h + d(L^{-1})d(L)\right) y_t = a_t$$

The factorization of the characteristic polynomial (5.16) becomes

$$\left(h + d\,(z^{-1})d(z)\right) = c\,(z^{-1})\,c\,(z)$$

where

$$\begin{aligned} c(z) &= c_0(1-\lambda_1 z)\dots(1-\lambda_m z) \\ c_0 &= \left[(-1)^m z_0 z_1 \dots z_m\right] \\ |\lambda_j| &< 1 \text{ for } j=1,\dots,m \\ \lambda_j &= \frac{1}{z_j} \text{ for } j=1,\dots,m \\ z_0 &= \text{ constant} \end{aligned}$$

The solution of the problem becomes

$$(1-\lambda_1 L)\cdots(1-\lambda_m L)y_t = \sum_{j=1}^m A_j \sum_{k=0}^\infty \lambda_j^k a_{t+k}$$

5.5.1 Transforming Discounted to Undiscounted Problem

Discounted problems can always be converted into undiscounted problems via a simple transformation.

Consider problem (5.1) with $0 < \beta < 1$.

Define the transformed variables

$$\tilde a_t = \beta^{t/2} a_t,\ \tilde y_t = \beta^{t/2} y_t \tag{5.21}$$

Then notice that $\beta^t\,[d\,(L)y_t]^2 = [\tilde d\,(L)\tilde y_t]^2$ with $\tilde d\,(L) = \sum_{j=0}^m \tilde d_j\, L^j$ and $\tilde d_j = \beta^{j/2} d_j$.

Then the original criterion function (5.1) is equivalent to

$$\lim_{N\to\infty} \sum_{t=0}^N \{\tilde a_t\, \tilde y_t - \frac{1}{2} h\, \tilde y_t^2 - \frac{1}{2}[\tilde d\,(L)\, \tilde y_t]^2\} \tag{5.22}$$

which is to be maximized over sequences $\{\tilde y_t,\ t=0,\dots\}$ subject to $\tilde y_{-1},\cdots,\tilde y_{-m}$ given and $\{\tilde a_t,\ t=1,\dots\}$ a known bounded sequence.

The Euler equation for this problem is $[h + \tilde d\,(L^{-1})\,\tilde d\,(L)]\,\tilde y_t = \tilde a_t$.

The solution is

$$(1-\tilde\lambda_1 L)\cdots(1-\tilde\lambda_m L)\,\tilde y_t = \sum_{j=1}^m \tilde A_j \sum_{k=0}^\infty \tilde\lambda_j^k\, \tilde a_{t+k}$$

or

$$\tilde y_t = \tilde f_1\, \tilde y_{t-1} + \cdots + \tilde f_m\, \tilde y_{t-m} + \sum_{j=1}^m \tilde A_j \sum_{k=0}^\infty \tilde\lambda_j^k\, \tilde a_{t+k}, \tag{5.23}$$

where $\tilde c\,(z^{-1})\tilde c\,(z) = h + \tilde d\,(z^{-1})\tilde d\,(z)$, and where

$$\left[(-1)^m\, \tilde z_0 \tilde z_1 \dots \tilde z_m\right]^{1/2}(1-\tilde\lambda_1\, z)\dots(1-\tilde\lambda_m\, z) = \tilde c\,(z),\ \text{ where }\ |\tilde\lambda_j| < 1$$

We leave it to the reader to show that (5.23) implies the equivalent form of the solution

$$y_t = f_1\, y_{t-1} + \cdots + f_m\, y_{t-m} + \sum_{j=1}^m A_j \sum_{k=0}^\infty (\lambda_j\, \beta)^k\, a_{t+k}$$

where

$$f_j = \tilde f_j\, \beta^{-j/2},\ A_j = \tilde A_j,\ \lambda_j = \tilde\lambda_j\, \beta^{-1/2} \tag{5.24}$$

The transformations (5.21) and the inverse formulas (5.24) allow us to solve a discounted problem by first solving a related undiscounted problem.

5.6 Implementation

Here’s the code that computes solutions to the LQ problem using the methods described above.

```
import numpy as np
import scipy.stats as spst
import scipy.linalg as la

class LQFilter:

    def __init__(self, d, h, y_m, r=None, h_eps=None, β=None):
        """

        Parameters
        ----------
            d : list or numpy.array (1-D or a 2-D column vector)
                    The order of the coefficients: [d_0, d_1, ..., d_m]
            h : scalar
                    Parameter of the objective function (corresponding to the
                    quadratic term)
            y_m : list or numpy.array (1-D or a 2-D column vector)
                    Initial conditions for y
            r : list or numpy.array (1-D or a 2-D column vector)
                    The order of the coefficients: [r_0, r_1, ..., r_k]
                    (optional, if not defined -> deterministic problem)
            β : scalar
                    Discount factor (optional, default value is one)
        """

        self.h = h
        self.d = np.asarray(d)
        self.m = self.d.shape[0] - 1

        self.y_m = np.asarray(y_m)

        if self.m == self.y_m.shape[0]:
            self.y_m = self.y_m.reshape(self.m, 1)
        else:
            raise ValueError("y_m must be of length m = {self.m:d}")

        #---------------------------------------------
        # Define the coefficients of ϕ upfront
        #---------------------------------------------
        ϕ = np.zeros(2 * self.m + 1)
        for i in range(- self.m, self.m + 1):
            ϕ[self.m - i] = np.sum(np.diag(self.d.reshape(self.m + 1, 1) \
                                            @ self.d.reshape(1, self.m + 1),
                                            k=-i
                                            )
                                    )
        ϕ[self.m] = ϕ[self.m] + self.h
        self.ϕ = ϕ

        #-----------------------------------------------------
        # If r is given calculate the vector ϕ_r
        #-----------------------------------------------------
        if r is None:
```

(continues on next page)

(continued from previous page)

```
                pass
            else:
                self.r = np.asarray(r)
                self.k = self.r.shape[0] - 1
                ϕ_r = np.zeros(2 * self.k + 1)
                for i in range(- self.k, self.k + 1):
                    ϕ_r[self.k - i] = np.sum(np.diag(self.r.reshape(self.k + 1, 1) \
                                                     @ self.r.reshape(1, self.k + 1),
                                                     k=-i
                                                     )
                                            )
                if h_eps is None:
                    self.ϕ_r = ϕ_r
                else:
                    ϕ_r[self.k] = ϕ_r[self.k] + h_eps
                    self.ϕ_r = ϕ_r

            #------------------------------------------------------
            # If β is given, define the transformed variables
            #------------------------------------------------------
            if β is None:
                self.β = 1
            else:
                self.β = β
                self.d = self.β**(np.arange(self.m + 1)/2) * self.d
                self.y_m = self.y_m * (self.β**(- np.arange(1, self.m + 1)/2)) \
                                    .reshape(self.m, 1)

        def construct_W_and_Wm(self, N):
            """
            This constructs the matrices W and W_m for a given number of periods N
            """

            m = self.m
            d = self.d

            W = np.zeros((N + 1, N + 1))
            W_m = np.zeros((N + 1, m))

            #----------------------------------------
            # Terminal conditions
            #----------------------------------------

            D_m1 = np.zeros((m + 1, m + 1))
            M = np.zeros((m + 1, m))

            # (1) Constuct the D_{m+1} matrix using the formula

            for j in range(m + 1):
                for k in range(j, m + 1):
                    D_m1[j, k] = d[:j + 1] @ d[k - j: k + 1]

            # Make the matrix symmetric
            D_m1 = D_m1 + D_m1.T - np.diag(np.diag(D_m1))

            # (2) Construct the M matrix using the entries of D_m1
```

(continues on next page)

(continued from previous page)

```
        for j in range(m):
            for i in range(j + 1, m + 1):
                M[i, j] = D_m1[i - j - 1, m]

        #----------------------------------------------
        # Euler equations for t = 0, 1, ..., N-(m+1)
        #----------------------------------------------
        ϕ = self.ϕ

        W[:(m + 1), :(m + 1)] = D_m1 + self.h * np.eye(m + 1)
        W[:(m + 1), (m + 1):(2 * m + 1)] = M

        for i, row in enumerate(np.arange(m + 1, N + 1 - m)):
            W[row, (i + 1):(2 * m + 2 + i)] = ϕ

        for i in range(1, m + 1):
            W[N - m + i, -(2 * m + 1 - i):] = ϕ[:-i]

        for i in range(m):
            W_m[N - i, :(m - i)] = ϕ[(m + 1 + i):]

        return W, W_m

    def roots_of_characteristic(self):
        """
        This function calculates z_0 and the 2m roots of the characteristic
        equation associated with the Euler equation (1.7)

        Note:
        ------
        numpy.poly1d(roots, True) defines a polynomial using its roots that can
        be evaluated at any point. If x_1, x_2, ... , x_m are the roots then
            p(x) = (x - x_1)(x - x_2)...(x - x_m)
        """
        m = self.m
        ϕ = self.ϕ

        # Calculate the roots of the 2m-polynomial
        roots = np.roots(ϕ)
        # Sort the roots according to their length (in descending order)
        roots_sorted = roots[np.argsort(abs(roots))[::-1]]

        z_0 = ϕ.sum() / np.poly1d(roots, True)(1)
        z_1_to_m = roots_sorted[:m]     # We need only those outside the unit circle

        λ = 1 / z_1_to_m

        return z_1_to_m, z_0, λ

    def coeffs_of_c(self):
        '''
        This function computes the coefficients {c_j, j = 0, 1, ..., m} for
                c(z) = sum_{j = 0}^{m} c_j z^j

        Based on the expression (1.9). The order is
```

(continues on next page)

(continued from previous page)

```
            c_coeffs = [c_0, c_1, ..., c_{m-1}, c_m]
        '''
        z_1_to_m, z_0 = self.roots_of_characteristic()[:2]

        c_0 = (z_0 * np.prod(z_1_to_m).real * (- 1)**self.m)**(.5)
        c_coeffs = np.poly1d(z_1_to_m, True).c * z_0 / c_0

        return c_coeffs[::-1]

    def solution(self):
        """
        This function calculates {λ_j, j=1,...,m} and {A_j, j=1,...,m}
        of the expression (1.15)
        """
        λ = self.roots_of_characteristic()[2]
        c_0 = self.coeffs_of_c()[-1]

        A = np.zeros(self.m, dtype=complex)
        for j in range(self.m):
            denom = 1 - λ/λ[j]
            A[j] = c_0**(-2) / np.prod(denom[np.arange(self.m) != j])

        return λ, A

    def construct_V(self, N):
        '''
        This function constructs the covariance matrix for x^N (see section 6)
        for a given period N
        '''
        V = np.zeros((N, N))
        ϕ_r = self.ϕ_r

        for i in range(N):
            for j in range(N):
                if abs(i-j) <= self.k:
                    V[i, j] = ϕ_r[self.k + abs(i-j)]

        return V

    def simulate_a(self, N):
        """
        Assuming that the u's are normal, this method draws a random path
        for x^N
        """
        V = self.construct_V(N + 1)
        d = spst.multivariate_normal(np.zeros(N + 1), V)

        return d.rvs()

    def predict(self, a_hist, t):
        """
        This function implements the prediction formula discussed in section 6 (1.59)
        It takes a realization for a^N, and the period in which the prediction is
        formed

        Output:  E[abar | a_t, a_{t-1}, ..., a_1, a_0]
```

(continues on next page)

(continued from previous page)

```
        """

        N = np.asarray(a_hist).shape[0] - 1
        a_hist = np.asarray(a_hist).reshape(N + 1, 1)
        V = self.construct_V(N + 1)

        aux_matrix = np.zeros((N + 1, N + 1))
        aux_matrix[:(t + 1), :(t + 1)] = np.eye(t + 1)
        L = la.cholesky(V).T
        Ea_hist = la.inv(L) @ aux_matrix @ L @ a_hist

        return Ea_hist

    def optimal_y(self, a_hist, t=None):
        """
        - if t is NOT given it takes a_hist (list or numpy.array) as a
          deterministic a_t
        - if t is given, it solves the combined control prediction problem
          (section 7)(by default, t == None -> deterministic)

        for a given sequence of a_t (either deterministic or a particular
        realization), it calculates the optimal y_t sequence using the method
        of the lecture

        Note:
        ------
        scipy.linalg.lu normalizes L, U so that L has unit diagonal elements
        To make things consistent with the lecture, we need an auxiliary
        diagonal matrix D which renormalizes L and U
        """

        N = np.asarray(a_hist).shape[0] - 1
        W, W_m = self.construct_W_and_Wm(N)

        L, U = la.lu(W, permute_l=True)
        D = np.diag(1 / np.diag(U))
        U = D @ U
        L = L @ np.diag(1 / np.diag(D))

        J = np.fliplr(np.eye(N + 1))

        if t is None:   # If the problem is deterministic

            a_hist = J @ np.asarray(a_hist).reshape(N + 1, 1)

            #--------------------------------------------
            # Transform the 'a' sequence if β is given
            #--------------------------------------------
            if self.β != 1:
                a_hist =  a_hist * (self.β**(np.arange(N + 1) / 2))[::-1] \
                                    .reshape(N + 1, 1)

            a_bar = a_hist - W_m @ self.y_m             # a_bar from the lecture
            Uy = np.linalg.solve(L, a_bar)              # U @ y_bar = L^{-1}
            y_bar = np.linalg.solve(U, Uy)              # y_bar = U^{-1}L^{-1}
```

(continues on next page)

(continued from previous page)

```
            # Reverse the order of y_bar with the matrix J
            J = np.fliplr(np.eye(N + self.m + 1))
            # y_hist : concatenated y_m and y_bar
            y_hist = J @ np.vstack([y_bar, self.y_m])

            #--------------------------------------------
            # Transform the optimal sequence back if β is given
            #--------------------------------------------
            if self.β != 1:
                y_hist = y_hist * (self.β**(- np.arange(-self.m, N + 1)/2)) \
                                    .reshape(N + 1 + self.m, 1)

            return y_hist, L, U, y_bar

        else:           # If the problem is stochastic and we look at it

            Ea_hist = self.predict(a_hist, t).reshape(N + 1, 1)
            Ea_hist = J @ Ea_hist

            a_bar = Ea_hist - W_m @ self.y_m           # a_bar from the lecture
            Uy = np.linalg.solve(L, a_bar)             # U @ y_bar = L^{-1}
            y_bar = np.linalg.solve(U, Uy)             # y_bar = U^{-1}L^{-1}

            # Reverse the order of y_bar with the matrix J
            J = np.fliplr(np.eye(N + self.m + 1))
            # y_hist : concatenated y_m and y_bar
            y_hist = J @ np.vstack([y_bar, self.y_m])

            return y_hist, L, U, y_bar
```

5.6.1 Example

In this application, we'll have one lag, with

$$d(L)y_t = \gamma(I - L)y_t = \gamma(y_t - y_{t-1})$$

Suppose for the moment that $\gamma = 0$.

Then the intertemporal component of the LQ problem disappears, and the agent simply wants to maximize $a_t y_t - h y_t^2/2$ in each period.

This means that the agent chooses $y_t = a_t/h$.

In the following we'll set $h = 1$, so that the agent just wants to track the $\{a_t\}$ process.

However, as we increase γ, the agent gives greater weight to a smooth time path.

Hence $\{y_t\}$ evolves as a smoothed version of $\{a_t\}$.

The $\{a_t\}$ sequence we'll choose as a stationary cyclic process plus some white noise.

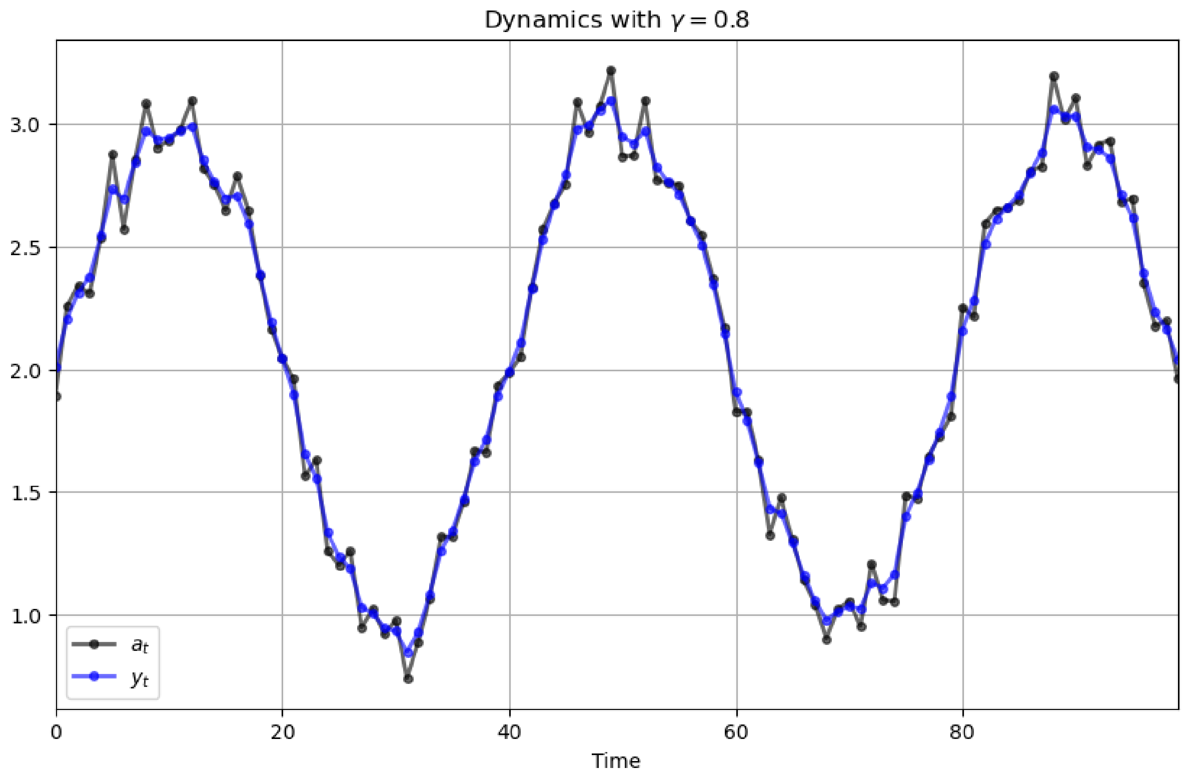
Here's some code that generates a plot when $\gamma = 0.8$

```
# Set seed and generate a_t sequence
np.random.seed(123)
n = 100
```

(continues on next page)

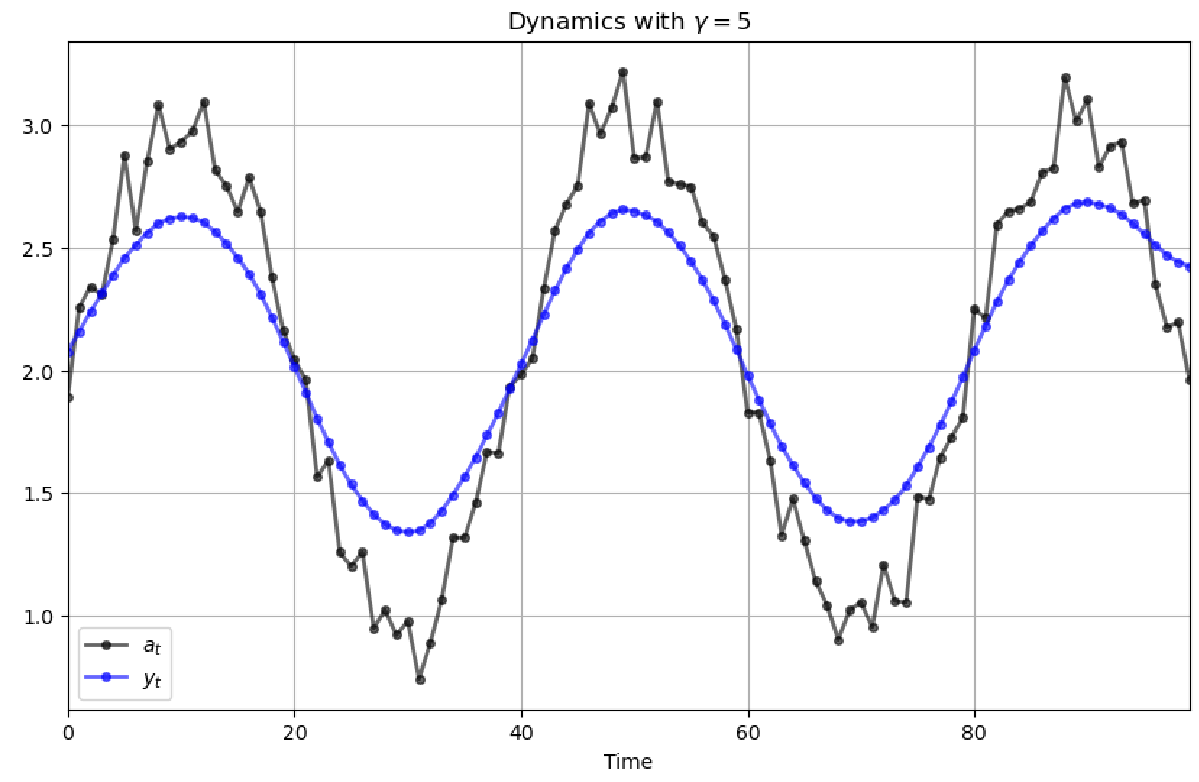
(continued from previous page)

```
a_seq = np.sin(np.linspace(0, 5 * np.pi, n)) + 2 + 0.1 * np.random.randn(n)

def plot_simulation(γ=0.8, m=1, h=1, y_m=2):

    d = γ * np.asarray([1, -1])
    y_m = np.asarray(y_m).reshape(m, 1)

    testlq = LQFilter(d, h, y_m)
    y_hist, L, U, y = testlq.optimal_y(a_seq)
    y = y[::-1]  # Reverse y

    # Plot simulation results

    fig, ax = plt.subplots(figsize=(10, 6))
    p_args = {'lw' : 2, 'alpha' : 0.6}
    time = range(len(y))
    ax.plot(time, a_seq / h, 'k-o', ms=4, lw=2, alpha=0.6, label='$a_t$')
    ax.plot(time, y, 'b-o', ms=4, lw=2, alpha=0.6, label='$y_t$')
    ax.set(title=rf'Dynamics with $\gamma = {γ}$',
           xlabel='Time',
           xlim=(0, max(time))
          )
    ax.legend()
    ax.grid()
    plt.show()

plot_simulation()
```



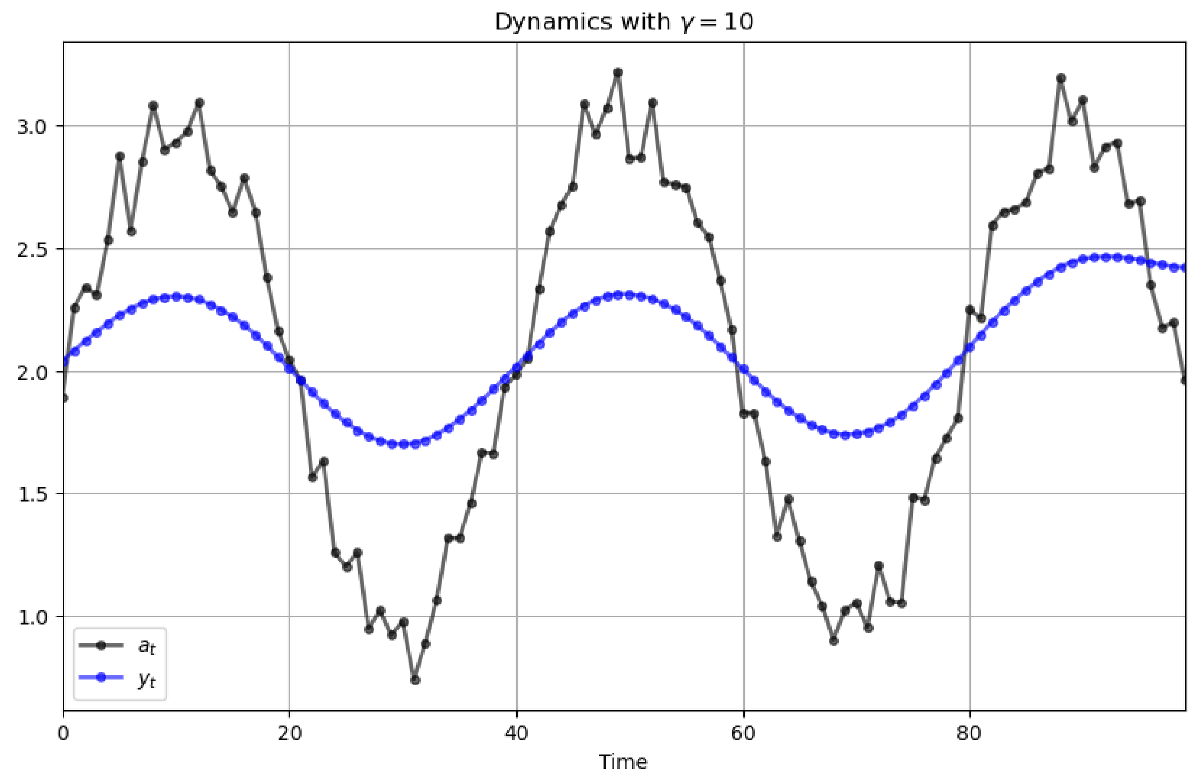
Here's what happens when we change γ to 5.0

```
plot_simulation(γ=5)
```



And here's $\gamma = 10$

```
plot_simulation(γ=10)
```



5.7 Exercises

Exercise 5.7.1

Consider solving a discounted version ($\beta < 1$) of problem (5.1), as follows.

Convert (5.1) to the undiscounted problem (5.22).

Let the solution of (5.22) in feedback form be

$$(1 - \tilde{\lambda}_1 L) \cdots (1 - \tilde{\lambda}_m L)\tilde{y}_t = \sum_{j=1}^{m} \tilde{A}_j \sum_{k=0}^{\infty} \tilde{\lambda}_j^k \tilde{a}_{t+k}$$

or

$$\tilde{y}_t = \tilde{f}_1 \tilde{y}_{t-1} + \cdots + \tilde{f}_m \tilde{y}_{t-m} + \sum_{j=1}^{m} \tilde{A}_j \sum_{k=0}^{\infty} \tilde{\lambda}_j^k \tilde{a}_{t+k} \tag{5.25}$$

Here

- $h + \tilde{d}(z^{-1})\tilde{d}(z) = \tilde{c}(z^{-1})\tilde{c}(z)$
- $\tilde{c}(z) = [(-1)^m \tilde{z}_0 \tilde{z}_1 \cdots \tilde{z}_m]^{1/2}(1 - \tilde{\lambda}_1 z) \cdots (1 - \tilde{\lambda}_m z)$

where the $\tilde{z}_j$ are the zeros of $h + \tilde{d}(z^{-1})\,\tilde{d}(z)$.

Prove that (5.25) implies that the solution for y_t in feedback form is

$$y_t = f_1 y_{t-1} + \ldots + f_m y_{t-m} + \sum_{j=1}^{m} A_j \sum_{k=0}^{\infty} \beta^k \lambda_j^k a_{t+k}$$

where $f_j = \tilde{f}_j \beta^{-j/2}$, $A_j = \tilde{A}_j$, and $\lambda_j = \tilde{\lambda}_j \beta^{-1/2}$.

Exercise 5.7.2

Solve the optimal control problem, maximize

$$\sum_{t=0}^{2} \left\{ a_t y_t - \frac{1}{2} [(1-2L) y_t]^2 \right\}$$

subject to y_{-1} given, and $\{a_t\}$ a known bounded sequence.

Express the solution in the "feedback form" (5.20), giving numerical values for the coefficients.

Make sure that the boundary conditions (5.5) are satisfied.

Note: This problem differs from the problem in the text in one important way: instead of $h > 0$ in (5.1), $h = 0$. This has an important influence on the solution.

Exercise 5.7.3

Solve the infinite time-optimal control problem to maximize

$$\lim_{N \to \infty} \sum_{t=0}^{N} -\frac{1}{2} [(1-2L) y_t]^2,$$

subject to y_{-1} given. Prove that the solution is

$$y_t = 2 y_{t-1} = 2^{t+1} y_{-1} \qquad t > 0$$

Exercise 5.7.4

Solve the infinite time problem, to maximize

$$\lim_{N \to \infty} \sum_{t=0}^{N} (.0000001)\, y_t^2 - \frac{1}{2} [(1-2L) y_t]^2$$

subject to y_{-1} given. Prove that the solution $y_t = 2 y_{t-1}$ violates condition (5.12), and so is not optimal.

Prove that the optimal solution is approximately $y_t = .5 y_{t-1}$.

CHAPTER

SIX

CLASSICAL PREDICTION AND FILTERING WITH LINEAR ALGEBRA

6.1 Overview

This is a sequel to the earlier lecture *Classical Control with Linear Algebra*.

That lecture used linear algebra – in particular, the LU decomposition – to formulate and solve a class of linear-quadratic optimal control problems.

In this lecture, we'll be using a closely related decomposition, the Cholesky decomposition, to solve linear prediction and filtering problems.

We exploit the useful fact that there is an intimate connection between two superficially different classes of problems:

- deterministic linear-quadratic (LQ) optimal control problems
- linear least squares prediction and filtering problems

The first class of problems involves no randomness, while the second is all about randomness.

Nevertheless, essentially the same mathematics solves both types of problem.

This connection, which is often termed "duality," is present whether one uses "classical" or "recursive" solution procedures.

In fact, we saw duality at work earlier when we formulated control and prediction problems recursively in lectures LQ dynamic programming problems, A first look at the Kalman filter, and The permanent income model.

A useful consequence of duality is that

- With every LQ control problem, there is implicitly affiliated a linear least squares prediction or filtering problem.
- With every linear least squares prediction or filtering problem there is implicitly affiliated a LQ control problem.

An understanding of these connections has repeatedly proved useful in cracking interesting applied problems.

For example, Sargent [Sargent, 1987] [chs. IX, XIV] and Hansen and Sargent [Hansen and Sargent, 1980] formulated and solved control and filtering problems using z-transform methods.

In this lecture, we begin to investigate these ideas by using mostly elementary linear algebra.

This is the main purpose and focus of the lecture.

However, after showing matrix algebra formulas, we'll summarize classic infinite-horizon formulas built on z-transform and lag operator methods.

And we'll occasionally refer to some of these formulas from the infinite dimensional problems as we present the finite time formulas and associated linear algebra.

We'll start with the following standard import:

```
import numpy as np
```

6.1.1 References

Useful references include [Whittle, 1963], [Hansen and Sargent, 1980], [Orfanidis, 1988], [Athanasios and Pillai, 1991], and [Muth, 1960].

6.2 Finite Dimensional Prediction

Let $(x_1, x_2, \dots, x_T)' = x$ be a $T \times 1$ vector of random variables with mean $\mathbb{E}x = 0$ and covariance matrix $\mathbb{E}xx' = V$.

Here V is a $T \times T$ positive definite matrix.

The i, j component $Ex_i x_j$ of V is the **inner product** between x_i and x_j.

We regard the random variables as being ordered in time so that x_t is thought of as the value of some economic variable at time t.

For example, x_t could be generated by the random process described by the Wold representation presented in equation (6.16) in the section below on infinite dimensional prediction and filtering.

In that case, V_{ij} is given by the coefficient on $z^{|i-j|}$ in the expansion of $g_x(z) = d(z)\,d(z^{-1}) + h$, which equals $h + \sum_{k=0}^{\infty} d_k d_{k+|i-j|}$.

We want to construct j step ahead linear least squares predictors of the form

$$\hat{\mathbb{E}}\left[x_T | x_{T-j}, x_{T-j+1}, \dots, x_1\right]$$

where $\hat{\mathbb{E}}$ is the linear least squares projection operator.

(Sometimes $\hat{\mathbb{E}}$ is called the wide-sense expectations operator)

To find linear least squares predictors it is helpful first to construct a $T \times 1$ vector ε of random variables that form an orthonormal basis for the vector of random variables x.

The key insight here comes from noting that because the covariance matrix V of x is a positive definite and symmetric, there exists a (Cholesky) decomposition of V such that

$$V = L^{-1}(L^{-1})'$$

and

$$L\,V\,L' = I$$

where L and L^{-1} are both lower triangular.

Form the $T \times 1$ random vector $\varepsilon = Lx$.

The random vector ε is an orthonormal basis for x because

- L is nonsingular
- $\mathbb{E}\,\varepsilon\,\varepsilon' = L\mathbb{E}xx'L' = I$
- $x = L^{-1}\varepsilon$

It is enlightening to write out and interpret the equations $Lx = \varepsilon$ and $L^{-1}\varepsilon = x$.

First, we'll write $Lx = \varepsilon$

$$\begin{aligned} L_{11}x_1 &= \varepsilon_1 \\ L_{21}x_1 + L_{22}x_2 &= \varepsilon_2 \\ &\vdots \\ L_{T1}\,x_1\,\ldots + L_{TT}x_T &= \varepsilon_T \end{aligned} \tag{6.1}$$

or

$$\sum_{j=0}^{t-1} L_{t,t-j}\,x_{t-j} = \varepsilon_t, \quad t = 1,\,2,\ldots T \tag{6.2}$$

Next, we write $L^{-1}\varepsilon = x$

$$\begin{aligned} x_1 &= L_{11}^{-1}\varepsilon_1 \\ x_2 &= L_{22}^{-1}\varepsilon_2 + L_{21}^{-1}\varepsilon_1 \\ \vdots \\ x_T &= L_{TT}^{-1}\varepsilon_T + L_{T,T-1}^{-1}\varepsilon_{T-1}\,\ldots + L_{T,1}^{-1}\varepsilon_1 \end{aligned}, \tag{6.3}$$

or

$$x_t = \sum_{j=0}^{t-1} L_{t,t-j}^{-1}\,\varepsilon_{t-j} \tag{6.4}$$

where $L_{i,j}^{-1}$ denotes the i, j element of L^{-1}.

From (6.2), it follows that ε_t is in the linear subspace spanned by $x_t,\,x_{t-1},\ldots,\,x_1$.

From (6.4) it follows that that x_t is in the linear subspace spanned by $\varepsilon_t,\,\varepsilon_{t-1},\ldots,\varepsilon_1$.

Equation (6.2) forms a sequence of **autoregressions** that for $t = 1,\ldots,T$ express x_t as linear functions of $x_s, s = 1,\ldots,t-1$ and a random variable $(L_{t,t})^{-1}\varepsilon_t$ that is orthogonal to each componenent of $x_s, s = 1,\ldots,t-1$.

(Here $(L_{t,t})^{-1}$ denotes the reciprocal of $L_{t,t}$ while $L_{t,t}^{-1}$ denotes the t, t element of L^{-1}).

The equivalence of the subspaces spanned by $\varepsilon_t,\ldots,\varepsilon_1$ and $x_t,\ldots,x_1$ means that for $t-1 \geq m \geq 1$

$$\hat{\mathbb{E}}[x_t \mid x_{t-m},\,x_{t-m-1},\ldots,x_1] = \hat{\mathbb{E}}[x_t \mid \varepsilon_{t-m},\varepsilon_{t-m-1},\ldots,\varepsilon_1] \tag{6.5}$$

To proceed, it is useful to drill down and note that for $t-1 \geq m \geq 1$ we can rewrite (6.4) in the form of the **moving average representation**

$$x_t = \sum_{j=0}^{m-1} L_{t,t-j}^{-1}\,\varepsilon_{t-j} + \sum_{j=m}^{t-1} L_{t,t-j}^{-1}\,\varepsilon_{t-j} \tag{6.6}$$

Representation (6.6) is an orthogonal decomposition of x_t into a part $\sum_{j=m}^{t-1} L_{t,t-j}^{-1}\,\varepsilon_{t-j}$ that lies in the space spanned by $[x_{t-m},\,x_{t-m+1},\,\ldots,x_1]$ and an orthogonal component $\sum_{j=m}^{t-1} L_{t,t-j}^{-1}\,\varepsilon_{t-j}$ that does not lie in that space but instead in a linear space knowns as its **orthogonal complement**.

It follows that

$$\hat{\mathbb{E}}[x_t \mid x_{t-m},\,x_{t-m-1},\ldots,x_1] = \sum_{j=0}^{m-1} L_{t,t-j}^{-1}\,\varepsilon_{t-j}$$

6.2.1 Implementation

Here's the code that computes solutions to LQ control and filtering problems using the methods described here and in *Classical Control with Linear Algebra*.

```
import numpy as np
import scipy.stats as spst
import scipy.linalg as la

class LQFilter:

    def __init__(self, d, h, y_m, r=None, h_eps=None, β=None):
        """

        Parameters
        ----------
            d : list or numpy.array (1-D or a 2-D column vector)
                    The order of the coefficients: [d_0, d_1, ..., d_m]
            h : scalar
                    Parameter of the objective function (corresponding to the
                    quadratic term)
            y_m : list or numpy.array (1-D or a 2-D column vector)
                    Initial conditions for y
            r : list or numpy.array (1-D or a 2-D column vector)
                    The order of the coefficients: [r_0, r_1, ..., r_k]
                    (optional, if not defined -> deterministic problem)
            β : scalar
                    Discount factor (optional, default value is one)
        """

        self.h = h
        self.d = np.asarray(d)
        self.m = self.d.shape[0] - 1

        self.y_m = np.asarray(y_m)

        if self.m == self.y_m.shape[0]:
            self.y_m = self.y_m.reshape(self.m, 1)
        else:
            raise ValueError("y_m must be of length m = {self.m:d}")

        #---------------------------------------------
        # Define the coefficients of ϕ upfront
        #---------------------------------------------
        ϕ = np.zeros(2 * self.m + 1)
        for i in range(- self.m, self.m + 1):
            ϕ[self.m - i] = np.sum(np.diag(self.d.reshape(self.m + 1, 1) \
                                           @ self.d.reshape(1, self.m + 1),
                                           k=-i
                                           )
                                   )
        ϕ[self.m] = ϕ[self.m] + self.h
        self.ϕ = ϕ

        #-----------------------------------------------------
        # If r is given calculate the vector ϕ_r
        #-----------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
        if r is None:
            pass
        else:
            self.r = np.asarray(r)
            self.k = self.r.shape[0] - 1
            ϕ_r = np.zeros(2 * self.k + 1)
            for i in range(- self.k, self.k + 1):
                ϕ_r[self.k - i] = np.sum(np.diag(self.r.reshape(self.k + 1, 1) \
                                                 @ self.r.reshape(1, self.k + 1),
                                                 k=-i
                                                 )
                                        )
            if h_eps is None:
                self.ϕ_r = ϕ_r
            else:
                ϕ_r[self.k] = ϕ_r[self.k] + h_eps
                self.ϕ_r = ϕ_r

        #------------------------------------------------------
        # If β is given, define the transformed variables
        #------------------------------------------------------
        if β is None:
            self.β = 1
        else:
            self.β = β
            self.d = self.β**(np.arange(self.m + 1)/2) * self.d
            self.y_m = self.y_m * (self.β**(- np.arange(1, self.m + 1)/2)) \
                                 .reshape(self.m, 1)

    def construct_W_and_Wm(self, N):
        """
        This constructs the matrices W and W_m for a given number of periods N
        """

        m = self.m
        d = self.d

        W = np.zeros((N + 1, N + 1))
        W_m = np.zeros((N + 1, m))

        #---------------------------------------
        # Terminal conditions
        #---------------------------------------

        D_m1 = np.zeros((m + 1, m + 1))
        M = np.zeros((m + 1, m))

        # (1) Constuct the D_{m+1} matrix using the formula

        for j in range(m + 1):
            for k in range(j, m + 1):
                D_m1[j, k] = d[:j + 1] @ d[k - j: k + 1]

        # Make the matrix symmetric
        D_m1 = D_m1 + D_m1.T - np.diag(np.diag(D_m1))
```

(continues on next page)

(continued from previous page)

```
        # (2) Construct the M matrix using the entries of D_m1

        for j in range(m):
            for i in range(j + 1, m + 1):
                M[i, j] = D_m1[i - j - 1, m]

        #----------------------------------------------
        # Euler equations for t = 0, 1, ..., N-(m+1)
        #----------------------------------------------
        ϕ = self.ϕ

        W[:(m + 1), :(m + 1)] = D_m1 + self.h * np.eye(m + 1)
        W[:(m + 1), (m + 1):(2 * m + 1)] = M

        for i, row in enumerate(np.arange(m + 1, N + 1 - m)):
            W[row, (i + 1):(2 * m + 2 + i)] = ϕ

        for i in range(1, m + 1):
            W[N - m + i, -(2 * m + 1 - i):] = ϕ[:-i]

        for i in range(m):
            W_m[N - i, :(m - i)] = ϕ[(m + 1 + i):]

        return W, W_m

    def roots_of_characteristic(self):
        """
        This function calculates z_0 and the 2m roots of the characteristic
        equation associated with the Euler equation (1.7)

        Note:
        ------
        numpy.poly1d(roots, True) defines a polynomial using its roots that can
        be evaluated at any point. If x_1, x_2, ... , x_m are the roots then
            p(x) = (x - x_1)(x - x_2)...(x - x_m)
        """
        m = self.m
        ϕ = self.ϕ

        # Calculate the roots of the 2m-polynomial
        roots = np.roots(ϕ)
        # Sort the roots according to their length (in descending order)
        roots_sorted = roots[np.argsort(abs(roots))[::-1]]

        z_0 = ϕ.sum() / np.poly1d(roots, True)(1)
        z_1_to_m = roots_sorted[:m]     # We need only those outside the unit circle

        λ = 1 / z_1_to_m

        return z_1_to_m, z_0, λ

    def coeffs_of_c(self):
        '''
        This function computes the coefficients {c_j, j = 0, 1, ..., m} for
                c(z) = sum_{j = 0}^{m} c_j z^j
```

(continues on next page)

(continued from previous page)

```
        Based on the expression (1.9). The order is
            c_coeffs = [c_0, c_1, ..., c_{m-1}, c_m]
        '''
        z_1_to_m, z_0 = self.roots_of_characteristic()[:2]

        c_0 = (z_0 * np.prod(z_1_to_m).real * (- 1)**self.m)**(.5)
        c_coeffs = np.poly1d(z_1_to_m, True).c * z_0 / c_0

        return c_coeffs[::-1]

    def solution(self):
        """
        This function calculates {λ_j, j=1,...,m} and {A_j, j=1,...,m}
        of the expression (1.15)
        """
        λ = self.roots_of_characteristic()[2]
        c_0 = self.coeffs_of_c()[-1]

        A = np.zeros(self.m, dtype=complex)
        for j in range(self.m):
            denom = 1 - λ/λ[j]
            A[j] = c_0**(-2) / np.prod(denom[np.arange(self.m) != j])

        return λ, A

    def construct_V(self, N):
        '''
        This function constructs the covariance matrix for x^N (see section 6)
        for a given period N
        '''
        V = np.zeros((N, N))
        ϕ_r = self.ϕ_r

        for i in range(N):
            for j in range(N):
                if abs(i-j) <= self.k:
                    V[i, j] = ϕ_r[self.k + abs(i-j)]

        return V

    def simulate_a(self, N):
        """
        Assuming that the u's are normal, this method draws a random path
        for x^N
        """
        V = self.construct_V(N + 1)
        d = spst.multivariate_normal(np.zeros(N + 1), V)

        return d.rvs()

    def predict(self, a_hist, t):
        """
        This function implements the prediction formula discussed in section 6 (1.59)
        It takes a realization for a^N, and the period in which the prediction is
        formed
```

(continues on next page)

(continued from previous page)

```python
        Output:  E[abar | a_t, a_{t-1}, ..., a_1, a_0]
        """

        N = np.asarray(a_hist).shape[0] - 1
        a_hist = np.asarray(a_hist).reshape(N + 1, 1)
        V = self.construct_V(N + 1)

        aux_matrix = np.zeros((N + 1, N + 1))
        aux_matrix[:(t + 1), :(t + 1)] = np.eye(t + 1)
        L = la.cholesky(V).T
        Ea_hist = la.inv(L) @ aux_matrix @ L @ a_hist

        return Ea_hist

    def optimal_y(self, a_hist, t=None):
        """
        - if t is NOT given it takes a_hist (list or numpy.array) as a
          deterministic a_t
        - if t is given, it solves the combined control prediction problem
          (section 7)(by default, t == None -> deterministic)

        for a given sequence of a_t (either deterministic or a particular
        realization), it calculates the optimal y_t sequence using the method
        of the lecture

        Note:
        ------
        scipy.linalg.lu normalizes L, U so that L has unit diagonal elements
        To make things consistent with the lecture, we need an auxiliary
        diagonal matrix D which renormalizes L and U
        """

        N = np.asarray(a_hist).shape[0] - 1
        W, W_m = self.construct_W_and_Wm(N)

        L, U = la.lu(W, permute_l=True)
        D = np.diag(1 / np.diag(U))
        U = D @ U
        L = L @ np.diag(1 / np.diag(D))

        J = np.fliplr(np.eye(N + 1))

        if t is None:   # If the problem is deterministic

            a_hist = J @ np.asarray(a_hist).reshape(N + 1, 1)

            #--------------------------------------------
            # Transform the 'a' sequence if β is given
            #--------------------------------------------
            if self.β != 1:
                a_hist =  a_hist * (self.β**(np.arange(N + 1) / 2))[::-1] \
                                    .reshape(N + 1, 1)

            a_bar = a_hist - W_m @ self.y_m            # a_bar from the lecture
            Uy = np.linalg.solve(L, a_bar)             # U @ y_bar = L^{-1}
            y_bar = np.linalg.solve(U, Uy)             # y_bar = U^{-1}L^{-1}
```

(continues on next page)

(continued from previous page)

```
            # Reverse the order of y_bar with the matrix J
            J = np.fliplr(np.eye(N + self.m + 1))
            # y_hist : concatenated y_m and y_bar
            y_hist = J @ np.vstack([y_bar, self.y_m])

            #--------------------------------------------
            # Transform the optimal sequence back if β is given
            #--------------------------------------------
            if self.β != 1:
                y_hist = y_hist * (self.β**(- np.arange(-self.m, N + 1)/2)) \
                                    .reshape(N + 1 + self.m, 1)

            return y_hist, L, U, y_bar

        else:           # If the problem is stochastic and we look at it

            Ea_hist = self.predict(a_hist, t).reshape(N + 1, 1)
            Ea_hist = J @ Ea_hist

            a_bar = Ea_hist - W_m @ self.y_m           # a_bar from the lecture
            Uy = np.linalg.solve(L, a_bar)             # U @ y_bar = L^{-1}
            y_bar = np.linalg.solve(U, Uy)             # y_bar = U^{-1}L^{-1}

            # Reverse the order of y_bar with the matrix J
            J = np.fliplr(np.eye(N + self.m + 1))
            # y_hist : concatenated y_m and y_bar
            y_hist = J @ np.vstack([y_bar, self.y_m])

            return y_hist, L, U, y_bar
```

Let's use this code to tackle two interesting examples.

6.2.2 Example 1

Consider a stochastic process with moving average representation

$$x_t = (1 - 2L)\varepsilon_t$$

where ε_t is a serially uncorrelated random process with mean zero and variance unity.

If we were to use the tools associated with infinite dimensional prediction and filtering to be described below, we would use the Wiener-Kolmogorov formula (6.21) to compute the linear least squares forecasts $\mathbb{E}[x_{t+j} \mid x_t, x_{t-1}, \ldots]$, for $j = 1,\ 2$.

But we can do everything we want by instead using our finite dimensional tools and setting $d = r$, generating an instance of LQFilter, then invoking pertinent methods of LQFilter.

```
m = 1
y_m = np.asarray([.0]).reshape(m, 1)
d = np.asarray([1, -2])
r = np.asarray([1, -2])
h = 0.0
example = LQFilter(d, h, y_m, r=d)
```

The Wold representation is computed by `example.coeffs_of_c()`.

Let's check that it "flips roots" as required

```
example.coeffs_of_c()
```

```
array([ 2., -1.])
```

```
example.roots_of_characteristic()
```

```
(array([2.]), -2.0, array([0.5]))
```

Now let's form the covariance matrix of a time series vector of length N and put it in V.

Then we'll take a Cholesky decomposition of $V = L^{-1}L^{-1}$ and use it to form the vector of "moving average representations" $x = L^{-1}\varepsilon$ and the vector of "autoregressive representations" $Lx = \varepsilon$.

```
V = example.construct_V(N=5)
print(V)
```

```
[[ 5. -2.  0.  0.  0.]
 [-2.  5. -2.  0.  0.]
 [ 0. -2.  5. -2.  0.]
 [ 0.  0. -2.  5. -2.]
 [ 0.  0.  0. -2.  5.]]
```

Notice how the lower rows of the "moving average representations" are converging to the appropriate infinite history Wold representation to be described below when we study infinite horizon-prediction and filtering

```
Li = np.linalg.cholesky(V)
print(Li)
```

```
[[ 2.23606798  0.          0.          0.          0.        ]
 [-0.89442719  2.04939015  0.          0.          0.        ]
 [ 0.         -0.97590007  2.01186954  0.          0.        ]
 [ 0.          0.         -0.99410024  2.00293902  0.        ]
 [ 0.          0.          0.         -0.99853265  2.000733  ]]
```

Notice how the lower rows of the "autoregressive representations" are converging to the appropriate infinite-history autoregressive representation to be described below when we study infinite horizon-prediction and filtering

```
L = np.linalg.inv(Li)
print(L)
```

```
[[0.4472136  0.         0.         0.         0.        ]
 [0.19518001 0.48795004 0.         0.         0.        ]
 [0.09467621 0.23669053 0.49705012 0.         0.        ]
 [0.04698977 0.11747443 0.2466963  0.49926632 0.        ]
 [0.02345182 0.05862954 0.12312203 0.24917554 0.49981682]]
```

6.2.3 Example 2

Consider a stochastic process X_t with moving average representation

$$X_t = (1 - \sqrt{2}L^2)\varepsilon_t$$

where ε_t is a serially uncorrelated random process with mean zero and variance unity.

Let's find a Wold moving average representation for x_t that will prevail in the infinite-history context to be studied in detail below.

To do this, we'll use the Wiener-Kolomogorov formula (6.21) presented below to compute the linear least squares forecasts $\hat{\mathbb{E}}\left[X_{t+j} \mid X_{t-1}, \ldots\right]$ for $j = 1, 2, 3$.

We proceed in the same way as in example 1

```
m = 2
y_m = np.asarray([.0, .0]).reshape(m, 1)
d = np.asarray([1, 0, -np.sqrt(2)])
r = np.asarray([1, 0, -np.sqrt(2)])
h = 0.0
example = LQFilter(d, h, y_m, r=d)
example.coeffs_of_c()
```

```
array([ 1.41421356, -0.        , -1.        ])
```

```
example.roots_of_characteristic()
```

```
(array([ 1.18920712, -1.18920712]),
 -1.4142135623731122,
 array([ 0.84089642, -0.84089642]))
```

```
V = example.construct_V(N=8)
print(V)
```

```
[[ 3.          0.         -1.41421356  0.          0.          0.
   0.          0.        ]
 [ 0.          3.          0.         -1.41421356  0.          0.
   0.          0.        ]
 [-1.41421356  0.          3.          0.         -1.41421356  0.
   0.          0.        ]
 [ 0.         -1.41421356  0.          3.          0.         -1.41421356
   0.          0.        ]
 [ 0.          0.         -1.41421356  0.          3.          0.
  -1.41421356  0.        ]
 [ 0.          0.          0.         -1.41421356  0.          3.
   0.         -1.41421356]
 [ 0.          0.          0.          0.         -1.41421356  0.
   3.          0.        ]
 [ 0.          0.          0.          0.          0.         -1.41421356
   0.          3.        ]]
```

```
Li = np.linalg.cholesky(V)
print(Li[-3:, :])
```

```
[[ 0.          0.          0.         -0.9258201   0.          1.46385011
   0.          0.        ]
 [ 0.          0.          0.          0.         -0.96609178  0.
   1.43759058  0.        ]
 [ 0.          0.          0.          0.          0.         -0.96609178
   0.          1.43759058]]
```

```python
L = np.linalg.inv(Li)
print(L)
```

```
[[0.57735027 0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.57735027 0.         0.         0.         0.
  0.         0.        ]
 [0.3086067  0.         0.65465367 0.         0.         0.
  0.         0.        ]
 [0.         0.3086067  0.         0.65465367 0.         0.
  0.         0.        ]
 [0.19518001 0.         0.41403934 0.         0.68313005 0.
  0.         0.        ]
 [0.         0.19518001 0.         0.41403934 0.         0.68313005
  0.         0.        ]
 [0.13116517 0.         0.27824334 0.         0.45907809 0.
  0.69560834 0.        ]
 [0.         0.13116517 0.         0.27824334 0.         0.45907809
  0.         0.69560834]]
```

6.2.4 Prediction

It immediately follows from the “orthogonality principle” of least squares (see [Athanasios and Pillai, 1991] or [Sargent, 1987] [ch. X]) that

$$\begin{aligned}\hat{\mathbb{E}}[x_t \mid x_{t-m}, x_{t-m+1}, \dots x_1] &= \sum_{j=m}^{t-1} L^{-1}_{t,t-j}\, \varepsilon_{t-j} \\ &= [L^{-1}_{t,1}\, L^{-1}_{t,2}, \dots, L^{-1}_{t,t-m}\; 0\, 0 \dots 0] L\, x\end{aligned} \tag{6.7}$$

This can be interpreted as a finite-dimensional version of the Wiener-Kolmogorov m-step ahead prediction formula.

We can use (6.7) to represent the linear least squares projection of the vector x conditioned on the first s observations $[x_s, x_{s-1} \dots, x_1]$.

We have

$$\hat{\mathbb{E}}[x \mid x_s, x_{s-1}, \dots, x_1] = L^{-1} \begin{bmatrix} I_s & 0 \\ 0 & 0_{(t-s)} \end{bmatrix} Lx \tag{6.8}$$

This formula will be convenient in representing the solution of control problems under uncertainty.

Equation (6.4) can be recognized as a finite dimensional version of a moving average representation.

Equation (6.2) can be viewed as a finite dimension version of an autoregressive representation.

Notice that even if the x_t process is covariance stationary, so that V is such that V_{ij} depends only on $|i-j|$, the coefficients in the moving average representation are time-dependent, there being a different moving average for each t.

If x_t is a covariance stationary process, the last row of L^{-1} converges to the coefficients in the Wold moving average representation for $\{x_t\}$ as $T \to \infty$.

Further, if x_t is covariance stationary, for fixed k and $j > 0$, $L^{-1}_{T,T-j}$ converges to $L^{-1}_{T-k,T-k-j}$ as $T \to \infty$.

That is, the "bottom" rows of L^{-1} converge to each other and to the Wold moving average coefficients as $T \to \infty$.

This last observation gives one simple and widely-used practical way of forming a finite T approximation to a Wold moving average representation.

First, form the covariance matrix $\mathbb{E}xx' = V$, then obtain the Cholesky decomposition $L^{-1}L^{-1'}$ of V, which can be accomplished quickly on a computer.

The last row of L^{-1} gives the approximate Wold moving average coefficients.

This method can readily be generalized to multivariate systems.

6.3 Combined Finite Dimensional Control and Prediction

Consider the finite-dimensional control problem, maximize

$$\mathbb{E} \sum_{t=0}^{N} \left\{ a_t y_t - \frac{1}{2} h y_t^2 - \frac{1}{2} [d(L) y_t]^2 \right\}, \qquad h > 0$$

where $d(L) = d_0 + d_1 L + \ldots + d_m L^m$, L is the lag operator, $\bar{a} = [a_N, a_{N-1} \ldots, a_1, a_0]'$ a random vector with mean zero and $\mathbb{E}\,\bar{a}\bar{a}' = V$.

The variables $y_{-1}, \ldots, y_{-m}$ are given.

Maximization is over choices of $y_0, y_1 \ldots, y_N$, where y_t is required to be a linear function of $\{y_{t-s-1}, t + m - 1 \geq 0;\ a_{t-s}, t \geq s \geq 0\}$.

We saw in the lecture *Classical Control with Linear Algebra* that the solution of this problem under certainty could be represented in the feedback-feedforward form

$$U\bar{y} = L^{-1}\bar{a} + K \begin{bmatrix} y_{-1} \\ \vdots \\ y_{-m} \end{bmatrix}$$

for some $(N + 1) \times m$ matrix K.

Using a version of formula (6.7), we can express $\hat{\mathbb{E}}[\bar{a} \mid a_s,\ a_{s-1}, \ldots, a_0]$ as

$$\hat{\mathbb{E}}[\bar{a} \mid a_s,\ a_{s-1}, \ldots, a_0] = \tilde{U}^{-1} \begin{bmatrix} 0 & 0 \\ 0 & I_{(s+1)} \end{bmatrix} \tilde{U}\bar{a}$$

where $I_{(s+1)}$ is the $(s+1) \times (s+1)$ identity matrix, and $V = \tilde{U}^{-1}\tilde{U}^{-1'}$, where $\tilde{U}$ is the *upper* triangular Cholesky factor of the covariance matrix V.

(We have reversed the time axis in dating the a's relative to earlier)

The time axis can be reversed in representation (6.8) by replacing L with L^T.

The optimal decision rule to use at time $0 \leq t \leq N$ is then given by the $(N - t + 1)^{\text{th}}$ row of

$$U\bar{y} = L^{-1}\tilde{U}^{-1} \begin{bmatrix} 0 & 0 \\ 0 & I_{(t+1)} \end{bmatrix} \tilde{U}\bar{a} + K \begin{bmatrix} y_{-1} \\ \vdots \\ y_{-m} \end{bmatrix}$$

6.4 Infinite Horizon Prediction and Filtering Problems

It is instructive to compare the finite-horizon formulas based on linear algebra decompositions of finite-dimensional covariance matrices with classic formulas for infinite horizon and infinite history prediction and control problems.

These classic infinite horizon formulas used the mathematics of z-transforms and lag operators.

We'll meet interesting lag operator and z-transform counterparts to our finite horizon matrix formulas.

We pose two related prediction and filtering problems.

We let Y_t be a univariate m^{th} order moving average, covariance stationary stochastic process,

$$Y_t = d(L)u_t \tag{6.9}$$

where $d(L) = \sum_{j=0}^{m} d_j L^j$, and u_t is a serially uncorrelated stationary random process satisfying

$$\begin{aligned} \mathbb{E}u_t &= 0 \\ \mathbb{E}u_t u_s &= \begin{cases} 1 & \text{if } t = s \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{6.10}$$

We impose no conditions on the zeros of $d(z)$.

A second covariance stationary process is X_t given by

$$X_t = Y_t + \varepsilon_t \tag{6.11}$$

where ε_t is a serially uncorrelated stationary random process with $\mathbb{E}\varepsilon_t = 0$ and $\mathbb{E}\varepsilon_t\varepsilon_s$ = 0 for all distinct t and s.

We also assume that $\mathbb{E}\varepsilon_t u_s = 0$ for all t and s.

The **linear least squares prediction problem** is to find the L_2 random variable $\hat{X}_{t+j}$ among linear combinations of $\{X_t,\ X_{t-1}, ...\}$ that minimizes $\mathbb{E}(\hat{X}_{t+j} - X_{t+j})^2$.

That is, the problem is to find a $\gamma_j(L) = \sum_{k=0}^{\infty} \gamma_{jk}\, L^k$ such that $\sum_{k=0}^{\infty} |\gamma_{jk}|^2 < \infty$ and $\mathbb{E}[\gamma_j\,(L)X_t - X_{t+j}]^2$ is minimized.

The **linear least squares filtering problem** is to find a $b\,(L) = \sum_{j=0}^{\infty} b_j\, L^j$ such that $\sum_{j=0}^{\infty} |b_j|^2 < \infty$ and $\mathbb{E}[b\,(L)X_t - Y_t]^2$ is minimized.

Interesting versions of these problems related to the permanent income theory were studied by [Muth, 1960].

6.4.1 Problem Formulation

These problems are solved as follows.

The covariograms of Y and X and their cross covariogram are, respectively,

$$\begin{aligned} C_X(\tau) &= \mathbb{E}X_t X_{t-\tau} \\ C_Y(\tau) &= \mathbb{E}Y_t Y_{t-\tau} \qquad \tau = 0, \pm 1, \pm 2, ... \\ C_{Y,X}(\tau) &= \mathbb{E}Y_t X_{t-\tau} \end{aligned} \tag{6.12}$$

The covariance and cross-covariance generating functions are defined as

$$\begin{aligned} g_X(z) &= \sum_{\tau=-\infty}^{\infty} C_X(\tau) z^{\tau} \\ g_Y(z) &= \sum_{\tau=-\infty}^{\infty} C_Y(\tau) z^{\tau} \\ g_{YX}(z) &= \sum_{\tau=-\infty}^{\infty} C_{YX}(\tau) z^{\tau} \end{aligned} \tag{6.13}$$

The generating functions can be computed by using the following facts.

Let v_{1t} and v_{2t} be two mutually and serially uncorrelated white noises with unit variances.

That is, $\mathbb{E}v_{1t}^2 = \mathbb{E}v_{2t}^2 = 1, \mathbb{E}v_{1t} = \mathbb{E}v_{2t} = 0, \mathbb{E}v_{1t}v_{2s} = 0$ for all t and s, $\mathbb{E}v_{1t}v_{1t-j} = \mathbb{E}v_{2t}v_{2t-j} = 0$ for all $j \neq 0$.

Let x_t and y_t be two random processes given by

$$\begin{aligned} y_t &= A(L)v_{1t} + B(L)v_{2t} \\ x_t &= C(L)v_{1t} + D(L)v_{2t} \end{aligned}$$

Then, as shown for example in [Sargent, 1987] [ch. XI], it is true that

$$\begin{aligned} g_y(z) &= A(z)A(z^{-1}) + B(z)B(z^{-1}) \\ g_x(z) &= C(z)C(z^{-1}) + D(z)D(z^{-1}) \\ g_{yx}(z) &= A(z)C(z^{-1}) + B(z)D(z^{-1}) \end{aligned} \tag{6.14}$$

Applying these formulas to (6.9) – (6.12), we have

$$\begin{aligned} g_Y(z) &= d(z)d(z^{-1}) \\ g_X(z) &= d(z)d(z^{-1}) + h \\ g_{YX}(z) &= d(z)d(z^{-1}) \end{aligned} \tag{6.15}$$

The key step in obtaining solutions to our problems is to factor the covariance generating function $g_X(z)$ of X.

The solutions of our problems are given by formulas due to Wiener and Kolmogorov.

These formulas utilize the Wold moving average representation of the X_t process,

$$X_t = c\,(L)\,\eta_t \tag{6.16}$$

where $c(L) = \sum_{j=0}^{m} c_j\, L^j$, with

$$c_0 \eta_t = X_t - \hat{\mathbb{E}}[X_t | X_{t-1}, X_{t-2}, ...] \tag{6.17}$$

Here $\hat{\mathbb{E}}$ is the linear least squares projection operator.

Equation (6.17) is the condition that $c_0 \eta_t$ can be the one-step-ahead error in predicting X_t from its own past values.

Condition (6.17) requires that η_t lie in the closed linear space spanned by $[X_t,\ X_{t-1}, ...]$.

This will be true if and only if the zeros of $c(z)$ do not lie inside the unit circle.

It is an implication of (6.17) that η_t is a serially uncorrelated random process and that normalization can be imposed so that $\mathbb{E}\eta_t^2 = 1$.

Consequently, an implication of (6.16) is that the covariance generating function of X_t can be expressed as

$$g_X(z) = c\,(z)\,c\,(z^{-1}) \tag{6.18}$$

It remains to discuss how $c(L)$ is to be computed.

Combining (6.14) and (6.18) gives

$$d(z)\, d(z^{-1}) + h = c\,(z)\, c\,(z^{-1}) \tag{6.19}$$

Therefore, we have already shown constructively how to factor the covariance generating function $g_X(z) = d(z)\, d\,(z^{-1}) + h$.

We now introduce the **annihilation operator**:

$$\left[\sum_{j=-\infty}^{\infty} f_j\, L^j\right]_+ \equiv \sum_{j=0}^{\infty} f_j\, L^j \tag{6.20}$$

In words, $[\quad]_+$ means "ignore negative powers of L".

We have defined the solution of the prediction problem as $\hat{\mathbb{E}}[X_{t+j}|X_t,\, X_{t-1}, ...] = \gamma_j\,(L)X_t$.

Assuming that the roots of $c(z) = 0$ all lie outside the unit circle, the Wiener-Kolmogorov formula for $\gamma_j(L)$ holds:

$$\gamma_j\,(L) = \left[\frac{c(L)}{L^j}\right]_+ c\,(L)^{-1} \tag{6.21}$$

We have defined the solution of the filtering problem as $\hat{\mathbb{E}}[Y_t \mid X_t, X_{t-1}, ...] = b(L)X_t$.

The Wiener-Kolomogorov formula for $b(L)$ is

$$b(L) = \left[\frac{g_{YX}(L)}{c(L^{-1})}\right]_+ c(L)^{-1}$$

or

$$b(L) = \left[\frac{d(L)d(L^{-1})}{c(L^{-1})}\right]_+ c(L)^{-1} \tag{6.22}$$

Formulas (6.21) and (6.22) are discussed in detail in [Whittle, 1983] and [Sargent, 1987].

The interested reader can there find several examples of the use of these formulas in economics Some classic examples using these formulas are due to [Muth, 1960].

As an example of the usefulness of formula (6.22), we let X_t be a stochastic process with Wold moving average representation

$$X_t = c(L)\eta_t$$

where $\mathbb{E}\eta_t^2 = 1,$ and $c_0\eta_t = X_t - \hat{\mathbb{E}}[X_t|X_{t-1}, ...], c(L) = \sum_{j=0}^m c_j L.$

Suppose that at time t, we wish to predict a geometric sum of future X's, namely

$$y_t \equiv \sum_{j=0}^{\infty} \delta^j X_{t+j} = \frac{1}{1-\delta L^{-1}} X_t$$

given knowledge of $X_t, X_{t-1}, ...$

We shall use (6.22) to obtain the answer.

Using the standard formulas (6.14), we have that

$$g_{yx}(z) = (1-\delta z^{-1})c(z)c(z^{-1})$$
$$g_x(z) = c(z)c(z^{-1})$$

Then (6.22) becomes

$$b(L) = \left[\frac{c(L)}{1 - \delta L^{-1}} \right]_+ c(L)^{-1} \tag{6.23}$$

In order to evaluate the term in the annihilation operator, we use the following result from [Hansen and Sargent, 1980].

Proposition Let

- $g(z) = \sum_{j=0}^\infty g_j \, z^j$ where $\sum_{j=0}^\infty |g_j|^2 < +\infty$.
- $h\left(z^{-1}\right) = (1 - \delta_1 z^{-1}) \dots (1 - \delta_n z^{-1})$, where $|\delta_j| < 1$, for $j = 1, \dots, n$.

Then

$$\left[\frac{g(z)}{h(z^{-1})} \right]_+ = \frac{g(z)}{h(z^{-1})} - \sum_{j=1}^n \frac{\delta_j g(\delta_j)}{\prod_{\substack{k=1 \\ k \neq j}}^n (\delta_j - \delta_k)} \left(\frac{1}{z - \delta_j} \right) \tag{6.24}$$

and, alternatively,

$$\left[\frac{g(z)}{h(z^{-1})} \right]_+ = \sum_{j=1}^n B_j \left(\frac{z g(z) - \delta_j g(\delta_j)}{z - \delta_j} \right) \tag{6.25}$$

where $B_j = 1 / \prod_{\substack{k=1 \\ k+j}}^n (1 - \delta_k / \delta_j)$.

Applying formula (6.25) of the proposition to evaluating (6.23) with $g(z) = c(z)$ and $h(z^{-1}) = 1 - \delta z^{-1}$ gives

$$b(L) = \left[\frac{Lc(L) - \delta c(\delta)}{L - \delta} \right] c(L)^{-1}$$

or

$$b(L) = \left[\frac{1 - \delta c(\delta) L^{-1} c(L)^{-1}}{1 - \delta L^{-1}} \right]$$

Thus, we have

$$\hat{\mathbb{E}} \left[\sum_{j=0}^\infty \delta^j X_{t+j} | X_t, \, x_{t-1}, \dots \right] = \left[\frac{1 - \delta c(\delta) L^{-1} c(L)^{-1}}{1 - \delta L^{-1}} \right] X_t \tag{6.26}$$

This formula is useful in solving stochastic versions of problem 1 of lecture *Classical Control with Linear Algebra* in which the randomness emerges because $\{a_t\}$ is a stochastic process.

The problem is to maximize

$$\mathbb{E}_0 \lim_{N \to \infty} \sum_{t-0}^N \beta^t \left[a_t \, y_t - \frac{1}{2} \, h y_t^2 - \frac{1}{2} \, [d(L) y_t]^2 \right] \tag{6.27}$$

where $\mathbb{E}_t$ is mathematical expectation conditioned on information known at t, and where $\{a_t\}$ is a covariance stationary stochastic process with Wold moving average representation

$$a_t = c(L) \, \eta_t$$

where

$$c(L) = \sum_{j=0}^{\tilde{n}} c_j L^j$$

and

$$\eta_t = a_t - \hat{\mathbb{E}}[a_t | a_{t-1}, ...]$$

The problem is to maximize (6.27) with respect to a contingency plan expressing y_t as a function of information known at t, which is assumed to be $(y_{t-1},\ y_{t-2}, \ldots, a_t,\ a_{t-1}, \ldots)$.

The solution of this problem can be achieved in two steps.

First, ignoring the uncertainty, we can solve the problem assuming that $\{a_t\}$ is a known sequence.

The solution is, from above,

$$c(L) y_t = c(\beta L^{-1})^{-1} a_t$$

or

$$(1 - \lambda_1 L) \ldots (1 - \lambda_m L) y_t = \sum_{j=1}^m A_j \sum_{k=0}^\infty (\lambda_j \beta)^k \, a_{t+k} \tag{6.28}$$

Second, the solution of the problem under uncertainty is obtained by replacing the terms on the right-hand side of the above expressions with their linear least squares predictors.

Using (6.26) and (6.28), we have the following solution

$$(1 - \lambda_1 L) \ldots (1 - \lambda_m L) y_t = \sum_{j=1}^m A_j \left[\frac{1 - \beta \lambda_j \, c (\beta \lambda_j) L^{-1} c(L)^{-1}}{1 - \beta \lambda_j L^{-1}} \right] a_t$$

Blaschke factors

The following is a useful piece of mathematics underlying "root flipping".

Let $\pi(z) = \sum_{j=0}^m \pi_j z^j$ and let $z_1, \ldots, z_k$ be the zeros of $\pi(z)$ that are inside the unit circle, $k < m$.

Then define

$$\theta(z) = \pi(z) \left(\frac{(z_1 z - 1)}{(z - z_1)} \right) \left(\frac{(z_2 z - 1)}{(z - z_2)} \right) \ldots \left(\frac{(z_k z - 1)}{(z - z_k)} \right)$$

The term multiplying $\pi(z)$ is termed a "Blaschke factor".

Then it can be proved directly that

$$\theta(z^{-1}) \theta(z) = \pi(z^{-1}) \pi(z)$$

and that the zeros of $\theta(z)$ are not inside the unit circle.

6.5 Exercises

Exercise 6.5.1

Let $Y_t = (1 - 2L) u_t$ where u_t is a mean zero white noise with $\mathbb{E} u^2_t = 1$. Let

$$X_t = Y_t + \varepsilon_t$$

where ε_t is a serially uncorrelated white noise with $\mathbb{E} \varepsilon^2_t = 9$, and $\mathbb{E} \varepsilon_t u_s = 0$ for all t and s.

Find the Wold moving average representation for X_t.

Find a formula for the A_{1j}'s in

$$\mathbb{E}\widehat{X}_{t+1} \mid X_t, X_{t-1}, \ldots = \sum_{j=0}^{\infty} A_{1j} X_{t-j}$$

Find a formula for the A_{2j}'s in

$$\hat{\mathbb{E}} X_{t+2} \mid X_t, X_{t-1}, \ldots = \sum_{j=0}^{\infty} A_{2j} X_{t-j}$$

Exercise 6.5.2

Multivariable Prediction: Let Y_t be an $(n \times 1)$ vector stochastic process with moving average representation

$$Y_t = D(L) U_t$$

where $D(L) = \sum_{j=0}^{m} D_j L^J$, D_j an $n \times n$ matrix, U_t an $(n \times 1)$ vector white noise with $\mathbb{E} U_t = 0$ for all t, $\mathbb{E} U_t U_s' = 0$ for all $s \neq t$, and $\mathbb{E} U_t U_t' = I$ for all t.

Let ε_t be an $n \times 1$ vector white noise with mean 0 and contemporaneous covariance matrix H, where H is a positive definite matrix.

Let $X_t = Y_t + \varepsilon_t$.

Define the covariograms as $C_X(\tau) = \mathbb{E} X_t X_{t-\tau}'$, $C_Y(\tau) = \mathbb{E} Y_t Y_{t-\tau}'$, $C_{YX}(\tau) = \mathbb{E} Y_t X_{t-\tau}'$.

Then define the matrix covariance generating function, as in (5.21), only interpret all the objects in (5.21) as matrices.

Show that the covariance generating functions are given by

$$\begin{aligned} g_y(z) &= D(z) D(z^{-1})' \\ g_X(z) &= D(z) D(z^{-1})' + H \\ g_{YX}(z) &= D(z) D(z^{-1})' \end{aligned}$$

A factorization of $g_X(z)$ can be found (see [Rozanov, 1967] or [Whittle, 1983]) of the form

$$D(z) D(z^{-1})' + H = C(z) C(z^{-1})', \quad C(z) = \sum_{j=0}^{m} C_j z^j$$

where the zeros of $|C(z)|$ do not lie inside the unit circle.

A vector Wold moving average representation of X_t is then

$$X_t = C(L) \eta_t$$

where η_t is an $(n \times 1)$ vector white noise that is "fundamental" for X_t.

That is, $X_t - \hat{\mathbb{E}} \left[X_t \mid X_{t-1}, X_{t-2} \ldots \right] = C_0 \, \eta_t$.

The optimum predictor of X_{t+j} is

$$\hat{\mathbb{E}} \left[X_{t+j} \mid X_t, X_{t-1}, \ldots \right] = \left[\frac{C(L)}{L^j} \right]_+ \eta_t$$

If $C(L)$ is invertible, i.e., if the zeros of det $C(z)$ lie strictly outside the unit circle, then this formula can be written

$$\hat{\mathbb{E}} \left[X_{t+j} \mid X_t, X_{t-1}, \ldots \right] = \left[\frac{C(L)}{L^J} \right]_+ C(L)^{-1} \, X_t$$

Part II

Linear Programming

CHAPTER

SEVEN

LINEAR PROGRAMMING

7.1 Overview

Linear programming problems either maximize or minimize a linear objective function subject to a set of linear equality and/or inequality constraints.

Linear programs come in pairs:

- an original **primal** problem, and
- an associated **dual** problem.

If a primal problem involves **maximization**, the dual problem involves **minimization**.

If a primal problem involves **minimization**, the dual problem involves **maximization**.

We provide a standard form of a linear program and methods to transform other forms of linear programming problems into a standard form.

We tell how to solve a linear programming problem using SciPy.

We describe the important concept of complementary slackness and how it relates to the dual problem.

Let's start with some standard imports.

```python
import numpy as np
from scipy.optimize import linprog
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
```

7.2 Objective Function and Constraints

We want to minimize a **cost function** $c'x = \sum_{i=1}^n c_i x_i$ over feasible values of $x = (x_1, x_2, \dots, x_n)'$.

Here

- $c = (c_1, c_2, \dots, c_n)'$ is a **unit cost vector**, and
- $x = (x_1, x_2, \dots, x_n)'$ is a vector of **decision variables**

Decision variables are restricted to satisfy a set of linear equality and/or inequality constraints.

We describe the constraints with the following collections of n-dimensional vectors a_i and scalars b_i and associated sets indexing the equality and inequality constraints:

- a_i for $i \in M_i$, where M_1, M_2, M_3 are each sets of indexes and a collection of scalers

- b_i for $i \in N_i$, where N_1, N_2, N_3 are each sets of indexes.

A linear programming can be stated as [Bertsimas, 1997]:

$$\begin{aligned}
\min_{x} \ & c'x \\
\text{subject to} \ & a_i'x \geq b_i, && i \in M_1 \\
& a_i'x \leq b_i, && i \in M_2 \\
& a_i'x = b_i, && i \in M_3 \\
& x_j \geq 0, && j \in N_1 \\
& x_j \leq 0, && j \in N_2 \\
& x_j \text{ unrestricted}, && j \in N_3
\end{aligned} \tag{7.1}$$

A vector x that satisfies all of the constraints is called a **feasible solution**.

A collection of all feasible solutions is called a **feasible set**.

A feasible solution x that minimizes the cost function is called an **optimal solution**.

The corresponding value of cost function $c'x$ is called the **optimal value**.

If the feasible set is empty, we say that solving the linear programming problem is **infeasible**.

If, for any $K \in \mathbb{R}$, there exists a feasible solution x such that $c'x < K$, we say that the problem is **unbounded** or equivalently that the optimal value is $-\infty$.

7.3 Example 1: Production Problem

This example was created by [Bertsimas, 1997]

Suppose that a factory can produce two goods called Product 1 and Product 2.

To produce each product requires both material and labor.

Selling each product generates revenue.

Required per unit material and labor inputs and revenues are shown in table below:

	Product 1	Product 2
Material	2	5
Labor	4	2
Revenue	3	4

30 units of material and 20 units of labor available.

A firm's problem is to construct a production plan that uses its 30 units of materials and 20 unites of labor to maximize its revenue.

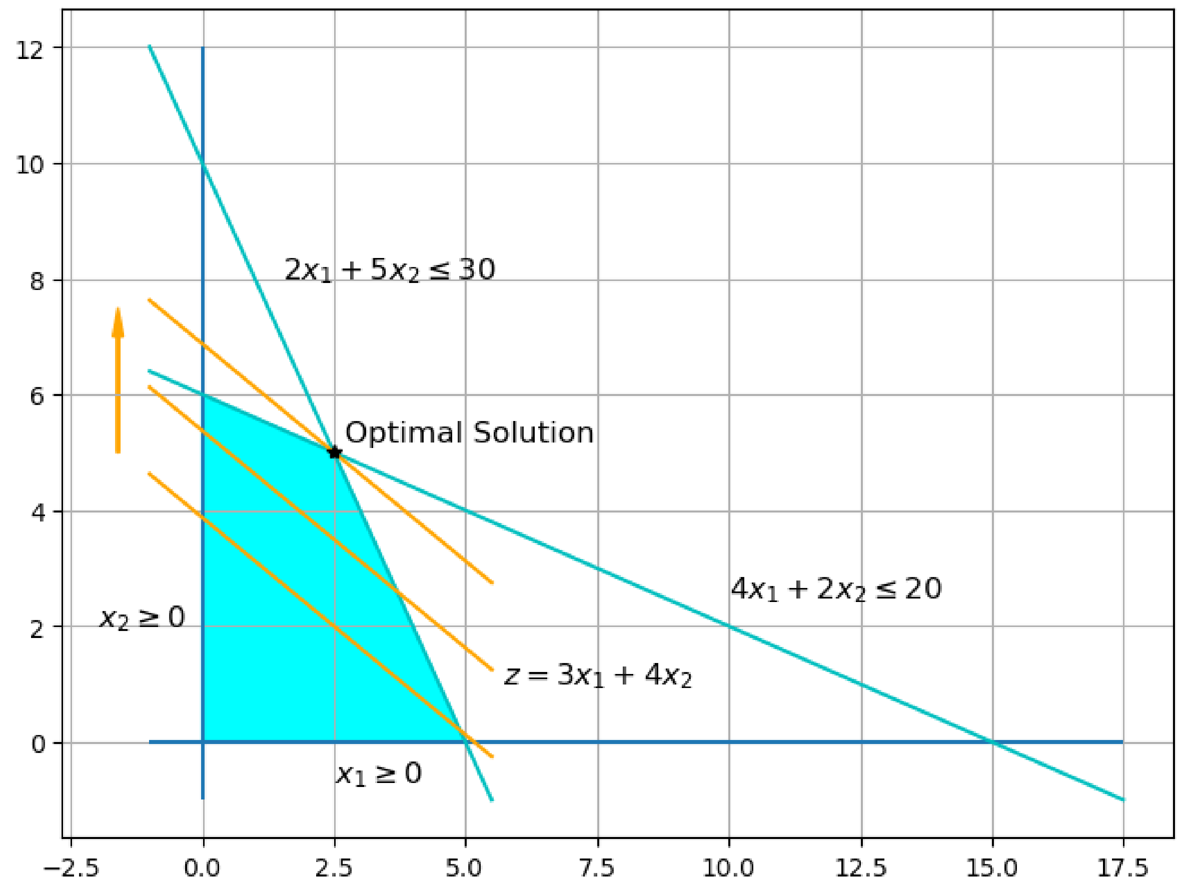
Let x_i denote the quantity of Product i that the firm produces.

This problem can be formulated as:

$$\begin{aligned}\max_{x_1,x_2}\ & z = 3x_1 + 4x_2 \\ \text{subject to}\ & 2x_1 + 5x_2 \leq 30 \\ & 4x_1 + 2x_2 \leq 20 \\ & x_1, x_2 \geq 0\end{aligned}$$

The following graph illustrates the firm's constraints and iso-revenue lines.

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.grid()

# Draw constraint lines
ax.hlines(0, -1, 17.5)
ax.vlines(0, -1, 12)
ax.plot(np.linspace(-1, 17.5, 100), 6-0.4*np.linspace(-1, 17.5, 100), color="c")
ax.plot(np.linspace(-1, 5.5, 100), 10-2*np.linspace(-1, 5.5, 100), color="c")
ax.text(1.5, 8, "$2x_1 + 5x_2 \leq 30$", size=12)
ax.text(10, 2.5, "$4x_1 + 2x_2 \leq 20$", size=12)
ax.text(-2, 2, "$x_2 \geq 0$", size=12)
ax.text(2.5, -0.7, "$x_1 \geq 0$", size=12)

# Draw the feasible region
feasible_set = Polygon(np.array([[0, 0],
                                 [0, 6],
                                 [2.5, 5],
                                 [5, 0]]),
                       color="cyan")
ax.add_patch(feasible_set)

# Draw the objective function
ax.plot(np.linspace(-1, 5.5, 100), 3.875-0.75*np.linspace(-1, 5.5, 100), color="orange
↪")
ax.plot(np.linspace(-1, 5.5, 100), 5.375-0.75*np.linspace(-1, 5.5, 100), color="orange
↪")
ax.plot(np.linspace(-1, 5.5, 100), 6.875-0.75*np.linspace(-1, 5.5, 100), color="orange
↪")
ax.arrow(-1.6, 5, 0, 2, width = 0.05, head_width=0.2, head_length=0.5, color="orange")
ax.text(5.7, 1, "$z = 3x_1 + 4x_2$", size=12)

# Draw the optimal solution
ax.plot(2.5, 5, "*", color="black")
ax.text(2.7, 5.2, "Optimal Solution", size=12)

plt.show()
```



The blue region is the feasible set within which all constraints are satisfied.

Parallel orange lines are iso-revenue lines.

The firm's objective is to find the parallel orange lines to the upper boundary of the feasible set.

The intersection of the feasible set and the highest orange line delineates the optimal set.

In this example, the optimal set is the point $(2.5, 5)$.

7.4 Example 2: Investment Problem

We now consider a problem posed and solved by [Hu, 2018].

A mutual fund has $\$100,000$ to be invested over a three year horizon.

Three investment options are available:

1. **Annuity:** the fund can pay a same amount of new capital at the beginning of each of three years and receive a payoff of 130% of **total capital** invested at the end of the third year. Once the mutual fund decides to invest in this annuity, it has to keep investing in all subsequent years in the three year horizon.
2. **Bank account:** the fund can deposit any amount into a bank at the beginning of each year and receive its capital plus 6% interest at the end of that year. In addition, the mutual fund is permitted to borrow no more than $20,000 at the beginning of each year and is asked to pay back the amount borrowed plus 6% interest at the end of the year. The mutual fund can choose whether to deposit or borrow at the beginning of each year.

3. **Corporate bond:** At the beginning of the second year, a corporate bond becomes available. The fund can buy an amount that is no more than \$50,000 of this bond at the beginning of the second year and at the end of the third year receive a payout of 130% of the amount invested in the bond.

The mutual fund's objective is to maximize total payout that it owns at the end of the third year.

We can formulate this as a linear programming problem.

Let x_1 be the amount of put in the annuity, x_2, x_3, x_4 be bank deposit balances at the beginning of the three years, and x_5 be the amount invested in the corporate bond.

When x_2, x_3, x_4 are negative, it means that the mutual fund has borrowed from bank.

The table below shows the mutual fund's decision variables together with the timing protocol described above:

	Year 1	Year 2	Year 3
Annuity	x_1	x_1	x_1
Bank account	x_2	x_3	x_4
Corporate bond	0	x_5	0

The mutual fund's decision making proceeds according to the following timing protocol:

1. At the beginning of the first year, the mutual fund decides how much to invest in the annuity and how much to deposit in the bank. This decision is subject to the constraint:

$$x_1 + x_2 = 100,000$$

2. At the beginning of the second year, the mutual fund has a bank balance of $1.06x_2$. It must keep x_1 in the annuity. It can choose to put x_5 into the corporate bond, and put x_3 in the bank. These decisions are restricted by

$$x_1 + x_5 = 1.06x_2 - x_3$$

3. At the beginning of the third year, the mutual fund has a bank account balance equal to $1.06x_3$. It must again invest x_1 in the annuity, leaving it with a bank account balance equal to x_4. This situation is summarized by the restriction:

$$x_1 = 1.06x_3 - x_4$$

The mutual fund's objective function, i.e., its wealth at the end of the third year is:

$$1.30 \cdot 3x_1 + 1.06x_4 + 1.30x_5$$

Thus, the mutual fund confronts the linear program:

$$\begin{aligned}
\max_x \; & 1.30 \cdot 3x_1 + 1.06x_4 + 1.30x_5 \\
\text{subject to} \; & x_1 + x_2 = 100,000 \\
& x_1 - 1.06x_2 + x_3 + x_5 = 0 \\
& x_1 - 1.06x_3 + x_4 = 0 \\
& x_2 \geq -20,000 \\
& x_3 \geq -20,000 \\
& x_4 \geq -20,000 \\
& x_5 \leq 50,000 \\
& x_j \geq 0, \quad j = 1,5 \\
& x_j \text{ unrestricted}, \quad j = 2,3,4
\end{aligned}$$

7.5 Standard Form

For purposes of

- unifying linear programs that are initially stated in superficially different forms, and
- having a form that is convenient to put into black-box software packages,

it is useful to devote some effort to describe a **standard form**.

Our standard form is:

$$\begin{aligned} \min_x \ & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{subject to } \ & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ & a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ & \quad \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \\ & x_1, x_2, \dots, x_n \geq 0 \end{aligned}$$

Let

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & & \vdots & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \quad c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

The standard form LP problem can be expressed concisely as:

$$\begin{aligned} \min_x \ & c'x \\ \text{subject to } \ & Ax = b \\ & x >= 0 \end{aligned} \tag{7.2}$$

Here, $Ax = b$ means that the i-th entry of Ax equals the i-th entry of b for every i.

Similarly, $x >= 0$ means that x_j is greater than 0 for every j.

7.5.1 Useful Transformations

It is useful to know how to transform a problem that initially is not stated in the standard form into one that is.

By deploying the following steps, any linear programming problem can be transformed into an equivalent standard form linear programming problem.

1. **Objective Function:** If a problem is originally a constrained **maximization** problem, we can construct a new objective function that is the additive inverse of the original objective function. The transformed problem is then a **minimization** problem.
2. **Decision Variables:** Given a variable x_j satisfying $x_j \leq 0$, we can introduce a new variable $x_j' = -x_j$ and subsitute it into original problem. Given a free variable x_i with no restriction on its sign, we can introduce two new variables x_j^+ and x_j^- satisfying $x_j^+, x_j^- \geq 0$ and replace x_j by $x_j^+ - x_j^-$.
3. **Inequality constraints:** Given an inequality constraint $\sum_{j=1}^n a_{ij}x_j \leq 0$, we can introduce a new variable s_i, called a **slack variable** that satisfies $s_i \geq 0$ and replace the original constraint by $\sum_{j=1}^n a_{ij}x_j + s_i = 0$.

Let's apply the above steps to the two examples described above.

7.5.2 Example 1: Production Problem

The original problem is:

$$\begin{aligned}\max_{x_1,x_2}\ & 3x_1 + 4x_2\\ \text{subject to}\ & 2x_1 + 5x_2 \le 30\\ & 4x_1 + 2x_2 \le 20\\ & x_1, x_2 \ge 0\end{aligned}$$

This problem is equivalent to the following problem with a standard form:

$$\begin{aligned}\min_{x_1,x_2}\ & -(3x_1 + 4x_2)\\ \text{subject to}\ & 2x_1 + 5x_2 + s_1 = 30\\ & 4x_1 + 2x_2 + s_2 = 20\\ & x_1, x_2, s_1, s_2 \ge 0\end{aligned}$$

7.5.3 Example 2: Investment Problem

The original problem is:

$$\begin{aligned}\max_{x}\ & 1.30 \cdot 3x_1 + 1.06x_4 + 1.30x_5\\ \text{subject to}\ & x_1 + x_2 = 100,000\\ & x_1 - 1.06x_2 + x_3 + x_5 = 0\\ & x_1 - 1.06x_3 + x_4 = 0\\ & x_2 \ge -20,000\\ & x_3 \ge -20,000\\ & x_4 \ge -20,000\\ & x_5 \le 50,000\\ & x_j \ge 0, \quad j = 1,5\\ & x_j \text{ unrestricted}, \quad j = 2,3,4\end{aligned}$$

This problem is equivalent to the following problem with a standard form:

$$\begin{aligned}\min_{x}\ & -(1.30 \cdot 3x_1 + 1.06x_4^+ - 1.06x_4^- + 1.30x_5)\\ \text{subject to}\ & x_1 + x_2^+ - x_2^- = 100,000\\ & x_1 - 1.06(x_2^+ - x_2^-) + x_3^+ - x_3^- + x_5 = 0\\ & x_1 - 1.06(x_3^+ - x_3^-) + x_4^+ - x_4^- = 0\\ & x_2^- - x_2^+ + s_1 = 20,000\\ & x_3^- - x_3^+ + s_2 = 20,000\\ & x_4^- - x_4^+ + s_3 = 20,000\\ & x_5 + s_4 = 50,000\\ & x_j \ge 0, \quad j = 1,5\\ & x_j^+, x_j^- \ge 0, \quad j = 2,3,4\\ & s_j \ge 0, \quad j = 1,2,3,4\end{aligned}$$

7.6 Computations

The package *scipy.optimize* provides a function ***linprog*** to solve linear programming problems with a form below:

$$
\begin{aligned}
\min_x \ & c'x \\
\text{subject to } \ & A_{ub}x \leq b_{ub} \\
& A_{eq}x = b_{eq} \\
& l \leq x \leq u
\end{aligned}
$$

Note: By default $l = 0$ and $u = \text{None}$ unless explicitly specified with the argument 'bounds'.

Let's apply this great Python tool to solve our two example problems.

7.6.1 Example 1: Production Problem

The problem is:

$$
\begin{aligned}
\max_{x_1, x_2} \ & 3x_1 + 4x_2 \\
\text{subject to } \ & 2x_1 + 5x_2 \leq 30 \\
& 4x_1 + 2x_2 \leq 20 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

```python
# Construct parameters
c_ex1 = np.array([3, 4])

# Inequality constraints
A_ex1 = np.array([[2, 5],
                  [4, 2]])
b_ex1 = np.array([30,20])

# Solve the problem
# we put a negative sign on the objective as linprog does minimization
res_ex1 = linprog(-c_ex1, A_ub=A_ex1, b_ub=b_ex1)

res_ex1
```

```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -27.5
              x: [ 2.500e+00  5.000e+00]
            nit: 2
          lower:  residual: [ 2.500e+00  5.000e+00]
                 marginals: [ 0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf]
                 marginals: [ 0.000e+00  0.000e+00]
          eqlin:  residual: []
                 marginals: []
        ineqlin:  residual: [ 0.000e+00  0.000e+00]
                 marginals: [-6.250e-01 -4.375e-01]
```

(continues on next page)

(continued from previous page)

```
mip_node_count: 0
mip_dual_bound: 0.0
       mip_gap: 0.0
```

The optimal plan tells the factory to produce 2.5 units of Product 1 and 5 units of Product 2; that generates a maximizing value of revenue of 27.5.

We are using the *linprog* function as a **black box**.

Inside it, Python first transforms the problem into standard form.

To do that, for each inequality constraint it generates one slack variable.

Here the vector of slack variables is a two-dimensional NumPy array that equals $b_{ub} - A_{ub}x$.

See the official documentation for more details.

Note: This problem is to maximize the objective, so that we need to put a minus sign in front of parameter vector c.

7.6.2 Example 2: Investment Problem

The problem is:

$$\begin{aligned}
\max_x \ & 1.30 \cdot 3x_1 + 1.06x_4 + 1.30x_5 \\
\text{subject to } \ & x_1 + x_2 = 100,000 \\
& x_1 - 1.06x_2 + x_3 + x_5 = 0 \\
& x_1 - 1.06x_3 + x_4 = 0 \\
& x_2 \geq -20,000 \\
& x_3 \geq -20,000 \\
& x_4 \geq -20,000 \\
& x_5 \leq 50,000 \\
& x_j \geq 0, \quad j = 1,5 \\
& x_j \text{ unrestricted}, \quad j = 2,3,4
\end{aligned}$$

Let's solve this problem using *linprog*.

```
# Construct parameters
rate = 1.06

# Objective function parameters
c_ex2 = np.array([1.30*3, 0, 0, 1.06, 1.30])

# Inequality constraints
A_ex2 = np.array([[1,  1,  0,  0,  0],
                  [1, -rate, 1, 0, 1],
                  [1, 0, -rate, 1, 0]])
b_ex2 = np.array([100000, 0, 0])

# Bounds on decision variables
bounds_ex2 = [(  0,    None),
              (-20000, None),
```

(continues on next page)

(continued from previous page)

```
              (-20000, None),
              (-20000, None),
              (  0,    50000)]

# Solve the problem
res_ex2 = linprog(-c_ex2, A_eq=A_ex2, b_eq=b_ex2,
                  bounds=bounds_ex2)

res_ex2
```

```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -141018.24349792697
              x: [ 2.493e+04  7.507e+04  4.649e+03 -2.000e+04  5.000e+04]
            nit: 0
          lower:  residual: [ 2.493e+04  9.507e+04  2.465e+04  0.000e+00
                              5.000e+04]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  1.650e-01
                              0.000e+00]
          upper:  residual: [       inf        inf        inf        inf
                              0.000e+00]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                             -1.470e-03]
          eqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00]
                 marginals: [-1.376e+00 -1.299e+00 -1.225e+00]
        ineqlin:  residual: []
                 marginals: []
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

Python tells us that the best investment strategy is:

1. At the beginning of the first year, the mutual fund should buy $\$24,927.75$ of the annuity. Its bank account balance should be $\$75,072.25$.
2. At the beginning of the second year, the mutual fund should buy $\$50,000$ of the corporate bond and keep invest in the annuity. Its bank account balance should be $\$4,648.83$.
3. At the beginning of the third year, the mutual fund should borrow $\$20,000$ from the bank and invest in the annuity.
4. At the end of the third year, the mutual fund will get payouts from the annuity and corporate bond and repay its loan from the bank. At the end it will own $141018.24, so that it's total net rate of return over the three periods is 41.02%.

7.7 Duality

Associated with a linear programming of form (7.1) with m constraints and n decision variables, there is an **dual** linear programming problem that takes the form (please see [Bertsimas, 1997])

$$\begin{aligned} \max_{p} \ & b'p \\ \text{subject to } \ & p_i \geq 0, && i \in M_1 \\ & p_i \leq 0, && i \in M_2 \\ & p_i \text{ unrestricted}, && i \in M_3 \\ & A_j'p \leq c_j, && j \in N_1 \\ & A_j'p \geq c_j, && j \in N_2 \\ & A_j'p = c_j, && j \in N_3 \end{aligned}$$

Where A_j is j-th column of the m by n matrix A.

Note: In what follows, we shall use a_i' to denote the i-th row of A and A_j to denote the j-th column of A.

$$A = \begin{bmatrix} a_1' \\ a_2' \\ \\ a_m' \end{bmatrix}.$$

To construct the dual of linear programming problem (7.1), we proceed as follows:

1. For every constraint $a_i'x \geq (\leq or =)b_i$, $j = 1, 2, ..., m$, in the primal problem, we construct a corresponding dual variable p_i. p_i is restricted to be positive if $a_i'x \geq b_i$ or negative if $a_i'x \leq b_i$ or unrestricted if $a_i'x = b_i$. We construct the m-dimensional vector p with entries p_i.
2. For every variable x_j, $j = 1, 2, ..., n$, we construct a corresponding dual constraint $A_j'p \geq (\leq or =)c_j$. The constraint is $A_j'p \geq c_j$ if $x_j \leq 0$, $A_j'p \leq c_j$ if $x_j \geq 0$ or $A_j'p = c_j$ if x_j is unrestricted.
3. The dual problem is to **maximize** objective function $b'p$.

For a **maximization** problem, we can first transform it to an equivalent minimization problem and then follow the above steps above to construct the dual **minimization** problem.

We can easily verify that **the dual of a dual problem is the primal problem**.

The following table summarizes relationships between objects in primal and dual problems.

Objective: Min	Objective: Max
m constraints	m variables
constraint $\geq$	variable ≥ 0
constraint $\leq$	variable ≤ 0
constraint $=$	variable free
n variables	n constraints
variable ≥ 0	constraint $\leq$
variable ≤ 0	constraint $\geq$
variable free	constraint $=$

As an example, the dual problem of the standard form (7.2) is:

$$\begin{aligned} \max_p \ & b'p \\ \text{subject to } \ & A'p \leq c \end{aligned}$$

As another example, consider a linear programming problem with form:

$$\begin{aligned} \max_x \ & c'x \\ \text{subject to } \ & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{7.3}$$

Its dual problem is:

$$\begin{aligned} \min_p \ & b'p \\ \text{subject to } \ & A'p \geq c \\ & p \geq 0 \end{aligned}$$

7.8 Duality Theorems

Primal and dual problems are linked by powerful **duality theorems** that have **weak** and **strong** forms.

The duality theorems provide the foundations of enlightening economic interpretations of linear programming problems.

Weak duality: For linear programming problem (7.1), if x and p are feasible solutions to the primal and the dual problems, respectively, then

$$b'p \leq c'x$$

Strong duality: For linear programming problem (7.1), if the primal problem has an optimal solution x, then the dual problem also has an optimal solution. Denote an optimal solution of the dual problem as p. Then

$$b'p = c'x$$

According to strong duality, we can find the optimal value for the primal problem by solving the dual problem.

But the dual problem tells us even more as we shall see next.

7.8.1 Complementary Slackness

Let x and p be feasible solutions to the primal problem (7.1) and its dual problem, respectively.

Then x and p are also optimal solutions of the primal and dual problems if and only if:

$$\begin{aligned} p_i(a_i'x - b_i) = 0, \quad \forall i, \\ x_j(A_j'p - c_j) = 0, \quad \forall j. \end{aligned}$$

This means that $p_i = 0$ if $a_i'x - b_i \neq 0$ and $x_j = 0$ if $A_j'p - c_j \neq 0$.

These are the celebrated **complementary slackness** conditions.

Let's interpret them.

7.8.2 Interpretations

Let's take a version of problem (7.3) as a production problem and consider its associated dual problem.

A factory produce n products with m types of resources.

Where $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$, let

- x_j denote quantities of product j to be produced
- a_{ij} denote required amount of resource i to make one unit of product j,
- b_i denotes the avaliable amount of resource i
- c_j denotes the revenue generated by producing one unit of product j.

Dual variables: By strong duality, we have

$$c_1 x_1 + c_2 x_2 + \dots + c_n x_n = b_1 p_1 + b_2 p_2 + \dots + b_m p_m.$$

Evidently, a one unit change of b_i results in p_i units change of revenue.

Thus, a dual variable can be interpreted as the **value** of one unit of resource i.

This is why it is often called the **shadow price** of resource i.

For feasible but not optimal primal and dual solutions x and p, by weak duality, we have

$$c_1 x_1 + c_2 x_2 + \dots + c_n x_n < b_1 p_1 + b_2 p_2 + \dots + b_m p_m.$$

Note: Here, the expression is opposite to the statement above since primal problem is a minimization problem.

When a strict inequality holds, the solution is not optimal because it doesn't fully utilize all valuable resources.

Evidently,

- if a shadow price p_i is larger than the market price for Resource i, the factory should buy more Resource i and expand its scale to generate more revenue;
- if a shadow price p_i is less than the market price for Resource i, the factory should sell its Resource i.

Complementary slackness: If there exists i such that $a_i' x - b_i < 0$ for some i, then $p_i = 0$ by complementary slackness. $a_i' x - b_i < 0$ means that to achieve its optimal production, the factory doesn't require as much Resource i as it has. It is reasonable that the shadow price of Resource i is 0: some of its resource i is redundant.

If there exists j such that $A_j' p - c_j > 0$, then $x_j = 0$ by complementary slackness. $A_j' p - c_j > 0$ means that the value of all resources used when producing one unit of product j is greater than its cost.

This means that producing another product that can more efficiently utilize these resources is a better choice than producing product j

Since producing product j is not optimal, x_j should equal 0.

7.8.3 Example 1: Production Problem

This problem is one specific instance of the problem (7.3), whose economic meaning is interpreted above.

Its dual problem is:

$$\begin{aligned}
\min_{x_1, x_2} \ & 30p_1 + 20p_2 \\
\text{subject to} \ & 2p_1 + 4p_2 \geq 3 \\
& 5p_1 + 2p_2 \geq 4 \\
& p_1, p_2 \geq 0
\end{aligned}$$

We solve this dual problem by using the function *linprog*.

Since parameters used here are defined before when solving the primal problem, we won't define them here.

```
# Solve the dual problem
res_ex1_dual = linprog(b_ex1, A_ub=-A_ex1.T, b_ub=-c_ex1)

res_ex1_dual
```

```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: 27.5
              x: [ 6.250e-01  4.375e-01]
            nit: 2
          lower:  residual: [ 6.250e-01  4.375e-01]
                 marginals: [ 0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf]
                 marginals: [ 0.000e+00  0.000e+00]
          eqlin:  residual: []
                 marginals: []
        ineqlin:  residual: [ 0.000e+00  0.000e+00]
                 marginals: [-2.500e+00 -5.000e+00]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

The optimal value for the dual problem equals 27.5.

This equals the optimal value of the primal problem, an illustration of strong duality.

Shadow prices for materials and labor are 0.625 and 0.4375, respectively.

7.8.4 Example 2: Investment Problem

The dual problem is:

$$\min_{p}\ 100,000p_1 - 20,000p_4 - 20,000p_5 - 20,000p_6 + 50,000p_7$$

$$\begin{aligned}
\text{subject to }\ & p_1 + p_2 + p_3 \geq 1.30 \cdot 3 \\
& p_1 - 1.06p_2 + p_4 = 0 \\
& p_2 - 1.06p_3 + p_5 = 0 \\
& p_3 + p_6 = 1.06 \\
& p_2 + p_7 \geq 1.30 \\
& p_i \text{ unrestricted}, \quad i = 1,2,3 \\
& p_i \leq 0, \quad i = 4,5,6 \\
& p_7 \geq 0
\end{aligned}$$

We solve this dual problem by using the function *linprog*.

```
# Objective function parameters
c_ex2_dual = np.array([100000, 0, 0, -20000, -20000, -20000, 50000])

# Equality constraints
A_eq_ex2_dual = np.array([[1, -1.06,     0,  1,  0,  0,  0],
                          [0,     1, -1.06,  0,  1,  0,  0],
                          [0,     0,     1,  0,  0,  1,  0]])
b_eq_ex2_dual = np.array([0, 0, 1.06])

# Inequality constraints
A_ub_ex2_dual = - np.array([[1, 1, 1, 0, 0, 0, 0],
                            [0, 1, 0, 0, 0, 0, 1]])
b_ub_ex2_dual = - np.array([1.30*3, 1.30])

# Bounds on decision variables
bounds_ex2_dual = [(None, None),
                   (None, None),
                   (None, None),
                   (None,    0),
                   (None,    0),
                   (None,    0),
                   (   0, None)]

# Solve the dual problem
res_ex2_dual = linprog(c_ex2_dual, A_eq=A_eq_ex2_dual, b_eq=b_eq_ex2_dual,
                       A_ub=A_ub_ex2_dual, b_ub=b_ub_ex2_dual, bounds=bounds_ex2_dual)

res_ex2_dual
```

```
         message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
         success: True
          status: 0
             fun: 141018.2434979269
               x: [ 1.376e+00  1.299e+00  1.225e+00  0.000e+00  0.000e+00
                   -1.650e-01  1.470e-03]
             nit: 0
           lower:  residual: [       inf        inf        inf        inf
                                     inf        inf  1.470e-03]
```

(continues on next page)

(continued from previous page)

```
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                              0.000e+00  0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf        inf  0.000e+00
                              0.000e+00  1.650e-01        inf]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00 -9.507e+04
                             -2.465e+04  0.000e+00  0.000e+00]
          eqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00]
                 marginals: [ 7.507e+04  4.649e+03 -2.000e+04]
        ineqlin:  residual: [ 0.000e+00  0.000e+00]
                 marginals: [-2.493e+04 -5.000e+04]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

The optimal value for the dual problem is 141018.24, which equals the value of the primal problem.

Now, let's interpret the dual variables.

By strong duality and also our numerical results, we have that optimal value is:

$$100,000p_1 - 20,000p_4 - 20,000p_5 - 20,000p_6 + 50,000p_7.$$

We know if b_i changes one dollor, then the optimal payoff in the end of the third year will change p_i dollars.

For $i = 1$, this means if the initial capital changes by one dollar, then the optimal payoff in the end of the third year will change p_1 dollars.

Thus, p_1 is the potential value of one more unit of initial capital, or the shadow price for initial capital.

We can also interpret p_1 as the prospective value in the end of the third year coming from having one more dollar to invest at the beginning of the first year.

If the mutual fund can raise money at a cost lower than $p_1 - 1$, then it should raise more money to increase its revenue.

But if it bears a cost of funds higher than $p_1 - 1$, the mutual fund shouldn't do that.

For $i = 4, 5, 6$, this means that if the amount of capital that the fund is permitted to borrow from the bank changes by one dollar, the optimal pay out at the end of the third year will change p_i dollars.

Thus, for $i = 4, 5, 6$, $|p_i|$ indicates the value of one dollar that the mutual fund can borrow from the bank at the beginning of the $i - 3$-th year.

$|p_i|$ is the shadow price for the loan amount. (We use absolute value here since $p_i \leq 0$.)

If the interest rate is lower than $|p_i|$, then the mutual fund should borrow to increase its optimal payoff; if the interest rate is higher, it is better to not do this.

For $i = 7$, this means that if the amount of the corporate bond the mutual fund can buy changes one dollar, then the optimal payoff will change p_7 dollars at the end of the third year. Again, p_7 is the shadow price for the amount of the corporate bond the mutual fund can buy.

As for numerical results

1. $p_1 = 1.38$, which means one dollar of initial capital is worth $\$1.38$ at the end of the third year.
2. $p_4 = p_5 = 0$, which means the loan amounts at the beginning of the first and second year are worth nothing. Recall that the optimal solution to the primal problem, $x_2, x_3 > 0$, which means at the beginning of the first and second year, the mutual fund has a postive bank account and borrows no capital from the bank. Thus, it is reasonable that the loan amounts at the beginning of the first and second year are valueless. This is what the complementary slackness conditions mean in this setting.

3. $p_6 = -0.16$, which means one dollar of the loan amount at the beginning of the third year is worth $\$0.16$. Since $|p_6|$ is higher than the interest rate 6%, the mutual fund should borrow as much as possible at the beginning of the third year. Recall that the optimal solution to the primal problem is $x_4 = -20,000$ which means the mutual fund borrows money from the bank as much as it can.
4. $p_7 = 0.0015$, which means one dollar of the amount of the corporate bond that the mutual fund can buy is worth $\$0.0015$.

CHAPTER EIGHT

OPTIMAL TRANSPORT

8.1 Overview

The **transportation** or **optimal transport** problem is interesting both because of its many applications and because of its important role in the history of economic theory.

In this lecture, we describe the problem, tell how *linear programming* is a key tool for solving it, and then provide some examples.

We will provide other applications in followup lectures.

The optimal transport problem was studied in early work about linear programming, as summarized for example by [Dorfman *et al.*, 1958]. A modern reference about applications in economics is [Galichon, 2016].

Below, we show how to solve the optimal transport problem using several implementations of linear programming, including, in order,

1. the linprog solver from SciPy,
2. the linprog_simplex solver from QuantEcon and
3. the simplex-based solvers included in the Python Optimal Transport package.

```
!pip install --upgrade quantecon
!pip install --upgrade POT
```

Let's start with some imports.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog
from quantecon.optimize.linprog_simplex import linprog_simplex
import ot
from scipy.stats import betabinom
import networkx as nx
```

8.2 The Optimal Transport Problem

Suppose that m factories produce goods that must be sent to n locations.

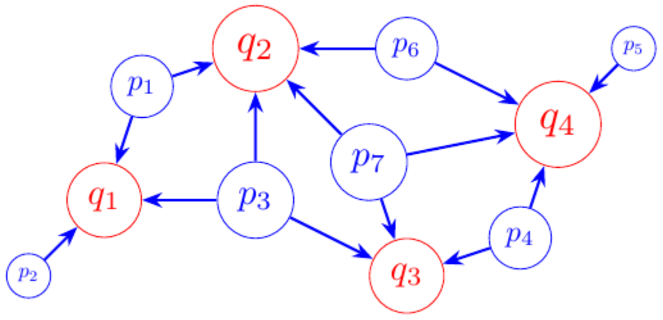
Let

- x_{ij} denote the quantity shipped from factory i to location j
- c_{ij} denote the cost of shipping one unit from factory i to location j
- p_i denote the capacity of factory i and q_j denote the amount required at location j.
- $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$.

A planner wants to minimize total transportation costs subject to the following constraints:

- The amount shipped **from** each factory must equal its capacity.
- The amount shipped **to** each location must equal the quantity required there.

The figure below shows one visualization of this idea, when factories and target locations are distributed in the plane.



The size of the vertices in the figure are proportional to

- capacity, for the factories, and
- demand (amount required) for the target locations.

The arrows show one possible transport plan, which respects the constraints stated above.

The planner's problem can be expressed as the following constrained minimization problem:

$$\begin{aligned} \min_{x_{ij}} \ & \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \text{subject to } \ & \sum_{j=1}^{n} x_{ij} = p_i, \quad i = 1, 2, \dots, m \\ & \sum_{i=1}^{m} x_{ij} = q_j, \quad j = 1, 2, \dots, n \\ & x_{ij} \geq 0 \end{aligned} \tag{8.1}$$

This is an **optimal transport problem** with

- mn decision variables, namely, the entries x_{ij} and
- $m + n$ constraints.

Summing the q_j's across all j's and the p_i's across all i's indicates that the total capacity of all the factories equals total requirements at all locations:

$$\sum_{j=1}^{n} q_j = \sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} = \sum_{i=1}^{m} p_i \tag{8.2}$$

The presence of the restrictions in (8.2) will be the source of one redundancy in the complete set of restrictions that we describe below.

More about this later.

8.3 The Linear Programming Approach

In this section we discuss using using standard linear programming solvers to tackle the optimal transport problem.

8.3.1 Vectorizing a Matrix of Decision Variables

A *matrix* of decision variables x_{ij} appears in problem (8.1).

The SciPy function `linprog` expects to see a *vector* of decision variables.

This situation impels us to rewrite our problem in terms of a *vector* of decision variables.

Let

- X, C be $m \times n$ matrices with entries x_{ij}, c_{ij},
- p be m-dimensional vector with entries p_i,
- q be n-dimensional vector with entries q_j.

With $\mathbf{1}_n$ denoting the n-dimensional column vector $(1, 1, \dots, 1)'$, our problem can now be expressed compactly as:

$$\begin{aligned} \min_{X} \ & \operatorname{tr}(C'X) \\ \text{subject to } \ & X \ \mathbf{1}_n = p \\ & X' \ \mathbf{1}_m = q \\ & X \geq 0 \end{aligned}$$

We can convert the matrix X into a vector by stacking all of its columns into a column vector.

Doing this is called **vectorization**, an operation that we denote $\operatorname{vec}(X)$.

Similarly, we convert the matrix C into an mn-dimensional vector $\operatorname{vec}(C)$.

The objective function can be expressed as the inner product between $\operatorname{vec}(C)$ and $\operatorname{vec}(X)$:

$$\operatorname{vec}(C)' \cdot \operatorname{vec}(X).$$

To express the constraints in terms of $\operatorname{vec}(X)$, we use a **Kronecker product** denoted by $\otimes$ and defined as follows.

Suppose A is an $m \times s$ matrix with entries (a_{ij}) and that B is an $n \times t$ matrix.

The **Kronecker product** of A and B is defined, in block matrix form, by

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1s}B \\ a_{21}B & a_{22}B & \dots & a_{2s}B \\ & & \vdots & \\ a_{m1}B & a_{m2}B & \dots & a_{ms}B \end{pmatrix}.$$

$A \otimes B$ is an $mn \times st$ matrix.

It has the property that for any $m \times n$ matrix X

$$\text{vec}(A'XB) = (B' \otimes A')\,\text{vec}(X). \tag{8.3}$$

We can now express our constraints in terms of $\text{vec}(X)$.

Let $A = \mathbf{I}'_m, B = \mathbf{1}_n$.

By equation (8.3)

$$X\,\mathbf{1}_n = \text{vec}(X\,\mathbf{1}_n) = \text{vec}(\mathbf{I}_m X\,\mathbf{1}_n) = (\mathbf{1}'_n \otimes \mathbf{I}_m)\,\text{vec}(X).$$

where $\mathbf{I}_m$ denotes the $m \times m$ identity matrix.

Constraint $X\,\mathbf{1}_n = p$ can now be written as:

$$(\mathbf{1}'_n \otimes \mathbf{I}_m)\,\text{vec}(X) = p.$$

Similarly, the constraint $X'\,\mathbf{1}_m = q$ can be rewriten as:

$$(\mathbf{I}_n \otimes \mathbf{1}'_m)\,\text{vec}(X) = q.$$

With $z := \text{vec}(X)$, our problem can now be expressed in terms of an mn-dimensional vector of decision variables:

$$\begin{aligned} \min_z \ & \text{vec}(C)'z \\ \text{subject to } \ & Az = b \\ & z \geq 0 \end{aligned} \tag{8.4}$$

where

$$A = \begin{pmatrix} \mathbf{1}'_n \otimes \mathbf{I}_m \\ \mathbf{I}_n \otimes \mathbf{1}'_m \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} p \\ q \end{pmatrix}$$

8.3.2 An Application

We now provide an example that takes the form (8.4) that we'll solve by deploying the function `linprog`.

The table below provides numbers for the requirements vector q, the capacity vector p, and entries c_{ij} of the cost-of-shipping matrix C.

The numbers in the above table tell us to set $m = 3$, $n = 5$, and construct the following objects:

$$p = \begin{pmatrix} 50 \\ 100 \\ 150 \end{pmatrix}, \quad q = \begin{pmatrix} 25 \\ 115 \\ 60 \\ 30 \\ 70 \end{pmatrix} \quad \text{and} \quad C = \begin{pmatrix} 10 & 15 & 20 & 20 & 40 \\ 20 & 40 & 15 & 30 & 30 \\ 30 & 35 & 40 & 55 & 25 \end{pmatrix}.$$

Let's write Python code that sets up the problem and solves it.

```
# Define parameters
m = 3
n = 5

p = np.array([50, 100, 150])
```

(continues on next page)

(continued from previous page)

```
q = np.array([25, 115, 60, 30, 70])

C = np.array([[10, 15, 20, 20, 40],
              [20, 40, 15, 30, 30],
              [30, 35, 40, 55, 25]])

# Vectorize matrix C
C_vec = C.reshape((m*n, 1), order='F')

# Construct matrix A by Kronecker product
A1 = np.kron(np.ones((1, n)), np.identity(m))
A2 = np.kron(np.identity(n), np.ones((1, m)))
A = np.vstack([A1, A2])

# Construct vector b
b = np.hstack([p, q])

# Solve the primal problem
res = linprog(C_vec, A_eq=A, b_eq=b)

# Print results
print("message:", res.message)
print("nit:", res.nit)
print("fun:", res.fun)
print("z:", res.x)
print("X:", res.x.reshape((m,n), order='F'))
```

```
message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
nit: 8
fun: 7225.0
z: [ 0. 10. 15. 50.  0. 65.  0. 60.  0.  0. 30.  0.  0.  0. 70.]
X: [[ 0. 50.  0.  0.  0.]
 [10.  0. 60. 30.  0.]
 [15. 65.  0.  0. 70.]]
```

Notice how, in the line `C_vec = C.reshape((m*n, 1), order='F')`, we are careful to vectorize using the flag `order='F'`.

This is consistent with converting C into a vector by stacking all of its columns into a column vector.

Here `'F'` stands for "Fortran", and we are using Fortran style column-major order.

(For an alternative approach, using Python's default row-major ordering, see this lecture by Alfred Galichon.)

Interpreting the warning:

The above warning message from SciPy points out that A is not full rank.

This indicates that the linear program has been set up to include one or more redundant constraints.

Here, the source of the redundancy is the structure of restrictions (8.2).

Let's explore this further by printing out A and staring at it.

```
A
```

```
array([[1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0.],
       [0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1.],
       [1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1.]])
```

The singularity of A reflects that the first three constraints and the last five constraints both require that "total requirements equal total capacities" expressed in (8.2).

One equality constraint here is redundant.

Below we drop one of the equality constraints, and use only 7 of them.

After doing this, we attain the same minimized cost.

However, we find a different transportation plan.

Though it is a different plan, it attains the same cost!

```
linprog(C_vec, A_eq=A[:-1], b_eq=b[:-1])
```

```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: 7225.0
              x: [ 0.000e+00  1.000e+01 ...  0.000e+00  7.000e+01]
            nit: 8
          lower:  residual: [ 0.000e+00  1.000e+01 ...  0.000e+00
                              7.000e+01]
                 marginals: [ 0.000e+00  0.000e+00 ...  1.500e+01
                              0.000e+00]
          upper:  residual: [       inf        inf ...        inf
                                    inf]
                 marginals: [ 0.000e+00  0.000e+00 ...  0.000e+00
                              0.000e+00]
          eqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                              0.000e+00  0.000e+00  0.000e+00]
                 marginals: [ 5.000e+00  1.500e+01  2.500e+01  5.000e+00
                              1.000e+01 -0.000e+00  1.500e+01]
        ineqlin:  residual: []
                 marginals: []
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

```
%time linprog(C_vec, A_eq=A[:-1], b_eq=b[:-1])
```

```
CPU times: user 811 µs, sys: 136 µs, total: 947 µs
Wall time: 875 µs
```

```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
```

(continues on next page)

(continued from previous page)

```
         status: 0
            fun: 7225.0
              x: [ 0.000e+00  1.000e+01 ...  0.000e+00  7.000e+01]
            nit: 8
          lower:  residual: [ 0.000e+00  1.000e+01 ...  0.000e+00
                              7.000e+01]
                 marginals: [ 0.000e+00  0.000e+00 ...  1.500e+01
                              0.000e+00]
          upper:  residual: [       inf        inf ...        inf
                                    inf]
                 marginals: [ 0.000e+00  0.000e+00 ...  0.000e+00
                              0.000e+00]
          eqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                              0.000e+00  0.000e+00  0.000e+00]
                 marginals: [ 5.000e+00  1.500e+01  2.500e+01  5.000e+00
                              1.000e+01 -0.000e+00  1.500e+01]
        ineqlin:  residual: []
                 marginals: []
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

```
%time linprog(C_vec, A_eq=A, b_eq=b)
```

```
CPU times: user 1.29 ms, sys: 0 ns, total: 1.29 ms
Wall time: 1.19 ms
```

```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: 7225.0
              x: [ 0.000e+00  1.000e+01 ...  0.000e+00  7.000e+01]
            nit: 8
          lower:  residual: [ 0.000e+00  1.000e+01 ...  0.000e+00
                              7.000e+01]
                 marginals: [ 0.000e+00  0.000e+00 ...  1.500e+01
                              0.000e+00]
          upper:  residual: [       inf        inf ...        inf
                                    inf]
                 marginals: [ 0.000e+00  0.000e+00 ...  0.000e+00
                              0.000e+00]
          eqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                              0.000e+00  0.000e+00  0.000e+00  0.000e+00]
                 marginals: [ 1.000e+01  2.000e+01  3.000e+01 -0.000e+00
                              5.000e+00 -5.000e+00  1.000e+01 -5.000e+00]
        ineqlin:  residual: []
                 marginals: []
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

Evidently, it is slightly quicker to work with the system that removed a redundant constraint.

Let's drill down and do some more calculations to help us understand whether or not our finding **two** different optimal transport plans reflects our having dropped a redundant equality constraint.

Hint

It will turn out that dropping a redundant equality isn't really what mattered.

To verify our hint, we shall simply use **all** of the original equality constraints (including a redundant one), but we'll just shuffle the order of the constraints.

```
arr = np.arange(m+n)
```

```
sol_found = []
cost = []

# simulate 1000 times
for i in range(1000):

    np.random.shuffle(arr)
    res_shuffle = linprog(C_vec, A_eq=A[arr], b_eq=b[arr])

    # if find a new solution
    sol = tuple(res_shuffle.x)
    if sol not in sol_found:
        sol_found.append(sol)
        cost.append(res_shuffle.fun)
```

```
for i in range(len(sol_found)):
    print(f"transportation plan {i}: ", sol_found[i])
    print(f"     minimized cost {i}: ", cost[i])
```

```
transportation plan 0:  (0.0, 10.0, 15.0, 50.0, 0.0, 65.0, 0.0, 60.0, 0.0, 0.0, 30.
↪0, 0.0, 0.0, 0.0, 70.0)
     minimized cost 0:  7225.0
```

Ah hah! As you can see, putting constraints in different orders in this case uncovers two optimal transportation plans that achieve the same minimized cost.

These are the same two plans computed earlier.

Next, we show that leaving out the first constraint "accidentally" leads to the initial plan that we computed.

```
linprog(C_vec, A_eq=A[1:], b_eq=b[1:])
```

```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: 7225.0
              x: [ 0.000e+00  1.000e+01 ...  0.000e+00  7.000e+01]
            nit: 8
          lower:  residual: [ 0.000e+00  1.000e+01 ...  0.000e+00
                              7.000e+01]
                 marginals: [ 0.000e+00  0.000e+00 ...  1.500e+01
                              0.000e+00]
          upper:  residual: [       inf        inf ...        inf
                                    inf]
```

(continues on next page)

(continued from previous page)

```
                  marginals: [ 0.000e+00  0.000e+00 ...  0.000e+00
                               0.000e+00]
           eqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                               0.000e+00  0.000e+00  0.000e+00]
                  marginals: [ 1.000e+01  2.000e+01  1.000e+01  1.500e+01
                               5.000e+00  2.000e+01  5.000e+00]
         ineqlin:  residual: []
                  marginals: []
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

Let's compare this transport plan with

```
res.x
```

```
array([ 0., 10., 15., 50.,  0., 65.,  0., 60.,  0.,  0., 30.,  0.,  0.,
        0., 70.])
```

Here the matrix X contains entries x_{ij} that tell amounts shipped **from** factor $i = 1, 2, 3$ **to** location $j = 1, 2, \ldots, 5$.

The vector z evidently equals $\text{vec}(X)$.

The minimized cost from the optimal transport plan is given by the fun variable.

8.3.3 Using a Just-in-Time Compiler

We can also solve optimal transportation problems using a powerful tool from QuantEcon, namely, `quantecon.optimize.linprog_simplex`.

While this routine uses the same simplex algorithm as `scipy.optimize.linprog`, the code is accelerated by using a just-in-time compiler shipped in the `numba` library.

As you will see very soon, by using `scipy.optimize.linprog` the time required to solve an optimal transportation problem can be reduced significantly.

```
# construct matrices/vectors for linprog_simplex
c = C.flatten()

# Equality constraints
A_eq = np.zeros((m+n, m*n))
for i in range(m):
    for j in range(n):
        A_eq[i, i*n+j] = 1
        A_eq[m+j, i*n+j] = 1

b_eq = np.hstack([p, q])
```

Since `quantecon.optimize.linprog_simplex` does maximization instead of minimization, we need to put a negative sign before vector `c`.

```
res_qe = linprog_simplex(-c, A_eq=A_eq, b_eq=b_eq)
```

Since the two LP solvers use the same simplex algorithm, we expect to get exactly the same solutions

```
res_qe.x.reshape((m, n), order='C')
```

```
array([[15., 35.,  0.,  0.,  0.],
       [10.,  0., 60., 30.,  0.],
       [ 0., 80.,  0.,  0., 70.]])
```

```
res.x.reshape((m, n), order='F')
```

```
array([[ 0., 50.,  0.,  0.,  0.],
       [10.,  0., 60., 30.,  0.],
       [15., 65.,  0.,  0., 70.]])
```

Let's do a speed comparison between `scipy.optimize.linprog` and `quantecon.optimize.linprog_simplex`.

```
# scipy.optimize.linprog
%time res = linprog(C_vec, A_eq=A[:-1, :], b_eq=b[:-1])
```

```
CPU times: user 1.23 ms, sys: 75 µs, total: 1.3 ms
Wall time: 1.13 ms
```

```
# quantecon.optimize.linprog_simplex
%time out = linprog_simplex(-c, A_eq=A_eq, b_eq=b_eq)
```

```
CPU times: user 59 µs, sys: 4 µs, total: 63 µs
Wall time: 66.8 µs
```

As you can see, the `quantecon.optimize.linprog_simplex` is much faster.

(Note however, that the SciPy version is probably more stable than the QuantEcon version, having been tested more extensively over a longer period of time.)

8.4 The Dual Problem

Let u, v denotes vectors of dual decision variables with entries $(u_i), (v_j)$.

The **dual** to **minimization** problem (8.1) is the **maximization** problem:

$$\begin{aligned}\max_{u_i, v_j} & \sum_{i=1}^{m} p_i u_i + \sum_{j=1}^{n} q_j v_j \\ \text{subject to } & u_i + v_j \leq c_{ij},\ i = 1, 2, \dots, m;\ j = 1, 2, \dots, n\end{aligned} \tag{8.5}$$

The dual problem is also a linear programming problem.

It has $m + n$ dual variables and mn constraints.

Vectors u and v of **values** are attached to the first and the second sets of primal constraits, respectively.

Thus, u is attached to the constraints

- $(\mathbf{1}_n' \otimes \mathbf{I}_m) \operatorname{vec}(X) = p$

and v is attached to constraints

- $(\mathbf{I}_n \otimes \mathbf{1}'_m)\,\text{vec}(X) = q.$

Components of the vectors u and v of per unit **values** are **shadow prices** of the quantities appearing on the right sides of those constraints.

We can write the dual problem as

$$\begin{aligned} \max_{u_i, v_j} \ & pu + qv \\ \text{subject to } \ & A' \begin{pmatrix} u \\ v \end{pmatrix} = \text{vec}(C) \end{aligned} \tag{8.6}$$

For the same numerical example described above, let's solve the dual problem.

```
# Solve the dual problem
res_dual = linprog(-b, A_ub=A.T, b_ub=C_vec,
                   bounds=[(None, None)]*(m+n))

#Print results
print("message:", res_dual.message)
print("nit:", res_dual.nit)
print("fun:", res_dual.fun)
print("u:", res_dual.x[:m])
print("v:", res_dual.x[-n:])
```

```
message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
nit: 9
fun: -7225.0
u: [-20. -10.   0.]
v: [30. 35. 25. 40. 25.]
```

We can also solve the dual problem using quantecon.optimize.linprog_simplex.

```
res_dual_qe = linprog_simplex(b_eq, A_ub=A_eq.T, b_ub=c)
```

And the shadow prices computed by the two programs are identical.

```
res_dual_qe.x
```

```
array([ 5., 15., 25.,  5., 10.,  0., 15.,  0.])
```

```
res_dual.x
```

```
array([-20., -10.,   0.,  30.,  35.,  25.,  40.,  25.])
```

We can compare computational times from using our two tools.

```
%time linprog(-b, A_ub=A.T, b_ub=C_vec, bounds=[(None, None)]*(m+n))
```

```
CPU times: user 1.32 ms, sys: 0 ns, total: 1.32 ms
Wall time: 1.22 ms
```

```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -7225.0
              x: [-2.000e+01 -1.000e+01  0.000e+00  3.000e+01  3.500e+01
                   2.500e+01  4.000e+01  2.500e+01]
            nit: 9
          lower:  residual: [       inf        inf        inf        inf
                                    inf        inf        inf        inf]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                              0.000e+00  0.000e+00  0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf        inf        inf
                                    inf        inf        inf        inf]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                              0.000e+00  0.000e+00  0.000e+00  0.000e+00]
          eqlin:  residual: []
                 marginals: []
        ineqlin:  residual: [ 0.000e+00  0.000e+00 ...  1.500e+01
                              0.000e+00]
                 marginals: [-0.000e+00 -1.000e+01 ... -0.000e+00
                             -7.000e+01]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

```
%time linprog_simplex(b_eq, A_ub=A_eq.T, b_ub=c)
```

```
CPU times: user 269 µs, sys: 0 ns, total: 269 µs
Wall time: 272 µs
```

```
SimplexResult(x=array([ 5., 15., 25.,  5., 10.,  0., 15.,  0.]), lambd=array([ 0.,␣
↪35.,  0., 15.,  0., 25.,  0., 60., 15.,  0.,  0., 80.,  0.,
        0., 70.]), fun=7225.0, success=True, status=0, num_iter=24)
```

`quantecon.optimize.linprog_simplex` solves the dual problem 10 times faster.

Just for completeness, let's solve the dual problems with nonsingular A matrices that we create by dropping a redundant equality constraint.

Try first leaving out the first constraint:

```
linprog(-b[1:], A_ub=A[1:].T, b_ub=C_vec,
        bounds=[(None, None)]*(m+n-1))
```

```
        message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -7225.0
              x: [ 1.000e+01  2.000e+01  1.000e+01  1.500e+01  5.000e+00
                   2.000e+01  5.000e+00]
            nit: 12
          lower:  residual: [       inf        inf        inf        inf
                                    inf        inf        inf]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
```

(continues on next page)

(continued from previous page)

```
                              0.000e+00  0.000e+00  0.000e+00]
              upper:  residual: [       inf        inf        inf        inf
                                         inf        inf        inf]
                     marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                                  0.000e+00  0.000e+00  0.000e+00]
              eqlin:  residual: []
                     marginals: []
            ineqlin:  residual: [ 0.000e+00  0.000e+00 ...  1.500e+01
                                  0.000e+00]
                     marginals: [-1.500e+01 -1.000e+01 ... -0.000e+00
                                 -7.000e+01]
     mip_node_count: 0
     mip_dual_bound: 0.0
            mip_gap: 0.0
```

Not let's instead leave out the last constraint:

```
linprog(-b[:-1], A_ub=A[:-1].T, b_ub=C_vec,
        bounds=[(None, None)]*(m+n-1))
```

```
            message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
            success: True
             status: 0
                fun: -7225.0
                  x: [ 5.000e+00  1.500e+01  2.500e+01  5.000e+00  1.000e+01
                      -0.000e+00  1.500e+01]
                nit: 9
              lower:  residual: [       inf        inf        inf        inf
                                         inf        inf        inf]
                     marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                                  0.000e+00  0.000e+00  0.000e+00]
              upper:  residual: [       inf        inf        inf        inf
                                         inf        inf        inf]
                     marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                                  0.000e+00  0.000e+00  0.000e+00]
              eqlin:  residual: []
                     marginals: []
            ineqlin:  residual: [ 0.000e+00  0.000e+00 ...  1.500e+01
                                  0.000e+00]
                     marginals: [-0.000e+00 -1.000e+01 ... -0.000e+00
                                 -7.000e+01]
     mip_node_count: 0
     mip_dual_bound: 0.0
            mip_gap: 0.0
```

8.4.1 Interpretation of dual problem

By **strong duality** (please see this lecture *Linear Programming*), we know that:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} = \sum_{i=1}^{m} p_i u_i + \sum_{j=1}^{n} q_j v_j$$

One unit more capacity in factory i, i.e. p_i, results in u_i more transportation costs.

Thus, u_i describes the cost of shipping one unit **from** factory i.

Call this the ship-out cost of one unit shipped from factory i.

Similarly, v_j is the cost of shipping one unit **to** location j.

Call this the ship-in cost of one unit to location j.

Strong duality implies that total transprotation costs equals total ship-out costs **plus** total ship-in costs.

It is reasonable that, for one unit of a product, ship-out cost u_i **plus** ship-in cost v_j should equal transportation cost c_{ij}.

This equality is assured by **complementary slackness** conditions that state that whenever $x_{ij} > 0$, meaning that there are positive shipments from factory i to location j, it must be true that $u_i + v_j = c_{ij}$.

8.5 The Python Optimal Transport Package

There is an excellent Python package for optimal transport that simplifies some of the steps we took above.

In particular, the package takes care of the vectorization steps before passing the data out to a linear programming routine.

(That said, the discussion provided above on vectorization remains important, since we want to understand what happens under the hood.)

8.5.1 Replicating Previous Results

The following line of code solves the example application discussed above using linear programming.

```
X = ot.emd(p, q, C)
X
```

```
/tmp/ipykernel_2392/1617639716.py:1: UserWarning: Input histogram consists of␣
↪integer. The transport plan will be casted accordingly, possibly resulting in a␣
↪loss of precision. If this behaviour is unwanted, please make sure your input␣
↪histogram consists of floating point elements.
  X = ot.emd(p, q, C)
```

```
array([[15, 35,  0,  0,  0],
       [10,  0, 60, 30,  0],
       [ 0, 80,  0,  0, 70]])
```

Sure enough, we have the same solution and the same cost

```
total_cost = np.sum(X * C)
total_cost
```

```
7225
```

8.5.2 A Larger Application

Now let's try using the same package on a slightly larger application.

The application has the same interpretation as above but we will also give each node (i.e., vertex) a location in the plane.

This will allow us to plot the resulting transport plan as edges in a graph.

The following class defines a node by

- its location $(x, y) \in \mathbb{R}^2$,
- its group (factory or location, denoted by `p` or `q`) and
- its mass (e.g., p_i or q_j).

```python
class Node:

    def __init__(self, x, y, mass, group, name):

        self.x, self.y = x, y
        self.mass, self.group = mass, group
        self.name = name
```

Next we write a function that repeatedly calls the class above to build instances.

It allocates to the nodes it creates their location, mass, and group.

Locations are assigned randomly.

```python
def build_nodes_of_one_type(group='p', n=100, seed=123):

    nodes = []
    np.random.seed(seed)

    for i in range(n):

        if group == 'p':
            m = 1/n
            x = np.random.uniform(-2, 2)
            y = np.random.uniform(-2, 2)
        else:
            m = betabinom.pmf(i, n-1, 2, 2)
            x = 0.6 * np.random.uniform(-1.5, 1.5)
            y = 0.6 * np.random.uniform(-1.5, 1.5)

        name = group + str(i)
        nodes.append(Node(x, y, m, group, name))

    return nodes
```

Now we build two lists of nodes, each one containing one type (factories or locations)

```python
n_p = 32
n_q = 32
```

(continues on next page)

(continued from previous page)

```
p_list = build_nodes_of_one_type(group='p', n=n_p)
q_list = build_nodes_of_one_type(group='q', n=n_q)

p_probs = [p.mass for p in p_list]
q_probs = [q.mass for q in q_list]
```

For the cost matrix C, we use the Euclidean distance between each factory and location.

```
c = np.empty((n_p, n_q))
for i in range(n_p):
    for j in range(n_q):
        x0, y0 = p_list[i].x, p_list[i].y
        x1, y1 = q_list[j].x, q_list[j].y
        c[i, j] = np.sqrt((x0-x1)**2 + (y0-y1)**2)
```

Now we are ready to apply the solver

```
%time pi = ot.emd(p_probs, q_probs, c)
```

```
CPU times: user 454 µs, sys: 20 µs, total: 474 µs
Wall time: 307 µs
```

Finally, let's plot the results using `networkx`.

In the plot below,

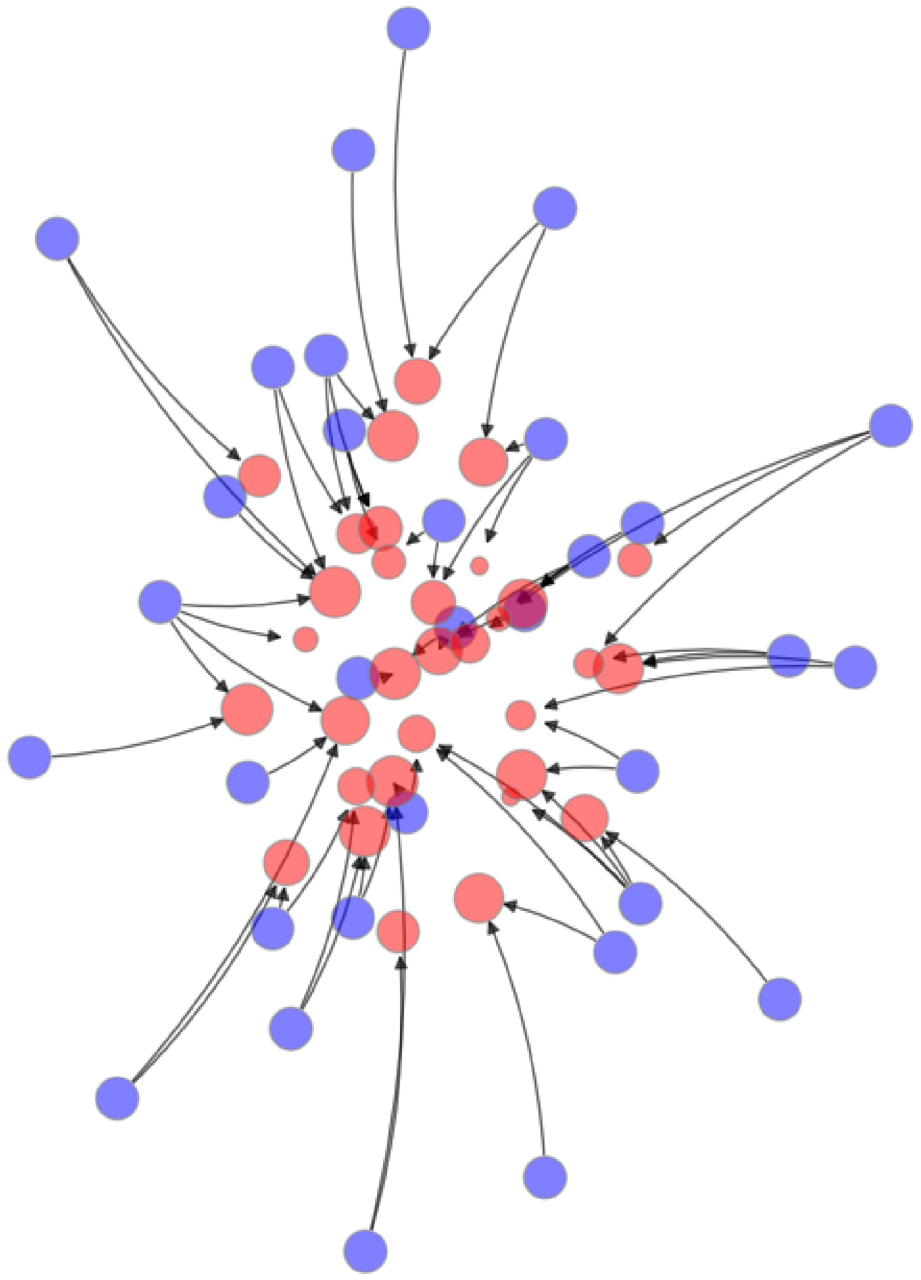
- node size is proportional to probability mass
- an edge (arrow) from i to j is drawn when a positive transfer is made from i to j under the optimal transport plan.

```
g = nx.DiGraph()
g.add_nodes_from([p.name for p in p_list])
g.add_nodes_from([q.name for q in q_list])

for i in range(n_p):
    for j in range(n_q):
        if pi[i, j] > 0:
            g.add_edge(p_list[i].name, q_list[j].name, weight=pi[i, j])

node_pos_dict={}
for p in p_list:
    node_pos_dict[p.name] = (p.x, p.y)

for q in q_list:
    node_pos_dict[q.name] = (q.x, q.y)

node_color_list = []
node_size_list = []
scale = 8_000
for p in p_list:
    node_color_list.append('blue')
    node_size_list.append(p.mass * scale)
for q in q_list:
    node_color_list.append('red')
    node_size_list.append(q.mass * scale)
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(figsize=(7, 10))
plt.axis('off')

nx.draw_networkx_nodes(g,
                       node_pos_dict,
                       node_color=node_color_list,
                       node_size=node_size_list,
                       edgecolors='grey',
                       linewidths=1,
                       alpha=0.5,
                       ax=ax)

nx.draw_networkx_edges(g,
                       node_pos_dict,
                       arrows=True,
                       connectionstyle='arc3,rad=0.1',
                       alpha=0.6)
plt.show()
```



CHAPTER

NINE

VON NEUMANN GROWTH MODEL (AND A GENERALIZATION)

This lecture uses the class `Neumann` to calculate key objects of a linear growth model of John von Neumann [von Neumann, 1937] that was generalized by Kemeny, Morgenstern and Thompson [Kemeny *et al.*, 1956].

Objects of interest are the maximal expansion rate (α), the interest factor (β), the optimal intensities (x), and prices (p).

In addition to watching how the towering mind of John von Neumann formulated an equilibrium model of price and quantity vectors in balanced growth, this lecture shows how fruitfully to employ the following important tools:

- a zero-sum two-player game
- linear programming
- the Perron-Frobenius theorem

We'll begin with some imports:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve, linprog
from textwrap import dedent

np.set_printoptions(precision=2)
```

The code below provides the `Neumann` class

```python
class Neumann(object):

    """
    This class describes the Generalized von Neumann growth model as it was
    discussed in Kemeny et al. (1956, ECTA) and Gale (1960, Chapter 9.5):

    Let:
    n ... number of goods
    m ... number of activities
    A ... input matrix is m-by-n
        a_{i,j} - amount of good j consumed by activity i
    B ... output matrix is m-by-n
        b_{i,j} - amount of good j produced by activity i

    x ... intensity vector (m-vector) with non-negative entries
        x'B - the vector of goods produced
        x'A - the vector of goods consumed
    p ... price vector (n-vector) with non-negative entries
        Bp - the revenue vector for every activity
        Ap - the cost of each activity
```

(continues on next page)

(continued from previous page)

```
    Both A and B have non-negative entries. Moreover, we assume that
    (1) Assumption I (every good which is consumed is also produced):
        for all j, b_{.,j} > 0, i.e. at least one entry is strictly positive
    (2) Assumption II (no free lunch):
        for all i, a_{i,.} > 0, i.e. at least one entry is strictly positive

    Parameters
    ----------
    A : array_like or scalar(float)
        Part of the state transition equation.  It should be `n x n`
    B : array_like or scalar(float)
        Part of the state transition equation.  It should be `n x k`
    """

    def __init__(self, A, B):

        self.A, self.B = list(map(self.convert, (A, B)))
        self.m, self.n = self.A.shape

        # Check if (A, B) satisfy the basic assumptions
        assert self.A.shape == self.B.shape, 'The input and output matrices \
              must have the same dimensions!'
        assert (self.A >= 0).all() and (self.B >= 0).all(), 'The input and \
              output matrices must have only non-negative entries!'

        # (1) Check whether Assumption I is satisfied:
        if (np.sum(B, 0) <= 0).any():
            self.AI = False
        else:
            self.AI = True

        # (2) Check whether Assumption II is satisfied:
        if (np.sum(A, 1) <= 0).any():
            self.AII = False
        else:
            self.AII = True

    def __repr__(self):
        return self.__str__()

    def __str__(self):

        me = """
        Generalized von Neumann expanding model:
          - number of goods          : {n}
          - number of activities     : {m}

        Assumptions:
          - AI:  every column of B has a positive entry    : {AI}
          - AII: every row of A has a positive entry       : {AII}

        """
        # Irreducible                                       : {irr}
        return dedent(me.format(n=self.n, m=self.m,
                                AI=self.AI, AII=self.AII))
```

(continues on next page)

(continued from previous page)

```
    def convert(self, x):
        """
        Convert array_like objects (lists of lists, floats, etc.) into
        well-formed 2D NumPy arrays
        """
        return np.atleast_2d(np.asarray(x))


    def bounds(self):
        """
        Calculate the trivial upper and lower bounds for alpha (expansion rate)
        and beta (interest factor). See the proof of Theorem 9.8 in Gale (1960)
        """

        n, m = self.n, self.m
        A, B = self.A, self.B

        f = lambda α: ((B - α * A) @ np.ones((n, 1))).max()
        g = lambda β: (np.ones((1, m)) @ (B - β * A)).min()

        UB = fsolve(f, 1).item()  # Upper bound for α, β
        LB = fsolve(g, 2).item()  # Lower bound for α, β

        return LB, UB


    def zerosum(self, γ, dual=False):
        """
        Given gamma, calculate the value and optimal strategies of a
        two-player zero-sum game given by the matrix

                M(gamma) = B - gamma * A

        Row player maximizing, column player minimizing

        Zero-sum game as an LP (primal --> α)

            max (0', 1) @ (x', v)
            subject to
            [-M', ones(n, 1)] @ (x', v)' <= 0
            (x', v) @ (ones(m, 1), 0) = 1
            (x', v) >= (0', -inf)

        Zero-sum game as an LP (dual --> beta)

            min (0', 1) @ (p', u)
            subject to
            [M, -ones(m, 1)] @ (p', u)' <= 0
            (p', u) @ (ones(n, 1), 0) = 1
            (p', u) >= (0', -inf)

        Outputs:
        --------
        value: scalar
            value of the zero-sum game
```

(continues on next page)

(continued from previous page)

```
        strategy: vector
            if dual = False, it is the intensity vector,
            if dual = True, it is the price vector
        """

        A, B, n, m = self.A, self.B, self.n, self.m
        M = B - γ * A

        if dual == False:
            # Solve the primal LP (for details see the description)
            # (1) Define the problem for v as a maximization (linprog minimizes)
            c = np.hstack([np.zeros(m), -1])

            # (2) Add constraints :
            # ... non-negativity constraints
            bounds = tuple(m * [(0, None)] + [(None, None)])
            # ... inequality constraints
            A_iq = np.hstack([-M.T, np.ones((n, 1))])
            b_iq = np.zeros((n, 1))
            # ... normalization
            A_eq = np.hstack([np.ones(m), 0]).reshape(1, m + 1)
            b_eq = 1

            res = linprog(c, A_ub=A_iq, b_ub=b_iq, A_eq=A_eq, b_eq=b_eq,
                          bounds=bounds)

        else:
            # Solve the dual LP (for details see the description)
            # (1) Define the problem for v as a maximization (linprog minimizes)
            c = np.hstack([np.zeros(n), 1])

            # (2) Add constraints :
            # ... non-negativity constraints
            bounds = tuple(n * [(0, None)] + [(None, None)])
            # ... inequality constraints
            A_iq = np.hstack([M, -np.ones((m, 1))])
            b_iq = np.zeros((m, 1))
            # ... normalization
            A_eq = np.hstack([np.ones(n), 0]).reshape(1, n + 1)
            b_eq = 1

            res = linprog(c, A_ub=A_iq, b_ub=b_iq, A_eq=A_eq, b_eq=b_eq,
                          bounds=bounds)

        if res.status != 0:
            print(res.message)

        # Pull out the required quantities
        value = res.x[-1]
        strategy = res.x[:-1]

        return value, strategy


    def expansion(self, tol=1e-8, maxit=1000):
```

(continues on next page)

(continued from previous page)

```
        """
        The algorithm used here is described in Hamburger-Thompson-Weil
        (1967, ECTA). It is based on a simple bisection argument and utilizes
        the idea that for a given γ (= α or β), the matrix "M = B - γ * A"
        defines a two-player zero-sum game, where the optimal strategies are
        the (normalized) intensity and price vector.

        Outputs:
        --------
        alpha: scalar
            optimal expansion rate
        """

        LB, UB = self.bounds()

        for iter in range(maxit):

            γ = (LB + UB) / 2
            ZS = self.zerosum(γ=γ)
            V = ZS[0]     # value of the game with γ

            if V >= 0:
                LB = γ
            else:
                UB = γ

            if abs(UB - LB) < tol:
                γ = (UB + LB) / 2
                x = self.zerosum(γ=γ)[1]
                p = self.zerosum(γ=γ, dual=True)[1]
                break

        return γ, x, p

    def interest(self, tol=1e-8, maxit=1000):
        """
        The algorithm used here is described in Hamburger-Thompson-Weil
        (1967, ECTA). It is based on a simple bisection argument and utilizes
        the idea that for a given gamma (= alpha or beta),
        the matrix "M = B - γ * A" defines a two-player zero-sum game,
        where the optimal strategies are the (normalized) intensity and price
        vector

        Outputs:
        --------
        beta: scalar
            optimal interest rate
        """

        LB, UB = self.bounds()

        for iter in range(maxit):
            γ = (LB + UB) / 2
            ZS = self.zerosum(γ=γ, dual=True)
            V = ZS[0]
```

(continues on next page)

(continued from previous page)

```
            if V > 0:
                LB = γ
            else:
                UB = γ

            if abs(UB - LB) < tol:
                γ = (UB + LB) / 2
                p = self.zerosum(γ=γ, dual=True)[1]
                x = self.zerosum(γ=γ)[1]
                break

        return γ, x, p
```

9.1 Notation

We use the following notation.

0 denotes a vector of zeros.

We call an n-vector positive and write $x \gg \mathbf{0}$ if $x_i > 0$ for all $i = 1, 2, \dots, n$.

We call a vector non-negative and write $x \geq \mathbf{0}$ if $x_i \geq 0$ for all $i = 1, 2, \dots, n$.

We call a vector semi-positive and written $x > \mathbf{0}$ if $x \geq \mathbf{0}$ and $x \neq \mathbf{0}$.

For two conformable vectors x and y, $x \gg y$, $x \geq y$ and $x > y$ mean $x - y \gg \mathbf{0}$, $x - y \geq \mathbf{0}$, and $x - y > \mathbf{0}$, respectively.

We let all vectors in this lecture be column vectors; x^T denotes the transpose of x (i.e., a row vector).

Let ι_n denote a column vector composed of n ones, i.e. $\iota_n = (1, 1, \dots, 1)^T$.

Let e^i denote a vector (of arbitrary size) containing zeros except for the i th position where it is one.

We denote matrices by capital letters. For an arbitrary matrix A, $a_{i,j}$ represents the entry in its i th row and j th column.

$a_{\cdot j}$ and $a_{i\cdot}$ denote the j th column and i th row of A, respectively.

9.2 Model Ingredients and Assumptions

A pair (A, B) of $m \times n$ non-negative matrices defines an economy.

- m is the number of *activities* (or sectors)
- n is the number of *goods* (produced and/or consumed).
- A is called the *input matrix*; $a_{i,j}$ denotes the amount of good j consumed by activity i
- B is called the *output matrix*; $b_{i,j}$ represents the amount of good j produced by activity i

Two key assumptions restrict economy (A, B):

- **Assumption I:** (every good that is consumed is also produced)

$$b_{.,j} > \mathbf{0} \qquad \forall j = 1, 2, \dots, n$$

- **Assumption II:** (no free lunch)

$$a_{i,.} > \mathbf{0} \qquad \forall i = 1, 2, \dots, m$$

A semi-positive *intensity* m-vector x denotes levels at which activities are operated.

Therefore,

- vector $x^T A$ gives the total amount of *goods used in production*
- vector $x^T B$ gives *total outputs*

An economy (A, B) is said to be *productive*, if there exists a non-negative intensity vector $x \geq 0$ such that $x^T B > x^T A$.

The semi-positive n-vector p contains prices assigned to the n goods.

The p vector implies *cost* and *revenue* vectors

- the vector Ap tells *costs* of the vector of activities
- the vector Bp tells *revenues* from the vector of activities

Satisfaction or a property of an input-output pair (A, B) called *irreducibility* (or indecomposability) determines whether an economy can be decomposed into multiple "sub-economies".

Definition: For an economy (A, B), the set of goods $S \subset \{1, 2, \dots, n\}$ is called an *independent subset* if it is possible to produce every good in S without consuming goods from outside S. Formally, the set S is independent if $\exists T \subset \{1, 2, \dots, m\}$ (a subset of activities) such that $a_{i,j} = 0 \; \forall i \in T$ and $j \in S^c$ and for all $j \in S$, $\exists i \in T$ for which $b_{i,j} > 0$. The economy is **irreducible** if there are no proper independent subsets.

We study two examples, both in Chapter 9.6 of Gale [Gale, 1989]

```
# (1) Irreducible (A, B) example: α_0 = β_0
A1 = np.array([[0, 1, 0, 0],
               [1, 0, 0, 1],
               [0, 0, 1, 0]])

B1 = np.array([[1, 0, 0, 0],
               [0, 0, 2, 0],
               [0, 1, 0, 1]])

# (2) Reducible (A, B) example: β_0 < α_0
A2 = np.array([[0, 1, 0, 0, 0, 0],
               [1, 0, 1, 0, 0, 0],
               [0, 0, 0, 1, 0, 0],
               [0, 0, 1, 0, 0, 1],
               [0, 0, 0, 0, 1, 0]])

B2 = np.array([[1, 0, 0, 1, 0, 0],
               [0, 1, 0, 0, 0, 0],
               [0, 0, 1, 0, 0, 0],
               [0, 0, 0, 0, 2, 0],
               [0, 0, 0, 1, 0, 1]])
```

The following code sets up our first Neumann economy or `Neumann` instance

```
n1 = Neumann(A1, B1)
n1
```

```
Generalized von Neumann expanding model:
  - number of goods          : 4
  - number of activities     : 3

Assumptions:
```

(continues on next page)

(continued from previous page)

```
  - AI:  every column of B has a positive entry    : True
  - AII: every row of A has a positive entry       : True
```

Here is a second instance of a Neumann economy

```
n2 = Neumann(A2, B2)
n2
```

```
Generalized von Neumann expanding model:
  - number of goods          : 6
  - number of activities     : 5

Assumptions:
  - AI:  every column of B has a positive entry    : True
  - AII: every row of A has a positive entry       : True
```

9.3 Dynamic Interpretation

Attach a time index t to the preceding objects, regard an economy as a dynamic system, and study sequences

$$\{(A_t, B_t)\}_{t\geq 0}, \qquad \{x_t\}_{t\geq 0}, \qquad \{p_t\}_{t\geq 0}$$

An interesting special case holds the technology process constant and investigates the dynamics of quantities and prices only.

Accordingly, in the rest of this lecture, we assume that $(A_t, B_t) = (A, B)$ for all $t \geq 0$.

A crucial element of the dynamic interpretation involves the timing of production.

We assume that production (consumption of inputs) takes place in period t, while the consequent output materializes in period $t+1$, i.e., consumption of $x_t^T A$ in period t results in $x_t^T B$ amounts of output in period $t+1$.

These timing conventions imply the following feasibility condition:

$$x_t^T B \geq x_{t+1}^T A \qquad \forall t \geq 1$$

which asserts that no more goods can be used today than were produced yesterday.

Accordingly, Ap_t tells the costs of production in period t and Bp_t tells revenues in period $t+1$.

9.3.1 Balanced Growth

We follow John von Neumann in studying “balanced growth”.

Let $./$ denote an elementwise division of one vector by another and let $\alpha > 0$ be a scalar.

Then *balanced growth* is a situation in which

$$x_{t+1}./x_t = \alpha, \quad \forall t \geq 0$$

With balanced growth, the law of motion of x is evidently $x_{t+1} = \alpha x_t$ and so we can rewrite the feasibility constraint as

$$x_t^T B \geq \alpha x_t^T A \qquad \forall t$$

In the same spirit, define $\beta \in \mathbb{R}$ as the **interest factor** per unit of time.

We assume that it is always possible to earn a gross return equal to the constant interest factor β by investing "outside the model".

Under this assumption about outside investment opportunities, a no-arbitrage condition gives rise to the following (no profit) restriction on the price sequence:

$$\beta A p_t \geq B p_t \qquad \forall t$$

This says that production cannot yield a return greater than that offered by the outside investment opportunity (here we compare values in period $t+1$).

The balanced growth assumption allows us to drop time subscripts and conduct an analysis purely in terms of a time-invariant growth rate α and interest factor β.

9.4 Duality

Two problems are connected by a remarkable dual relationship between technological and valuation characteristics of the economy:

Definition: The *technological expansion problem* (TEP) for the economy (A, B) is to find a semi-positive m-vector $x > 0$ and a number $\alpha \in \mathbb{R}$ that satisfy

$$\begin{aligned} \max_{\alpha} \quad & \alpha \\ \text{s.t.} \quad & x^T B \geq \alpha x^T A \end{aligned}$$

Theorem 9.3 of David Gale's book [Gale, 1989] asserts that if Assumptions I and II are both satisfied, then a maximum value of α exists and that it is positive.

The maximal value is called the *technological expansion rate* and is denoted by α_0. The associated intensity vector x_0 is the *optimal intensity vector*.

Definition: The economic expansion problem (EEP) for (A, B) is to find a semi-positive n-vector $p > 0$ and a number $\beta \in \mathbb{R}$ that satisfy

$$\begin{aligned} \min_{\beta} \quad & \beta \\ \text{s.t.} \quad & Bp \leq \beta A p \end{aligned}$$

Assumptions I and II imply existence of a minimum value $\beta_0 > 0$ called the *economic expansion rate*.

The corresponding price vector p_0 is the *optimal price vector*.

Because the criterion functions in the *technological expansion* problem and the *economical expansion problem* are both linearly homogeneous, the optimality of x_0 and p_0 are defined only up to a positive scale factor.

For convenience (and to emphasize a close connection to zero-sum games), we normalize both vectors x_0 and p_0 to have unit length.

A standard duality argument (see Lemma 9.4. in (Gale, 1960) [Gale, 1989]) implies that under Assumptions I and II, $\beta_0 \leq \alpha_0$.

But to deduce that $\beta_0 \geq \alpha_0$, Assumptions I and II are not sufficient.

Therefore, von Neumann [von Neumann, 1937] went on to prove the following remarkable "duality" result that connects TEP and EEP.

Theorem 1 (von Neumann): If the economy (A, B) satisfies Assumptions I and II, then there exist (γ^*, x_0, p_0), where $\gamma^* \in [\beta_0, \alpha_0] \subset \mathbb{R}$, $x_0 > 0$ is an m-vector, $p_0 > 0$ is an n-vector, and the following arbitrage true

$$\begin{aligned} x_0^T B &\geq \gamma^* x_0^T A \\ B p_0 &\leq \gamma^* A p_0 \\ x_0^T \left(B - \gamma^* A\right) p_0 &= 0 \end{aligned}$$

Note: *Proof (Sketch):* Assumption I and II imply that there exist (α_0, x_0) and (β_0, p_0) that solve the TEP and EEP, respectively. If $\gamma^* > \alpha_0$, then by definition of α_0, there cannot exist a semi-positive x that satisfies $x^T B \geq \gamma^* x^T A$. Similarly, if $\gamma^* < \beta_0$, there is no semi-positive p for which $Bp \leq \gamma^* Ap$. Let $\gamma^* \in [\beta_0, \alpha_0]$, then $x_0^T B \geq \alpha_0 x_0^T A \geq \gamma^* x_0^T A$. Moreover, $Bp_0 \leq \beta_0 A p_0 \leq \gamma^* A p_0$. These two inequalities imply $x_0 \left(B - \gamma^* A\right) p_0 = 0$.

Here the constant γ^* is both an expansion factor and an interest factor (not necessarily optimal).

We have already encountered and discussed the first two inequalities that represent feasibility and no-profit conditions.

Moreover, the equality $x_0^T \left(B - \gamma^* A\right) p_0 = 0$ concisely expresses the requirements that if any good grows at a rate larger than γ^* (i.e., if it is *oversupplied*), then its price must be zero; and that if any activity provides negative profit, it must be unused.

Therefore, the conditions stated in Theorem I ex encode all equilibrium conditions.

So Theorem I essentially states that under Assumptions I and II there always exists an equilibrium (γ^*, x_0, p_0) with balanced growth.

Note that Theorem I is silent about uniqueness of the equilibrium. In fact, it does not rule out (trivial) cases with $x_0^T B p_0 = 0$ so that nothing of value is produced.

To exclude such uninteresting cases, Kemeny, Morgenstern and Thomspson [Kemeny *et al.*, 1956] add an extra requirement

$$x_0^T B p_0 > 0$$

and call the associated equilibria *economic solutions*.

They show that this extra condition does not affect the existence result, while it significantly reduces the number of (relevant) solutions.

9.5 Interpretation as Two-player Zero-sum Game

To compute the equilibrium (γ^*, x_0, p_0), we follow the algorithm proposed by Hamburger, Thompson and Weil (1967), building on the key insight that an equilibrium (with balanced growth) can be solves a particular two-player zero-sum game. First, we introduce some notation.

Consider the $m \times n$ matrix C as a payoff matrix, with the entries representing payoffs from the **minimizing** column player to the **maximizing** row player and assume that the players can use mixed strategies. Thus,

- the row player chooses the m-vector $x > \mathbf{0}$ subject to $\iota_m^T x = 1$
- the column player chooses the n-vector $p > \mathbf{0}$ subject to $\iota_n^T p = 1$.

Definition: The $m \times n$ matrix game C has the *solution* $(x^*, p^*, V(C))$ in mixed strategies if

$$(x^*)^T C e^j \geq V(C) \quad \forall j \in \{1, \dots, n\} \qquad \text{and} \qquad (e^i)^T C p^* \leq V(C) \quad \forall i \in \{1, \dots, m\}$$

The number $V(C)$ is called the *value* of the game.

From the above definition, it is clear that the value $V(C)$ has two alternative interpretations:

- by playing the appropriate mixed stategy, the maximizing player can assure himself at least $V(C)$ (no matter what the column player chooses)
- by playing the appropriate mixed stategy, the minimizing player can make sure that the maximizing player will not get more than $V(C)$ (irrespective of what is the maximizing player's choice)

A famous theorem of Nash (1951) tells us that there always exists a mixed strategy Nash equilibrium for any *finite* two-player zero-sum game.

Moreover, von Neumann's Minmax Theorem [von Neumann, 1928] implies that

$$V(C) = \max_x \min_p \quad x^T C p = \min_p \max_x \quad x^T C p = (x^*)^T C p^*$$

9.5.1 Connection with Linear Programming (LP)

Nash equilibria of a finite two-player zero-sum game solve a linear programming problem.

To see this, we introduce the following notation

- For a fixed x, let v be the value of the minimization problem: $v \equiv \min_p x^T C p = \min_j x^T C e^j$
- For a fixed p, let u be the value of the maximization problem: $u \equiv \max_x x^T C p = \max_i (e^i)^T C p$

Then the *max-min problem* (the game from the maximizing player's point of view) can be written as the *primal* LP

$$\begin{aligned} V(C) = & \max \quad v \\ \text{s.t.} \quad & v \iota_n^T \leq x^T C \\ & x \geq \mathbf{0} \\ & \iota_n^T x = 1 \end{aligned}$$

while the *min-max problem* (the game from the minimizing player's point of view) is the *dual* LP

$$\begin{aligned} V(C) = & \min \quad u \\ \text{s.t.} \quad & u \iota_m \geq C p \\ & p \geq \mathbf{0} \\ & \iota_m^T p = 1 \end{aligned}$$

Hamburger, Thompson and Weil [Hamburger *et al.*, 1967] view the input-output pair of the economy as payoff matrices of two-player zero-sum games.

Using this interpretation, they restate Assumption I and II as follows

$$V(-A) < 0 \quad \text{and} \quad V(B) > 0$$

Note: *Proof (Sketch)*:

- $\Rightarrow V(B) > 0$ implies $x_0^T B \gg \mathbf{0}$, where x_0 is a maximizing vector. Since B is non-negative, this requires that each column of B has at least one positive entry, which is Assumption I.
- $\Leftarrow$ From Assumption I and the fact that $p > \mathbf{0}$, it follows that $Bp > \mathbf{0}$. This implies that the maximizing player can always choose x so that $x^T B p > 0$ so that it must be the case that $V(B) > 0$.

In order to (re)state Theorem I in terms of a particular two-player zero-sum game, we define a matrix for $\gamma \in \mathbb{R}$

$$M(\gamma) \equiv B - \gamma A$$

For fixed γ, treating $M(\gamma)$ as a matrix game, calculating the solution of the game implies

- If $\gamma > \alpha_0$, then for all $x > 0$, there $\exists j \in \{1, \dots, n\}$, s.t. $[x^T M(\gamma)]_j < 0$ implying that $V(M(\gamma)) < 0$.
- If $\gamma < \beta_0$, then for all $p > 0$, there $\exists i \in \{1, \dots, m\}$, s.t. $[M(\gamma)p]_i > 0$ implying that $V(M(\gamma)) > 0$.
- If $\gamma \in \{\beta_0, \alpha_0\}$, then (by Theorem I) the optimal intensity and price vectors x_0 and p_0 satisfy

$$x_0^T M(\gamma) \geq \mathbf{0}^T \qquad \text{and} \qquad M(\gamma)p_0 \leq \mathbf{0}$$

That is, $(x_0, p_0, 0)$ is a solution of the game $M(\gamma)$ so that $V(M(\beta_0)) = V(M(\alpha_0)) = 0$.

- If $\beta_0 < \alpha_0$ and $\gamma \in (\beta_0, \alpha_0)$, then $V(M(\gamma)) = 0$.

Moreover, if x' is optimal for the maximizing player in $M(\gamma')$ for $\gamma' \in (\beta_0, \alpha_0)$ and p'' is optimal for the minimizing player in $M(\gamma'')$ where $\gamma'' \in (\beta_0, \gamma')$, then $(x', p'', 0)$ is a solution for $M(\gamma)$ $\forall \gamma \in (\gamma'', \gamma')$.

Proof (Sketch): If x' is optimal for a maximizing player in game $M(\gamma')$, then $(x')^T M(\gamma') \geq \mathbf{0}^T$ and so for all $\gamma < \gamma'$.

$$(x')^T M(\gamma) = (x')^T M(\gamma') + (x')^T(\gamma' - \gamma)A \geq \mathbf{0}^T$$

hence $V(M(\gamma)) \geq 0$. If p'' is optimal for a minimizing player in game $M(\gamma'')$, then $M(\gamma)p \leq \mathbf{0}$ and so for all $\gamma'' < \gamma$

$$M(\gamma)p'' = M(\gamma'') + (\gamma'' - \gamma)Ap'' \leq \mathbf{0}$$

hence $V(M(\gamma)) \leq 0$.

It is clear from the above argument that β_0, α_0 are the minimal and maximal γ for which $V(M(\gamma)) = 0$.

Furthermore, Hamburger et al. [Hamburger *et al.*, 1967] show that the function $\gamma \mapsto V(M(\gamma))$ is continuous and nonincreasing in γ.

This suggests an algorithm to compute (α_0, x_0) and (β_0, p_0) for a given input-output pair (A, B).

9.5.2 Algorithm

Hamburger, Thompson and Weil [Hamburger *et al.*, 1967] propose a simple bisection algorithm to find the minimal and maximal roots (i.e. β_0 and α_0) of the function $\gamma \mapsto V(M(\gamma))$.

Step 1

First, notice that we can easily find trivial upper and lower bounds for α_0 and β_0.

- TEP requires that $x^T(B - \alpha A) \geq \mathbf{0}^T$ and $x > \mathbf{0}$, so if α is so large that $\max_i\{[(B - \alpha A)\iota_n]_i\} < 0$, then TEP ceases to have a solution.

Accordingly, let **UB** be the α^* that solves $\max_i\{[(B - \alpha^* A)\iota_n]_i\} = 0$.

- Similar to the upper bound, if β is so low that $\min_j\{[\iota_m^T(B - \beta A)]_j\} > 0$, then the EEP has no solution and so we can define **LB** as the β^* that solves $\min_j\{[\iota_m^T(B - \beta^* A)]_j\} = 0$.

The *bounds* method calculates these trivial bounds for us

```
n1.bounds()
```

```
(1.0, 2.0)
```

Step 2

Compute α_0 and β_0

- Finding α_0
  1. Fix $\gamma = \frac{UB+LB}{2}$ and compute the solution of the two-player zero-sum game associated with $M(\gamma)$. We can use either the primal or the dual LP problem.
  2. If $V(M(\gamma)) \geq 0$, then set $LB = \gamma$, otherwise let $UB = \gamma$.
  3. Iterate on 1. and 2. until $|UB - LB| < \epsilon$.
- Finding β_0
  1. Fix $\gamma = \frac{UB+LB}{2}$ and compute the solution of the two-player zero-sum game associated. with $M(\gamma)$. We can use either the primal or the dual LP problem.
  2. If $V(M(\gamma)) > 0$, then set $LB = \gamma$, otherwise let $UB = \gamma$.
  3. Iterate on 1. and 2. until $|UB - LB| < \epsilon$.
- *Existence*: Since $V(M(LB)) > 0$ and $V(M(UB)) < 0$ and $V(M(\cdot))$ is a continuous, nonincreasing function, there is at least one $\gamma \in [LB, UB]$, s.t. $V(M(\gamma)) = 0$.

The *zerosum* method calculates the value and optimal strategies associated with a given γ.

```
γ = 2

print(f'Value of the game with γ = {γ}')
print(n1.zerosum(γ=γ)[0])
print('Intensity vector (from the primal)')
print(n1.zerosum(γ=γ)[1])
print('Price vector (from the dual)')
print(n1.zerosum(γ=γ, dual=True)[1])
```

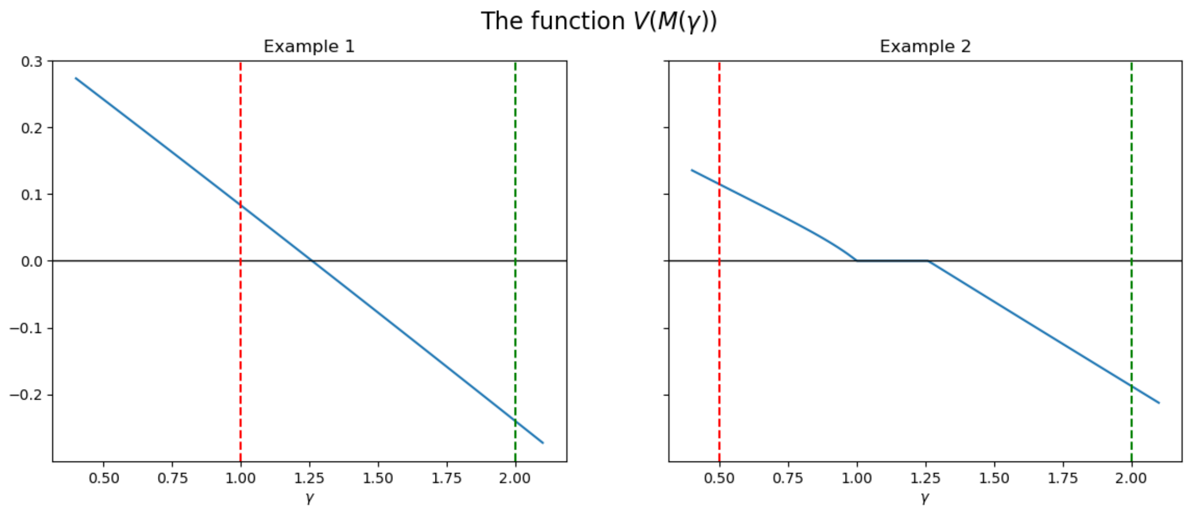
```
Value of the game with γ = 2
-0.24
Intensity vector (from the primal)
[0.32 0.28 0.4 ]
Price vector (from the dual)
[0.4  0.32 0.28 0.  ]
```

```
numb_grid = 100
γ_grid = np.linspace(0.4, 2.1, numb_grid)

value_ex1_grid = np.asarray([n1.zerosum(γ=γ_grid[i])[0]
                            for i in range(numb_grid)])
value_ex2_grid = np.asarray([n2.zerosum(γ=γ_grid[i])[0]
                            for i in range(numb_grid)])

fig, axes = plt.subplots(1, 2, figsize=(14, 5), sharey=True)
fig.suptitle(r'The function $V(M(\gamma))$', fontsize=16)

for ax, grid, N, i in zip(axes, (value_ex1_grid, value_ex2_grid),
                           (n1, n2), (1, 2)):
    ax.plot(γ_grid, grid)
    ax.set(title=f'Example {i}', xlabel='$\gamma$')
    ax.axhline(0, c='k', lw=1)
```

(continues on next page)

(continued from previous page)

```
    ax.axvline(N.bounds()[0], c='r', ls='--', label='lower bound')
    ax.axvline(N.bounds()[1], c='g', ls='--', label='upper bound')

plt.show()
```



The *expansion* method implements the bisection algorithm for α_0 (and uses the primal LP problem for x_0)

```
α_0, x, p = n1.expansion()
print(f'α_0 = {α_0}')
print(f'x_0 = {x}')
print(f'The corresponding p from the dual = {p}')
```

```
α_0 = 1.2599210478365421
x_0 = [0.33 0.26 0.41]
The corresponding p from the dual = [0.41 0.33 0.26 0.  ]
```

The *interest* method implements the bisection algorithm for β_0 (and uses the dual LP problem for p_0)

```
β_0, x, p = n1.interest()
print(f'β_0 = {β_0}')
print(f'p_0 = {p}')
print(f'The corresponding x from the primal = {x}')
```

```
β_0 = 1.2599210478365421
p_0 = [0.41 0.33 0.26 0.  ]
The corresponding x from the primal = [0.33 0.26 0.41]
```

Of course, when γ^* is unique, it is irrelevant which one of the two methods we use – both work.

In particular, as will be shown below, in case of an irreducible (A, B) (like in Example 1), the maximal and minimal roots of $V(M(\gamma))$ necessarily coincide implying a “full duality” result, i.e. $\alpha_0 = \beta_0 = \gamma^*$ so that the expansion (and interest) rate γ^* is unique.

9.5.3 Uniqueness and Irreducibility

As an illustration, compute first the maximal and minimal roots of $V(M(\cdot))$ for our Example 2 that has a reducible input-output pair (A, B)

```
α_0, x, p = n2.expansion()
print(f'α_0 = {α_0}')
print(f'x_0 = {x}')
print(f'The corresponding p from the dual = {p}')
```

```
α_0 = 1.259921052493155
x_0 = [5.27e-10 0.00e+00 3.27e-01 2.60e-01 4.13e-01]
The corresponding p from the dual = [0.   0.21 0.33 0.26 0.21 0.  ]
```

```
β_0, x, p = n2.interest()
print(f'β_0 = {β_0}')
print(f'p_0 = {p}')
print(f'The corresponding x from the primal = {x}')
```

```
β_0 = 1.0000000009313226
p_0 = [ 5.00e-01  5.00e-01 -1.55e-09 -1.24e-09 -9.31e-10  0.00e+00]
The corresponding x from the primal = [-0.    0.    0.25  0.25  0.5 ]
```

As we can see, with a reducible (A, B), the roots found by the bisection algorithms might differ, so there might be multiple γ^* that make the value of the game with $M(\gamma^*)$ zero. (see the figure above).

Indeed, although the von Neumann theorem assures existence of the equilibrium, Assumptions I and II are not sufficient for uniqueness. Nonetheless, Kemeny et al. (1967) show that there are at most finitely many economic solutions, meaning that there are only finitely many γ^* that satisfy $V(M(\gamma^*)) = 0$ and $x_0^T B p_0 > 0$ and that for each such γ_i^*, there is a self-contained part of the economy (a sub-economy) that in equilibrium can expand independently with the expansion coefficient γ_i^*.

The following theorem (see Theorem 9.10. in Gale [Gale, 1989]) asserts that imposing irreducibility is sufficient for uniqueness of (γ^*, x_0, p_0).

Theorem II: Adopt the conditions of Theorem 1. If the economy (A, B) is irreducible, then $\gamma^* = \alpha_0 = \beta_0$.

9.5.4 A Special Case

There is a special (A, B) that allows us to simplify the solution method significantly by invoking the powerful Perron-Frobenius theorem for non-negative matrices.

Definition: We call an economy *simple* if it satisfies

- $n = m$
- Each activity produces exactly one good
- Each good is produced by one and only one activity.

These assumptions imply that $B = I_n$, i.e., that B can be written as an identity matrix (possibly after reshuffling its rows and columns).

The simple model has the following special property (Theorem 9.11. in Gale [Gale, 1989]): if x_0 and $\alpha_0 > 0$ solve the TEP with (A, I_n), then

$$x_0^T = \alpha_0 x_0^T A \qquad \Leftrightarrow \qquad x_0^T A = \left(\frac{1}{\alpha_0}\right) x_0^T$$

The latter shows that $1/\alpha_0$ is a positive eigenvalue of A and x_0 is the corresponding non-negative left eigenvector.

The classic result of **Perron and Frobenius** implies that a non-negative matrix has a non-negative eigenvalue-eigenvector pair.

Moreover, if A is irreducible, then the optimal intensity vector x_0 is positive and *unique* up to multiplication by a positive scalar.

Suppose that A is reducible with k irreducible subsets $S_1, \dots, S_k$. Let A_i be the submatrix corresponding to S_i and let α_i and β_i be the associated expansion and interest factors, respectively. Then we have

$$\alpha_0 = \max_i \{\alpha_i\} \qquad \text{and} \qquad \beta_0 = \min_i \{\beta_i\}$$

Part III

Solution Methods

CHAPTER

TEN

USING NEWTON'S METHOD TO SOLVE ECONOMIC MODELS

See also:

GPU: A version of this lecture which makes use of jax to run the code on a `GPU` is available here

10.1 Overview

Many economic problems involve finding fixed points or zeros (sometimes called "roots") of functions.

For example, in a simple supply and demand model, an equilibrium price is one that makes excess demand zero.

In other words, an equilibrium is a zero of the excess demand function.

There are various computational techniques for solving for fixed points and zeros.

In this lecture we study an important gradient-based technique called Newton's method.

Newton's method does not always work but, in situations where it does, convergence is often fast when compared to other methods.

The lecture will apply Newton's method in one-dimensional and multi-dimensional settings to solve fixed-point and zero-finding problems.

- When finding the fixed point of a function f, Newton's method updates an existing guess of the fixed point by solving for the fixed point of a linear approximation to the function f.
- When finding the zero of a function f, Newton's method updates an existing guess by solving for the zero of a linear approximation to the function f.

To build intuition, we first consider an easy, one-dimensional fixed point problem where we know the solution and solve it using both successive approximation and Newton's method.

Then we apply Newton's method to multi-dimensional settings to solve market for equilibria with multiple goods.

At the end of the lecture we leverage the power of automatic differentiation in `autograd` to solve a very high-dimensional equilibrium problem

```
!pip install autograd
```

We use the following imports in this lecture

```
import matplotlib.pyplot as plt
from collections import namedtuple
from scipy.optimize import root
from autograd import jacobian
# Thinly-wrapped numpy to enable automatic differentiation
```

(continues on next page)

(continued from previous page)

```python
import autograd.numpy as np

plt.rcParams["figure.figsize"] = (10, 5.7)
```

10.2 Fixed Point Computation Using Newton's Method

In this section we solve the fixed point of the law of motion for capital in the setting of the Solow growth model.

We will inspect the fixed point visually, solve it by successive approximation, and then apply Newton's method to achieve faster convergence.

10.2.1 The Solow Model

In the Solow growth model, assuming Cobb-Douglas production technology and zero population growth, the law of motion for capital is

$$k_{t+1} = g(k_t) \quad \text{where} \quad g(k) := sAk^{\alpha} + (1 - \delta)k \tag{10.1}$$

Here

- k_t is capital stock per worker,
- $A, \alpha > 0$ are production parameters, $\alpha < 1$
- $s > 0$ is a savings rate, and
- $\delta \in (0, 1)$ is a rate of depreciation

In this example, we wish to calculate the unique strictly positive fixed point of g, the law of motion for capital.

In other words, we seek a $k^* > 0$ such that $g(k^*) = k^*$.

- such a k^* is called a steady state, since $k_t = k^*$ implies $k_{t+1} = k^*$.

Using pencil and paper to solve $g(k) = k$, you will be able to confirm that

$$k^* = \left(\frac{sA}{\delta}\right)^{1/(1-\alpha)}$$

10.2.2 Implementation

Let's store our parameters in `namedtuple` to help us keep our code clean and concise.

```python
SolowParameters = namedtuple("SolowParameters", ('A', 's', 'α', 'δ'))
```

This function creates a suitable `namedtuple` with default parameter values.

```python
def create_solow_params(A=2.0, s=0.3, α=0.3, δ=0.4):
    "Creates a Solow model parameterization with default values."
    return SolowParameters(A=A, s=s, α=α, δ=δ)
```

The next two functions implement the law of motion *(10.2.1)* and store the true fixed point k^*.

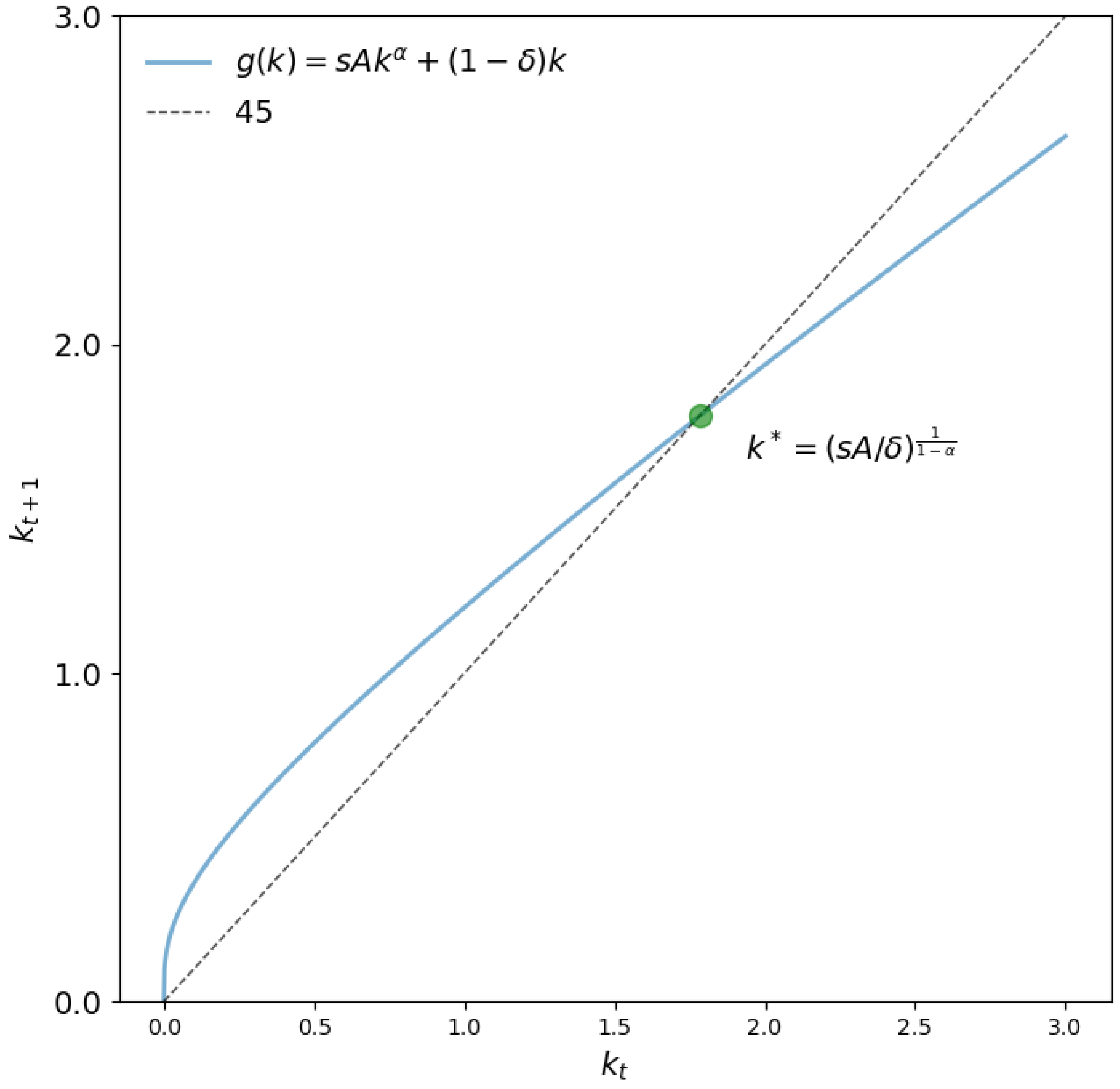
```
def g(k, params):
    A, s, α, δ = params
    return A * s * k**α + (1 - δ) * k

def exact_fixed_point(params):
    A, s, α, δ = params
    return ((s * A) / δ)**(1/(1 - α))
```

Here is a function to provide a 45 degree plot of the dynamics.

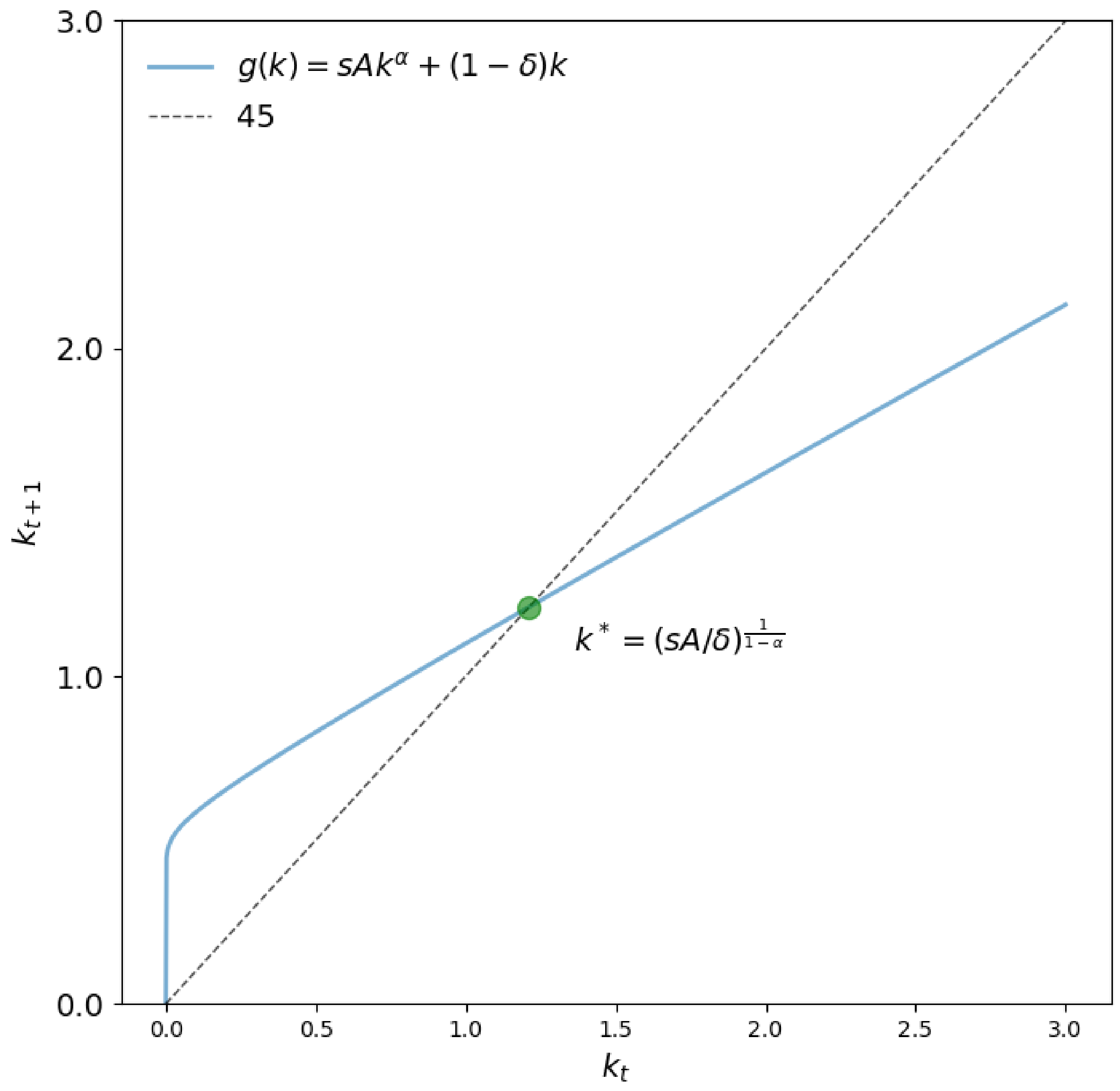
```
def plot_45(params, ax, fontsize=14):

    k_min, k_max = 0.0, 3.0
    k_grid = np.linspace(k_min, k_max, 1200)

    # Plot the functions
    lb = r"$g(k) = sAk^{\alpha} + (1 - \delta)k$"
    ax.plot(k_grid, g(k_grid, params),  lw=2, alpha=0.6, label=lb)
    ax.plot(k_grid, k_grid, "k--", lw=1, alpha=0.7, label="45")

    # Show and annotate the fixed point
    kstar = exact_fixed_point(params)
    fps = (kstar,)
    ax.plot(fps, fps, "go", ms=10, alpha=0.6)
    ax.annotate(r"$k^* = (sA / \delta)^{\frac{1}{1-\alpha}}$",
             xy=(kstar, kstar),
             xycoords="data",
             xytext=(20, -20),
             textcoords="offset points",
             fontsize=fontsize)

    ax.legend(loc="upper left", frameon=False, fontsize=fontsize)

    ax.set_yticks((0, 1, 2, 3))
    ax.set_yticklabels((0.0, 1.0, 2.0, 3.0), fontsize=fontsize)
    ax.set_ylim(0, 3)
    ax.set_xlabel("$k_t$", fontsize=fontsize)
    ax.set_ylabel("$k_{t+1}$", fontsize=fontsize)
```

Let’s look at the 45 degree diagram for two parameterizations.

```
params = create_solow_params()
fig, ax = plt.subplots(figsize=(8, 8))
plot_45(params, ax)
plt.show()
```



```
params = create_solow_params(α=0.05, δ=0.5)
fig, ax = plt.subplots(figsize=(8, 8))
plot_45(params, ax)
plt.show()
```



We see that k^* is indeed the unique positive fixed point.

Successive Approximation

First let's compute the fixed point using successive approximation.

In this case, successive approximation means repeatedly updating capital from some initial state k_0 using the law of motion.

Here's a time series from a particular choice of k_0.

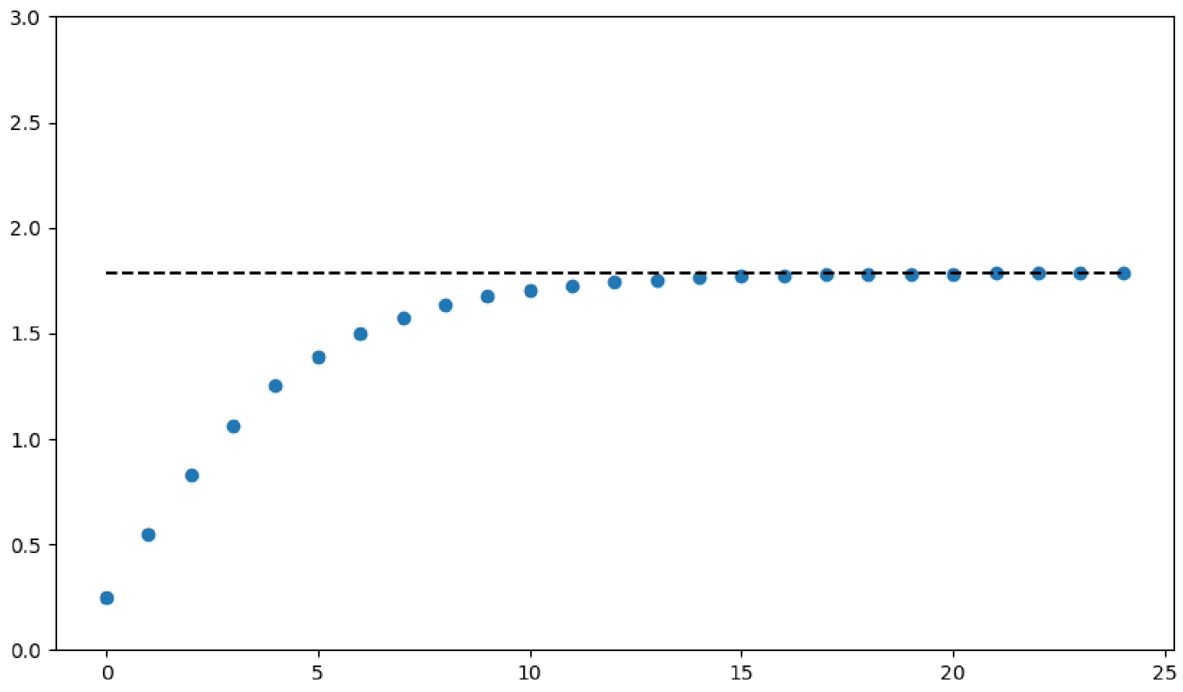
```python
def compute_iterates(k_0, f, params, n=25):
    "Compute time series of length n generated by arbitrary function f."
    k = k_0
    k_iterates = []
    for t in range(n):
```

(continues on next page)

(continued from previous page)

```
        k_iterates.append(k)
        k = f(k, params)
    return k_iterates
```

```
params = create_solow_params()
k_0 = 0.25
k_series = compute_iterates(k_0, g, params)
k_star = exact_fixed_point(params)

fig, ax = plt.subplots()
ax.plot(k_series, 'o')
ax.plot([k_star] * len(k_series), 'k--')
ax.set_ylim(0, 3)
plt.show()
```



Let's see the output for a long time series.

```
k_series = compute_iterates(k_0, g, params, n=10_000)
k_star_approx = k_series[-1]
k_star_approx
```

```
1.7846741842265788
```

This is close to the true value.

```
k_star
```

```
1.7846741842265788
```

Newton's Method

In general, when applying Newton's fixed point method to some function g, we start with a guess x_0 of the fixed point and then update by solving for the fixed point of a tangent line at x_0.

To begin with, we recall that the first-order approximation of g at x_0 (i.e., the first order Taylor approximation of g at x_0) is the function

$$\hat{g}(x) \approx g(x_0) + g'(x_0)(x - x_0) \tag{10.2}$$

We solve for the fixed point of $\hat{g}$ by calculating the x_1 that solves

$$x_1 = \frac{g(x_0) - g'(x_0)x_0}{1 - g'(x_0)}$$

Generalising the process above, Newton's fixed point method iterates on

$$x_{t+1} = \frac{g(x_t) - g'(x_t)x_t}{1 - g'(x_t)}, \quad x_0 \text{ given} \tag{10.3}$$

To implement Newton's method we observe that the derivative of the law of motion for capital *(10.2.1)* is

$$g'(k) = \alpha s A k^{\alpha-1} + (1 - \delta) \tag{10.4}$$

Let's define this:

```python
def Dg(k, params):
    A, s, α, δ = params
    return α * A * s * k**(α-1) + (1 - δ)
```

Here's a function q representing *(10.2.3)*.

```python
def q(k, params):
    return (g(k, params) - Dg(k, params) * k) / (1 - Dg(k, params))
```

Now let's plot some trajectories.

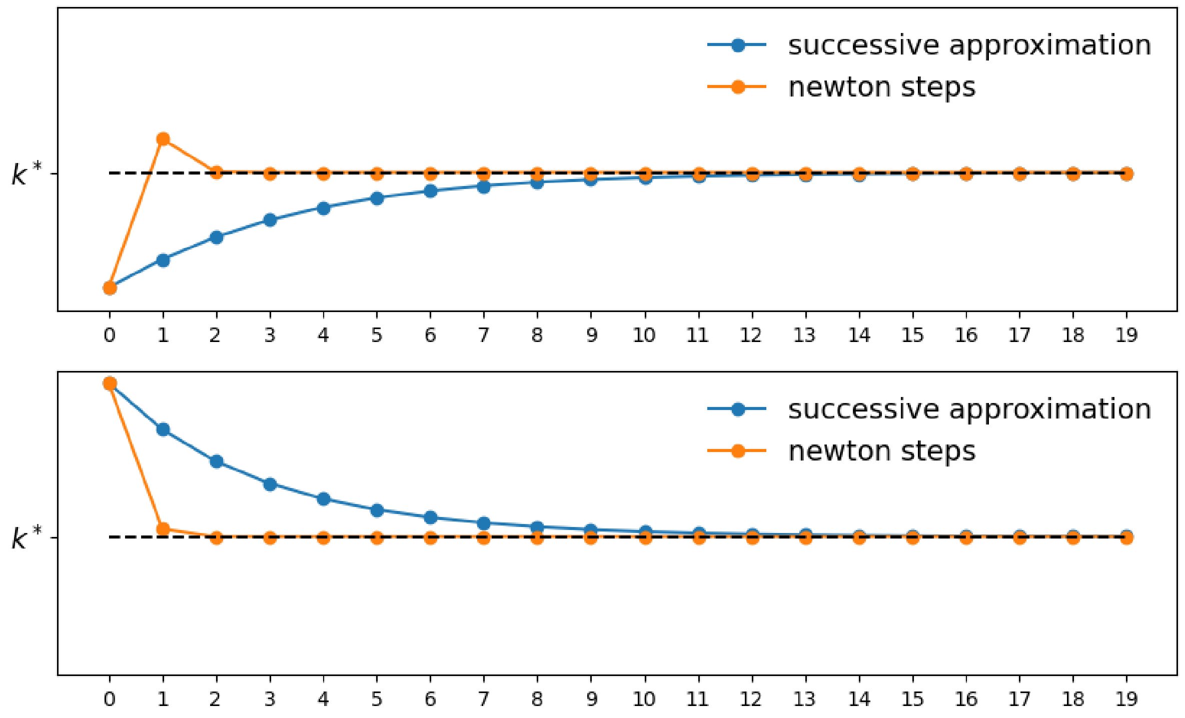
```python
def plot_trajectories(params,
                      k0_a=0.8,   # first initial condition
                      k0_b=3.1,   # second initial condition
                      n=20,       # length of time series
                      fs=14):     # fontsize

    fig, axes = plt.subplots(2, 1, figsize=(10, 6))
    ax1, ax2 = axes

    ks1 = compute_iterates(k0_a, g, params, n)
    ax1.plot(ks1, "-o", label="successive approximation")

    ks2 = compute_iterates(k0_b, g, params, n)
    ax2.plot(ks2, "-o", label="successive approximation")

    ks3 = compute_iterates(k0_a, q, params, n)
    ax1.plot(ks3, "-o", label="newton steps")

    ks4 = compute_iterates(k0_b, q, params, n)
    ax2.plot(ks4, "-o", label="newton steps")
```

(continues on next page)

(continued from previous page)

```
    for ax in axes:
        ax.plot(k_star * np.ones(n), "k--")
        ax.legend(fontsize=fs, frameon=False)
        ax.set_ylim(0.6, 3.2)
        ax.set_yticks((k_star,))
        ax.set_yticklabels(("$k^*$",), fontsize=fs)
        ax.set_xticks(np.linspace(0, 19, 20))

    plt.show()
```

```
params = create_solow_params()
plot_trajectories(params)
```



We can see that Newton's method converges faster than successive approximation.

10.3 Root-Finding in One Dimension

In the previous section we computed fixed points.

In fact Newton's method is more commonly associated with the problem of finding zeros of functions.

Let's discuss this "root-finding" problem and then show how it is connected to the problem of finding fixed points.

10.3.1 Newton's Method for Zeros

Let's suppose we want to find an x such that $f(x) = 0$ for some smooth function f mapping real numbers to real numbers.

Suppose we have a guess x_0 and we want to update it to a new point x_1.

As a first step, we take the first-order approximation of f around x_0:

$$\hat{f}(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

Now we solve for the zero of $\hat{f}$.

In particular, we set $\hat{f}(x_1) = 0$ and solve for x_1 to get

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}, \quad x_0 \text{ given}$$

Generalizing the formula above, for one-dimensional zero-finding problems, Newton's method iterates on

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}, \quad x_0 \text{ given} \tag{10.5}$$

The following code implements the iteration *(10.3.1)*

```python
def newton(f, Df, x_0, tol=1e-7, max_iter=100_000):
    x = x_0

    # Implement the zero-finding formula
    def q(x):
        return x - f(x) / Df(x)

    error = tol + 1
    n = 0
    while error > tol:
        n += 1
        if(n > max_iter):
            raise Exception('Max iteration reached without convergence')
        y = q(x)
        error = np.abs(x - y)
        x = y
        print(f'iteration {n}, error = {error:.5f}')
    return x
```

Numerous libraries implement Newton's method in one dimension, including SciPy, so the code is just for illustrative purposes.

(That said, when we want to apply Newton's method using techniques such as automatic differentiation or GPU acceleration, it will be helpful to know how to implement Newton's method ourselves.)

10.3.2 Application to Finding Fixed Points

Now consider again the Solow fixed-point calculation, where we solve for k satisfying $g(k) = k$.

We can convert to this to a zero-finding problem by setting $f(x) := g(x) - x$.

Any zero of f is clearly a fixed point of g.

Let's apply this idea to the Solow problem

```
params = create_solow_params()
k_star_approx_newton = newton(f=lambda x: g(x, params) - x,
                              Df=lambda x: Dg(x, params) - 1,
                              x_0=0.8)
```

```
iteration 1, error = 1.27209
iteration 2, error = 0.28180
iteration 3, error = 0.00561
iteration 4, error = 0.00000
iteration 5, error = 0.00000
```

```
k_star_approx_newton
```

```
1.7846741842265788
```

The result confirms the descent we saw in the graphs above: a very accurate result is reached with only 5 iterations.

10.4 Multivariate Newton's Method

In this section, we introduce a two-good problem, present a visualization of the problem, and solve for the equilibrium of the two-good market using both a zero finder in `SciPy` and Newton's method.

We then expand the idea to a larger market with 5,000 goods and compare the performance of the two methods again.

We will see a significant performance gain when using Netwon's method.

10.4.1 A Two Goods Market Equilibrium

Let's start by computing the market equilibrium of a two-good problem.

We consider a market for two related products, good 0 and good 1, with price vector $p = (p_0, p_1)$

Supply of good i at price p,

$$q_i^s(p) = b_i \sqrt{p_i}$$

Demand of good i at price p is,

$$q_i^d(p) = \exp(-(a_{i0}p_0 + a_{i1}p_1)) + c_i$$

Here c_i, b_i and a_{ij} are parameters.

For example, the two goods might be computer components that are typically used together, in which case they are complements. Hence demand depends on the price of both components.

The excess demand function is,

$$e_i(p) = q_i^d(p) - q_i^s(p), \quad i = 0, 1$$

An equilibrium price vector p^* satisfies $e_i(p^*) = 0$.

We set

$$A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}, \qquad b = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \qquad \text{and} \qquad c = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}$$

for this particular question.

A Graphical Exploration

Since our problem is only two-dimensional, we can use graphical analysis to visualize and help understand the problem.

Our first step is to define the excess demand function

$$e(p) = \begin{pmatrix} e_0(p) \\ e_1(p) \end{pmatrix}$$

The function below calculates the excess demand for given parameters

```python
def e(p, A, b, c):
    return np.exp(- A @ p) + c - b * np.sqrt(p)
```

Our default parameter values will be

$$A = \begin{pmatrix} 0.5 & 0.4 \\ 0.8 & 0.2 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \text{and} \qquad c = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

```python
A = np.array([
    [0.5, 0.4],
    [0.8, 0.2]
])
b = np.ones(2)
c = np.ones(2)
```

At a price level of $p = (1, 0.5)$, the excess demand is

```python
ex_demand = e((1.0, 0.5), A, b, c)

print(f'The excess demand for good 0 is {ex_demand[0]:.3f} \n'
      f'The excess demand for good 1 is {ex_demand[1]:.3f}')
```

```
The excess demand for good 0 is 0.497
The excess demand for good 1 is 0.699
```

Next we plot the two functions e_0 and e_1 on a grid of (p_0, p_1) values, using contour surfaces and lines.

We will use the following function to build the contour plots

```python
def plot_excess_demand(ax, good=0, grid_size=100, grid_max=4, surface=True):

    # Create a 100x100 grid
    p_grid = np.linspace(0, grid_max, grid_size)
    z = np.empty((100, 100))

    for i, p_1 in enumerate(p_grid):
        for j, p_2 in enumerate(p_grid):
            z[i, j] = e((p_1, p_2), A, b, c)[good]

    if surface:
        cs1 = ax.contourf(p_grid, p_grid, z.T, alpha=0.5)
        plt.colorbar(cs1, ax=ax, format="%.6f")

    ctr1 = ax.contour(p_grid, p_grid, z.T, levels=[0.0])
    ax.set_xlabel("$p_0$")
```

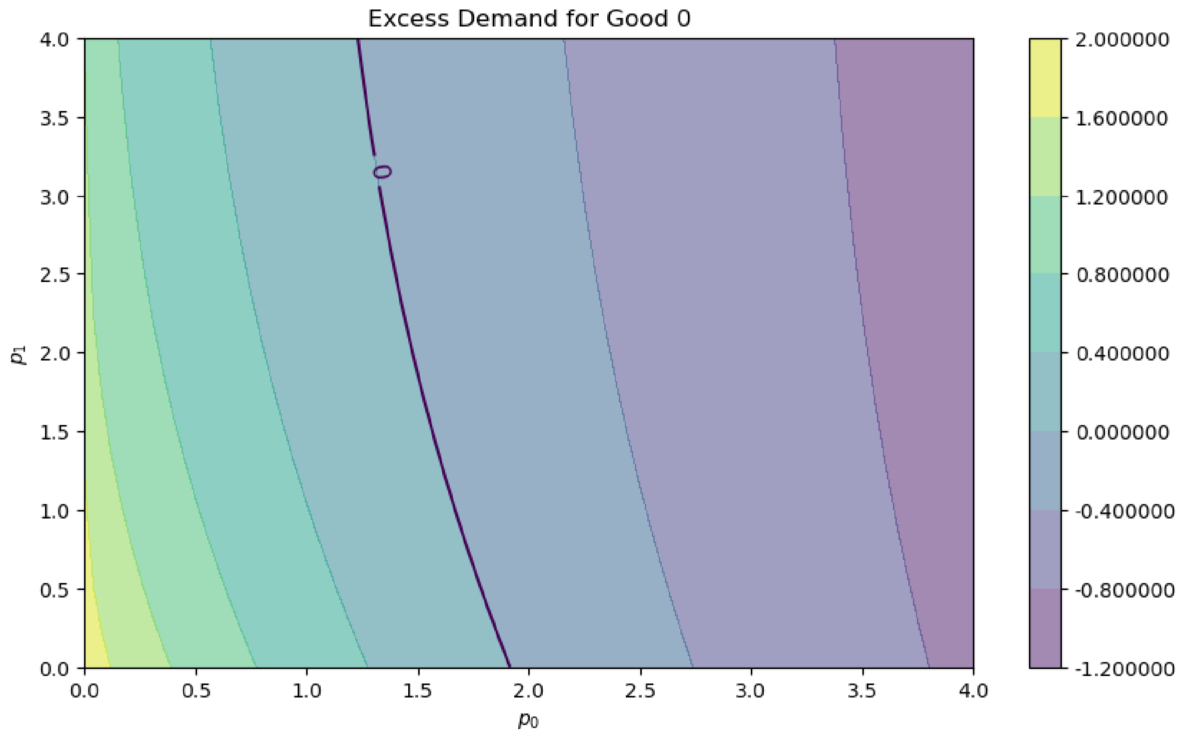
(continues on next page)

(continued from previous page)

```
    ax.set_ylabel("$p_1$")
    ax.set_title(f'Excess Demand for Good {good}')
    plt.clabel(ctr1, inline=1, fontsize=13)
```

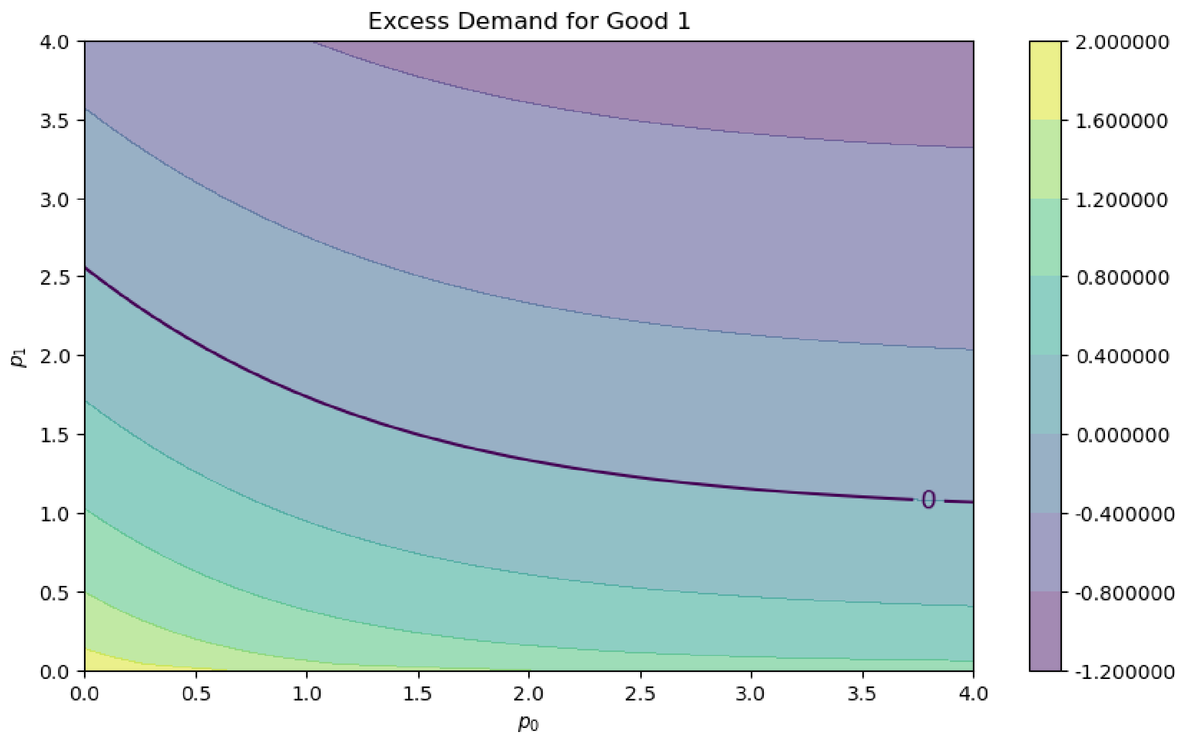
Here's our plot of e_0:

```
fig, ax = plt.subplots()
plot_excess_demand(ax, good=0)
plt.show()
```



Here's our plot of e_1:

```
fig, ax = plt.subplots()
plot_excess_demand(ax, good=1)
plt.show()
```

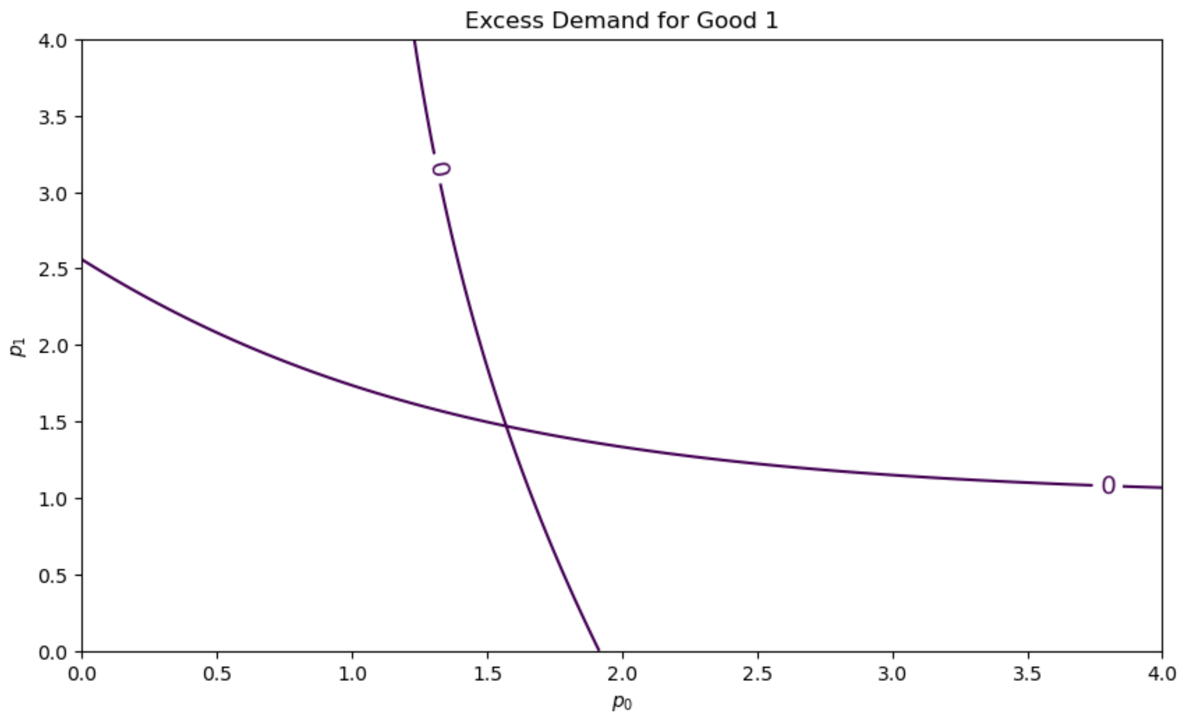


We see the black contour line of zero, which tells us when $e_i(p) = 0$.

For a price vector p such that $e_i(p) = 0$ we know that good i is in equilibrium (demand equals supply).

If these two contour lines cross at some price vector p^*, then p^* is an equilibrium price vector.

```
fig, ax = plt.subplots(figsize=(10, 5.7))
for good in (0, 1):
    plot_excess_demand(ax, good=good, surface=False)
plt.show()
```



It seems there is an equilibrium close to $p = (1.6, 1.5)$.

Using a Multidimensional Root Finder

To solve for p^* more precisely, we use a zero-finding algorithm from `scipy.optimize`.

We supply $p = (1, 1)$ as our initial guess.

```
init_p = np.ones(2)
```

This uses the modified Powell method to find the zero

```
%%time
solution = root(lambda p: e(p, A, b, c), init_p, method='hybr')
```

```
CPU times: user 132 µs, sys: 15 µs, total: 147 µs
Wall time: 145 µs
```

Here's the resulting value:

```
p = solution.x
p
```

```
array([1.57080182, 1.46928838])
```

This looks close to our guess from observing the figure. We can plug it back into e to test that $e(p) \approx 0$:

```
np.max(np.abs(e(p, A, b, c)))
```

```
2.0383694732117874e-13
```

This is indeed a very small error.

Adding Gradient Information

In many cases, for zero-finding algorithms applied to smooth functions, supplying the Jacobian of the function leads to better convergence properties.

Here we manually calculate the elements of the Jacobian

$$J(p) = \begin{pmatrix} \frac{\partial e_0}{\partial p_0}(p) & \frac{\partial e_0}{\partial p_1}(p) \\ \frac{\partial e_1}{\partial p_0}(p) & \frac{\partial e_1}{\partial p_1}(p) \end{pmatrix}$$

```python
def jacobian_e(p, A, b, c):
    p_0, p_1 = p
    a_00, a_01 = A[0, :]
    a_10, a_11 = A[1, :]
    j_00 = -a_00 * np.exp(-a_00 * p_0) - (b[0]/2) * p_0**(-1/2)
    j_01 = -a_01 * np.exp(-a_01 * p_1)
    j_10 = -a_10 * np.exp(-a_10 * p_0)
    j_11 = -a_11 * np.exp(-a_11 * p_1) - (b[1]/2) * p_1**(-1/2)
    J = [[j_00, j_01],
         [j_10, j_11]]
    return np.array(J)
```

```python
%%time
solution = root(lambda p: e(p, A, b, c),
                init_p,
                jac=lambda p: jacobian_e(p, A, b, c),
                method='hybr')
```

```
CPU times: user 266 µs, sys: 30 µs, total: 296 µs
Wall time: 257 µs
```

Now the solution is even more accurate (although, in this low-dimensional problem, the difference is quite small):

```python
p = solution.x
np.max(np.abs(e(p, A, b, c)))
```

```
1.3322676295501878e-15
```

Using Newton's Method

Now let's use Newton's method to compute the equilibrium price using the multivariate version of Newton's method

$$p_{n+1} = p_n - J_e(p_n)^{-1} e(p_n) \tag{10.6}$$

This is a multivariate version of *(10.3.1)*

(Here $J_e(p_n)$ is the Jacobian of e evaluated at p_n.)

The iteration starts from some initial guess of the price vector p_0.

Here, instead of coding Jacobian by hand, We use the `jacobian()` function in the `autograd` library to auto-differentiate and calculate the Jacobian.

With only slight modification, we can generalize *our previous attempt* to multi-dimensional problems

```
def newton(f, x_0, tol=1e-5, max_iter=10):
    x = x_0
    q = lambda x: x - np.linalg.solve(jacobian(f)(x), f(x))
    error = tol + 1
    n = 0
    while error > tol:
        n+=1
        if(n > max_iter):
            raise Exception('Max iteration reached without convergence')
        y = q(x)
        if(any(np.isnan(y))):
            raise Exception('Solution not found with NaN generated')
        error = np.linalg.norm(x - y)
        x = y
        print(f'iteration {n}, error = {error:.5f}')
    print('\n' + f'Result = {x} \n')
    return x
```

```
def e(p, A, b, c):
    return np.exp(- np.dot(A, p)) + c - b * np.sqrt(p)
```

We find the algorithm terminates in 4 steps

```
%%time
p = newton(lambda p: e(p, A, b, c), init_p)
```

```
iteration 1, error = 0.62515
iteration 2, error = 0.11152
iteration 3, error = 0.00258
iteration 4, error = 0.00000

Result = [1.57080182 1.46928838]

CPU times: user 2.36 ms, sys: 263 µs, total: 2.62 ms
Wall time: 2.22 ms
```

```
np.max(np.abs(e(p, A, b, c)))
```

```
1.461053500406706e-13
```

The result is very accurate.

With the larger overhead, the speed is not better than the optimized `scipy` function.

10.4.2 A High-Dimensional Problem

Our next step is to investigate a large market with 3,000 goods.

A JAX version of this section using GPU accelerated linear algebra and automatic differentiation is available here

The excess demand function is essentially the same, but now the matrix A is 3000×3000 and the parameter vectors b and c are 3000×1.

```
dim = 3000
np.random.seed(123)

# Create a random matrix A and normalize the rows to sum to one
A = np.random.rand(dim, dim)
A = np.asarray(A)
s = np.sum(A, axis=0)
A = A / s

# Set up b and c
b = np.ones(dim)
c = np.ones(dim)
```

Here's our initial condition

```
init_p = np.ones(dim)
```

```
%%time
p = newton(lambda p: e(p, A, b, c), init_p)
```

```
iteration 1, error = 23.22267
```

```
iteration 2, error = 3.94538
```

```
iteration 3, error = 0.08500
```

```
iteration 4, error = 0.00004
```

```
iteration 5, error = 0.00000

Result = [1.50185286 1.49865815 1.50028285 ... 1.50875149 1.48724784 1.48577532]

CPU times: user 29.4 s, sys: 156 ms, total: 29.5 s
Wall time: 27.6 s
```

```
np.max(np.abs(e(p, A, b, c)))
```

```
1.5543122344752192e-15
```

With the same tolerance, we compare the runtime and accuracy of Newton's method to SciPy's `root` function

```
%%time
solution = root(lambda p: e(p, A, b, c),
                init_p,
                jac=lambda p: jacobian(e)(p, A, b, c),
                method='hybr',
                tol=1e-5)
```

```
CPU times: user 32.4 s, sys: 88.3 ms, total: 32.5 s
Wall time: 32.2 s
```

```
p = solution.x
np.max(np.abs(e(p, A, b, c)))
```

```
8.295585953721485e-07
```

10.5 Exercises

Exercise 10.5.1

Consider a three-dimensional extension of the Solow fixed point problem with

$$A = \begin{pmatrix} 2 & 3 & 3 \\ 2 & 4 & 2 \\ 1 & 5 & 1 \end{pmatrix}, \quad s = 0.2, \quad \alpha = 0.5, \quad \delta = 0.8$$

As before the law of motion is

$$k_{t+1} = g(k_t) \quad \text{where} \quad g(k) := sAk^\alpha + (1-\delta)k$$

However k_t is now a 3×1 vector.

Solve for the fixed point using Newton's method with the following initial values:

$$\begin{aligned} k1_0 &= (1, 1, 1) \\ k2_0 &= (3, 5, 5) \\ k3_0 &= (50, 50, 50) \end{aligned}$$

Hint:

- The computation of the fixed point is equivalent to computing k^* such that $f(k^*) - k^* = 0$.
- If you are unsure about your solution, you can start with the solved example:

$$A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

with $s = 0.3$, $\alpha = 0.3$, and $\delta = 0.4$ and starting value:

$$k_0 = (1, 1, 1)$$

The result should converge to the *analytical solution*.

Solution to Exercise 10.5.1

Let's first define the parameters for this problem

```
A = np.array([[2.0, 3.0, 3.0],
              [2.0, 4.0, 2.0],
              [1.0, 5.0, 1.0]])

s = 0.2
α = 0.5
δ = 0.8

initLs = [np.ones(3),
          np.array([3.0, 5.0, 5.0]),
          np.repeat(50.0, 3)]
```

Then define the multivariate version of the formula for the *(10.2.1)*

```
def multivariate_solow(k, A=A, s=s, α=α, δ=δ):
    return (s * np.dot(A, k**α) + (1 - δ) * k)
```

Let's run through each starting value and see the output

```
attempt = 1
for init in initLs:
    print(f'Attempt {attempt}: Starting value is {init} \n')
    %time k = newton(lambda k: multivariate_solow(k) - k, \
                init)
    print('-'*64)
    attempt += 1
```

```
Attempt 1: Starting value is [1. 1. 1.]

iteration 1, error = 50.49630
iteration 2, error = 41.10937
iteration 3, error = 4.29413
iteration 4, error = 0.38543
iteration 5, error = 0.00544
iteration 6, error = 0.00000

Result = [3.84058108 3.87071771 3.41091933]

CPU times: user 4.51 ms, sys: 4 µs, total: 4.52 ms
Wall time: 3.87 ms
----------------------------------------------------------------
Attempt 2: Starting value is [3. 5. 5.]

iteration 1, error = 2.07011
iteration 2, error = 0.12642
iteration 3, error = 0.00060
iteration 4, error = 0.00000

Result = [3.84058108 3.87071771 3.41091933]
```

(continues on next page)

(continued from previous page)

```
CPU times: user 2.87 ms, sys: 0 ns, total: 2.87 ms
Wall time: 2.48 ms
----------------------------------------------------------------
Attempt 3: Starting value is [50. 50. 50.]

iteration 1, error = 73.00943
iteration 2, error = 6.49379
iteration 3, error = 0.68070
iteration 4, error = 0.01620
iteration 5, error = 0.00001
iteration 6, error = 0.00000

Result = [3.84058108 3.87071771 3.41091933]

CPU times: user 3.73 ms, sys: 0 ns, total: 3.73 ms
Wall time: 3.3 ms
----------------------------------------------------------------
```

We find that the results are invariant to the starting values given the well-defined property of this question.

But the number of iterations it takes to converge is dependent on the starting values.

Let substitute the output back to the formulate to check our last result

```
multivariate_solow(k) - k
```

```
array([-4.4408921e-16, -4.4408921e-16,  4.4408921e-16])
```

Note the error is very small.

We can also test our results on the known solution

```
A = np.array([[2.0, 0.0, 0.0],
               [0.0, 2.0, 0.0],
               [0.0, 0.0, 2.0]])

s = 0.3
α = 0.3
δ = 0.4

init = np.repeat(1.0, 3)


%time k = newton(lambda k: multivariate_solow(k, A=A, s=s, α=α, δ=δ) - k, \
                 init)
```

```
iteration 1, error = 1.57459
iteration 2, error = 0.21345
iteration 3, error = 0.00205
iteration 4, error = 0.00000

Result = [1.78467418 1.78467418 1.78467418]

CPU times: user 2.76 ms, sys: 0 ns, total: 2.76 ms
Wall time: 2.39 ms
```

The result is very close to the ground truth but still slightly different.

```
%time k = newton(lambda k: multivariate_solow(k, A=A, s=s, α=α, δ=δ) - k, \
                 init,\
                 tol=1e-7)
```

```
iteration 1, error = 1.57459
iteration 2, error = 0.21345
iteration 3, error = 0.00205
iteration 4, error = 0.00000
iteration 5, error = 0.00000

Result = [1.78467418 1.78467418 1.78467418]

CPU times: user 0 ns, sys: 3.73 ms, total: 3.73 ms
Wall time: 3.15 ms
```

We can see it steps towards a more accurate solution.

Exercise 10.5.2

In this exercise, let's try different initial values and check how Newton's method responds to different starting points.

Let's define a three-good problem with the following default values:

$$A = \begin{pmatrix} 0.2 & 0.1 & 0.7 \\ 0.3 & 0.2 & 0.5 \\ 0.1 & 0.8 & 0.1 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \qquad \text{and} \qquad c = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

For this exercise, use the following extreme price vectors as initial values:

$$\begin{aligned} p1_0 &= (5, 5, 5) \\ p2_0 &= (1, 1, 1) \\ p3_0 &= (4.5, 0.1, 4) \end{aligned}$$

Set the tolerance to 0.0 for more accurate output.

Solution to Exercise 10.5.2

Define parameters and initial values

```
A = np.array([
    [0.2, 0.1, 0.7],
    [0.3, 0.2, 0.5],
    [0.1, 0.8, 0.1]
])

b = np.array([1.0, 1.0, 1.0])
c = np.array([1.0, 1.0, 1.0])

initLs = [np.repeat(5.0, 3),
          np.ones(3),
          np.array([4.5, 0.1, 4.0])]
```

Let's run through each initial guess and check the output

```python
attempt = 1
for init in initLs:
    print(f'Attempt {attempt}: Starting value is {init} \n')
    %time p = newton(lambda p: e(p, A, b, c), \
                 init, \
                 tol=1e-15, \
                 max_iter=15)
    print('-'*64)
    attempt += 1
```

```
Attempt 1: Starting value is [5. 5. 5.]

iteration 1, error = 9.24381
```

```
/home/runner/miniconda3/envs/quantecon/lib/python3.11/site-packages/autograd/
↪tracer.py:48: RuntimeWarning: invalid value encountered in sqrt
  return f_raw(*args, **kwargs)
/home/runner/miniconda3/envs/quantecon/lib/python3.11/site-packages/autograd/numpy/
↪numpy_vjps.py:99: RuntimeWarning: invalid value encountered in power
  defvjp(anp.sqrt,    lambda ans, x : lambda g: g * 0.5 * x**-0.5)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
File <timed exec>:1

Cell In[34], line 12, in newton(f, x_0, tol, max_iter)
     10 y = q(x)
     11 if(any(np.isnan(y))):
---> 12     raise Exception('Solution not found with NaN generated')
     13 error = np.linalg.norm(x - y)
     14 x = y

Exception: Solution not found with NaN generated
```

```
----------------------------------------------------------------
Attempt 2: Starting value is [1. 1. 1.]

iteration 1, error = 0.73419
iteration 2, error = 0.12472
iteration 3, error = 0.00269
iteration 4, error = 0.00000
iteration 5, error = 0.00000
iteration 6, error = 0.00000

Result = [1.49744442 1.49744442 1.49744442]

CPU times: user 3.11 ms, sys: 0 ns, total: 3.11 ms
Wall time: 2.67 ms
----------------------------------------------------------------
Attempt 3: Starting value is [4.5 0.1 4. ]

iteration 1, error = 4.89202
iteration 2, error = 1.21206
iteration 3, error = 0.69421
```

(continues on next page)

(continued from previous page)

```
iteration 4, error = 0.16895
iteration 5, error = 0.00521
iteration 6, error = 0.00000
iteration 7, error = 0.00000
iteration 8, error = 0.00000

Result = [1.49744442 1.49744442 1.49744442]

CPU times: user 3.61 ms, sys: 0 ns, total: 3.61 ms
Wall time: 3.17 ms
-----------------------------------------------------------------
```

We can find that Newton's method may fail for some starting values.

Sometimes it may take a few initial guesses to achieve convergence.

Substitute the result back to the formula to check our result

```
e(p, A, b, c)
```

```
array([ 0.00000000e+00,  0.00000000e+00, -2.22044605e-16])
```

We can see the result is very accurate.

CHAPTER ELEVEN

DISCRETE STATE DYNAMIC PROGRAMMING

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
```

11.1 Overview

In this lecture we discuss a family of dynamic programming problems with the following features:

1. a discrete state space and discrete choices (actions)
2. an infinite horizon
3. discounted rewards
4. Markov state transitions

We call such problems discrete dynamic programs or discrete DPs.

Discrete DPs are the workhorses in much of modern quantitative economics, including

- monetary economics
- search and labor economics
- household savings and consumption theory
- investment theory
- asset pricing
- industrial organization, etc.

When a given model is not inherently discrete, it is common to replace it with a discretized version in order to use discrete DP techniques.

This lecture covers

- the theory of dynamic programming in a discrete setting, plus examples and applications
- a powerful set of routines for solving discrete DPs from the QuantEcon code library

Let's start with some imports:

```
import numpy as np
import matplotlib.pyplot as plt
import quantecon as qe
import scipy.sparse as sparse
```

(continues on next page)

(continued from previous page)

```
from quantecon import compute_fixed_point
from quantecon.markov import DiscreteDP
```

11.1.1 How to Read this Lecture

We use dynamic programming many applied lectures, such as

- The shortest path lecture
- The McCall search model lecture

The objective of this lecture is to provide a more systematic and theoretical treatment, including algorithms and implementation while focusing on the discrete case.

11.1.2 Code

Among other things, it offers

- a flexible, well-designed interface
- multiple solution methods, including value function and policy function iteration
- high-speed operations via carefully optimized JIT-compiled functions
- the ability to scale to large problems by minimizing vectorized operators and allowing operations on sparse matrices

JIT compilation relies on Numba, which should work seamlessly if you are using Anaconda as suggested.

11.1.3 References

For background reading on dynamic programming and additional applications, see, for example,

- [Ljungqvist and Sargent, 2018]
- [Hernandez-Lerma and Lasserre, 1996], section 3.5
- [Puterman, 2005]
- [Stokey *et al.*, 1989]
- [Rust, 1996]
- [Miranda and Fackler, 2002]
- EDTC, chapter 5

11.2 Discrete DPs

Loosely speaking, a discrete DP is a maximization problem with an objective function of the form

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t r(s_t, a_t) \tag{11.1}$$

where

- s_t is the state variable

- a_t is the action
- β is a discount factor
- $r(s_t, a_t)$ is interpreted as a current reward when the state is s_t and the action chosen is a_t

Each pair (s_t, a_t) pins down transition probabilities $Q(s_t, a_t, s_{t+1})$ for the next period state s_{t+1}.

Thus, actions influence not only current rewards but also the future time path of the state.

The essence of dynamic programming problems is to trade off current rewards vs favorable positioning of the future state (modulo randomness).

Examples:

- consuming today vs saving and accumulating assets
- accepting a job offer today vs seeking a better one in the future
- exercising an option now vs waiting

11.2.1 Policies

The most fruitful way to think about solutions to discrete DP problems is to compare *policies*.

In general, a policy is a randomized map from past actions and states to current action.

In the setting formalized below, it suffices to consider so-called *stationary Markov policies*, which consider only the current state.

In particular, a stationary Markov policy is a map σ from states to actions

- $a_t = \sigma(s_t)$ indicates that a_t is the action to be taken in state s_t

It is known that, for any arbitrary policy, there exists a stationary Markov policy that dominates it at least weakly.

- See section 5.5 of [Puterman, 2005] for discussion and proofs.

In what follows, stationary Markov policies are referred to simply as policies.

The aim is to find an optimal policy, in the sense of one that maximizes (11.1).

Let's now step through these ideas more carefully.

11.2.2 Formal Definition

Formally, a discrete dynamic program consists of the following components:

1. A finite set of *states* $S = \{0, \dots, n-1\}$.
2. A finite set of *feasible actions* $A(s)$ for each state $s \in S$, and a corresponding set of *feasible state-action pairs*.

$$SA := \{(s, a) \mid s \in S, \; a \in A(s)\}$$

3. A *reward function* $r \colon SA \to \mathbb{R}$.
4. A *transition probability function* $Q \colon SA \to \Delta(S)$, where $\Delta(S)$ is the set of probability distributions over S.
5. A *discount factor* $\beta \in [0, 1)$.

We also use the notation $A := \bigcup_{s \in S} A(s) = \{0, \ldots, m-1\}$ and call this set the *action space*.

A *policy* is a function $\sigma \colon S \to A$.

A policy is called *feasible* if it satisfies $\sigma(s) \in A(s)$ for all $s \in S$.

Denote the set of all feasible policies by Σ.

If a decision-maker uses a policy $\sigma \in \Sigma$, then

- the current reward at time t is $r(s_t, \sigma(s_t))$
- the probability that $s_{t+1} = s'$ is $Q(s_t, \sigma(s_t), s')$

For each $\sigma \in \Sigma$, define

- r_σ by $r_\sigma(s) := r(s, \sigma(s)))$
- Q_σ by $Q_\sigma(s, s') := Q(s, \sigma(s), s')$

Notice that Q_σ is a stochastic matrix on S.

It gives transition probabilities of the *controlled chain* when we follow policy σ.

If we think of r_σ as a column vector, then so is $Q_\sigma^t r_\sigma$, and the s-th row of the latter has the interpretation

$$(Q_\sigma^t r_\sigma)(s) = \mathbb{E}[r(s_t, \sigma(s_t)) \mid s_0 = s] \quad \text{when } \{s_t\} \sim Q_\sigma \tag{11.2}$$

Comments

- $\{s_t\} \sim Q_\sigma$ means that the state is generated by stochastic matrix Q_σ.
- See this discussion on computing expectations of Markov chains for an explanation of the expression in (11.2).

Notice that we're not really distinguishing between functions from S to $\mathbb{R}$ and vectors in $\mathbb{R}^n$.

This is natural because they are in one to one correspondence.

11.2.3 Value and Optimality

Let $v_\sigma(s)$ denote the discounted sum of expected reward flows from policy σ when the initial state is s.

To calculate this quantity we pass the expectation through the sum in (11.1) and use (11.2) to get

$$v_\sigma(s) = \sum_{t=0}^{\infty} \beta^t (Q_\sigma^t r_\sigma)(s) \qquad (s \in S)$$

This function is called the *policy value function* for the policy σ.

The *optimal value function*, or simply *value function*, is the function $v^* \colon S \to \mathbb{R}$ defined by

$$v^*(s) = \max_{\sigma \in \Sigma} v_\sigma(s) \qquad (s \in S)$$

(We can use max rather than sup here because the domain is a finite set)

A policy $\sigma \in \Sigma$ is called *optimal* if $v_\sigma(s) = v^*(s)$ for all $s \in S$.

Given any $w \colon S \to \mathbb{R}$, a policy $\sigma \in \Sigma$ is called w-greedy if

$$\sigma(s) \in \operatorname*{arg\,max}_{a \in A(s)} \left\{ r(s, a) + \beta \sum_{s' \in S} w(s') Q(s, a, s') \right\} \qquad (s \in S)$$

As discussed in detail below, optimal policies are precisely those that are v^*-greedy.

11.2.4 Two Operators

It is useful to define the following operators:

- The *Bellman operator* $T \colon \mathbb{R}^S \to \mathbb{R}^S$ is defined by

$$(Tv)(s) = \max_{a \in A(s)} \left\{ r(s,a) + \beta \sum_{s' \in S} v(s') Q(s,a,s') \right\} \qquad (s \in S)$$

- For any policy function $\sigma \in \Sigma$, the operator $T_\sigma \colon \mathbb{R}^S \to \mathbb{R}^S$ is defined by

$$(T_\sigma v)(s) = r(s, \sigma(s)) + \beta \sum_{s' \in S} v(s') Q(s, \sigma(s), s') \qquad (s \in S)$$

This can be written more succinctly in operator notation as

$$T_\sigma v = r_\sigma + \beta Q_\sigma v$$

The two operators are both monotone

- $v \leq w$ implies $Tv \leq Tw$ pointwise on S, and similarly for T_σ

They are also contraction mappings with modulus β

- $\|Tv - Tw\| \leq \beta \|v - w\|$ and similarly for T_σ, where $\|\cdot\|$ is the max norm

For any policy σ, its value v_σ is the unique fixed point of T_σ.

For proofs of these results and those in the next section, see, for example, EDTC, chapter 10.

11.2.5 The Bellman Equation and the Principle of Optimality

The main principle of the theory of dynamic programming is that

- the optimal value function v^* is a unique solution to the *Bellman equation*

$$v(s) = \max_{a \in A(s)} \left\{ r(s,a) + \beta \sum_{s' \in S} v(s') Q(s,a,s') \right\} \qquad (s \in S)$$

or in other words, v^* is the unique fixed point of T, and

- σ^* is an optimal policy function if and only if it is v^*-greedy

By the definition of greedy policies given above, this means that

$$\sigma^*(s) \in \operatorname*{arg\,max}_{a \in A(s)} \left\{ r(s,a) + \beta \sum_{s' \in S} v^*(s') Q(s,a,s') \right\} \qquad (s \in S)$$

11.3 Solving Discrete DPs

Now that the theory has been set out, let's turn to solution methods.

The code for solving discrete DPs is available in ddp.py from the QuantEcon.py code library.

It implements the three most important solution methods for discrete dynamic programs, namely

- value function iteration
- policy function iteration
- modified policy function iteration

Let's briefly review these algorithms and their implementation.

11.3.1 Value Function Iteration

Perhaps the most familiar method for solving all manner of dynamic programs is value function iteration.

This algorithm uses the fact that the Bellman operator T is a contraction mapping with fixed point v^*.

Hence, iterative application of T to any initial function $v^0 : S \to \mathbb{R}$ converges to v^*.

The details of the algorithm can be found in *the appendix*.

11.3.2 Policy Function Iteration

This routine, also known as Howard's policy improvement algorithm, exploits more closely the particular structure of a discrete DP problem.

Each iteration consists of

1. A policy evaluation step that computes the value v_σ of a policy σ by solving the linear equation $v = T_\sigma v$.
2. A policy improvement step that computes a v_σ-greedy policy.

In the current setting, policy iteration computes an exact optimal policy in finitely many iterations.

- See theorem 10.2.6 of EDTC for a proof.

The details of the algorithm can be found in *the appendix*.

11.3.3 Modified Policy Function Iteration

Modified policy iteration replaces the policy evaluation step in policy iteration with "partial policy evaluation".

The latter computes an approximation to the value of a policy σ by iterating T_σ for a specified number of times.

This approach can be useful when the state space is very large and the linear system in the policy evaluation step of policy iteration is correspondingly difficult to solve.

The details of the algorithm can be found in *the appendix*.

11.4 Example: A Growth Model

Let's consider a simple consumption-saving model.

A single household either consumes or stores its own output of a single consumption good.

The household starts each period with current stock s.

Next, the household chooses a quantity a to store and consumes $c = s - a$

- Storage is limited by a global upper bound M.
- Flow utility is $u(c) = c^\alpha$.

Output is drawn from a discrete uniform distribution on $\{0, \dots, B\}$.

The next period stock is therefore

$$s' = a + U \quad \text{where} \quad U \sim U[0, \dots, B]$$

The discount factor is $\beta \in [0, 1)$.

11.4.1 Discrete DP Representation

We want to represent this model in the format of a discrete dynamic program.

To this end, we take

- the state variable to be the stock s
- the state space to be $S = \{0, \dots, M + B\}$
  - hence $n = M + B + 1$
- the action to be the storage quantity a
- the set of feasible actions at s to be $A(s) = \{0, \dots, \min\{s, M\}\}$
  - hence $A = \{0, \dots, M\}$ and $m = M + 1$
- the reward function to be $r(s, a) = u(s - a)$
- the transition probabilities to be

$$Q(s, a, s') := \begin{cases} \frac{1}{B+1} & \text{if } a \leq s' \leq a + B \\ 0 & \text{otherwise} \end{cases} \tag{11.3}$$

11.4.2 Defining a DiscreteDP Instance

This information will be used to create an instance of DiscreteDP by passing the following information

1. An $n \times m$ reward array R.
2. An $n \times m \times n$ transition probability array Q.
3. A discount factor β.

For R we set $R[s, a] = u(s - a)$ if $a \leq s$ and $-\infty$ otherwise.

For Q we follow the rule in (11.3).

Note:

- The feasibility constraint is embedded into R by setting $R[s, a] = -\infty$ for $a \notin A(s)$.
- Probability distributions for (s, a) with $a \notin A(s)$ can be arbitrary.

The following code sets up these objects for us

```python
class SimpleOG:

    def __init__(self, B=10, M=5, α=0.5, β=0.9):
        """
        Set up R, Q and β, the three elements that define an instance of
        the DiscreteDP class.
        """

        self.B, self.M, self.α, self.β  = B, M, α, β
        self.n = B + M + 1
        self.m = M + 1

        self.R = np.empty((self.n, self.m))
```

(continues on next page)

(continued from previous page)

```
        self.Q = np.zeros((self.n, self.m, self.n))

        self.populate_Q()
        self.populate_R()

    def u(self, c):
        return c**self.α

    def populate_R(self):
        """
        Populate the R matrix, with R[s, a] = -np.inf for infeasible
        state-action pairs.
        """
        for s in range(self.n):
            for a in range(self.m):
                self.R[s, a] = self.u(s - a) if a <= s else -np.inf

    def populate_Q(self):
        """
        Populate the Q matrix by setting

            Q[s, a, s'] = 1 / (1 + B) if a <= s' <= a + B

        and zero otherwise.
        """

        for a in range(self.m):
            self.Q[:, a, a:(a + self.B + 1)] = 1.0 / (self.B + 1)
```

Let's run this code and create an instance of `SimpleOG`.

```
g = SimpleOG()  # Use default parameters
```

Instances of `DiscreteDP` are created using the signature `DiscreteDP(R, Q, β)`.

Let's create an instance using the objects stored in `g`

```
ddp = qe.markov.DiscreteDP(g.R, g.Q, g.β)
```

Now that we have an instance `ddp` of `DiscreteDP` we can solve it as follows

```
results = ddp.solve(method='policy_iteration')
```

Let's see what we've got here

```
dir(results)
```

```
['max_iter', 'mc', 'method', 'num_iter', 'sigma', 'v']
```

(In IPython version 4.0 and above you can also type `results.` and hit the tab key)

The most important attributes are `v`, the value function, and `σ`, the optimal policy

```
results.v
```

```
array([19.01740222, 20.01740222, 20.43161578, 20.74945302, 21.04078099,
       21.30873018, 21.54479816, 21.76928181, 21.98270358, 22.18824323,
       22.3845048 , 22.57807736, 22.76109127, 22.94376708, 23.11533996,
       23.27761762])
```

```
results.sigma
```

```
array([0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 4, 5, 5, 5, 5])
```

Since we've used policy iteration, these results will be exact unless we hit the iteration bound `max_iter`.

Let's make sure this didn't happen

```
results.max_iter
```

```
250
```

```
results.num_iter
```

```
3
```

Another interesting object is `results.mc`, which is the controlled chain defined by Q_{σ^*}, where σ^* is the optimal policy.

In other words, it gives the dynamics of the state when the agent follows the optimal policy.

Since this object is an instance of MarkovChain from QuantEcon.py (see this lecture for more discussion), we can easily simulate it, compute its stationary distribution and so on.

```
results.mc.stationary_distributions
```

```
array([[0.01732187, 0.04121063, 0.05773956, 0.07426848, 0.08095823,
        0.09090909, 0.09090909, 0.09090909, 0.09090909, 0.09090909,
        0.09090909, 0.07358722, 0.04969846, 0.03316953, 0.01664061,
        0.00995086]])
```

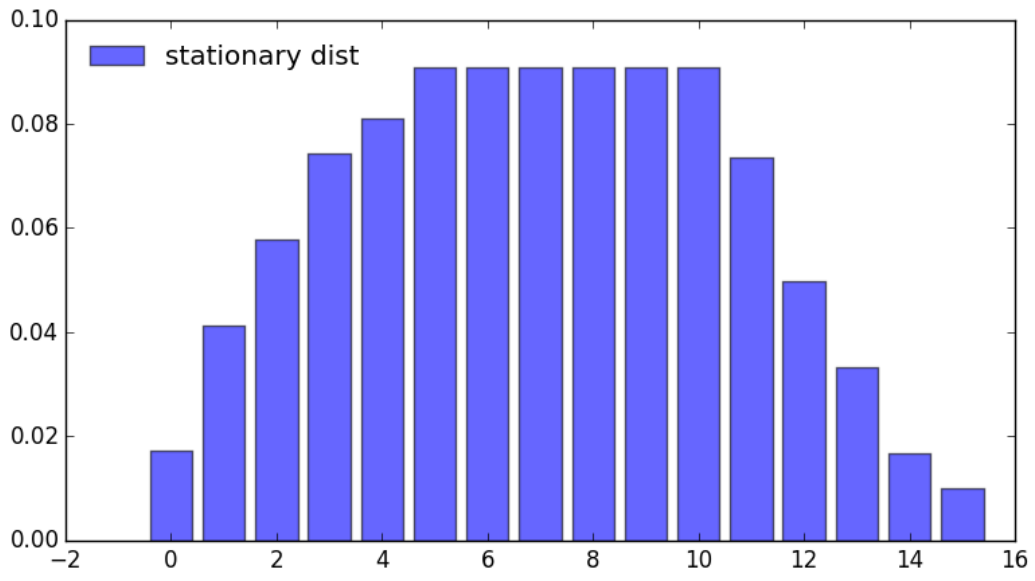
Here's the same information in a bar graph

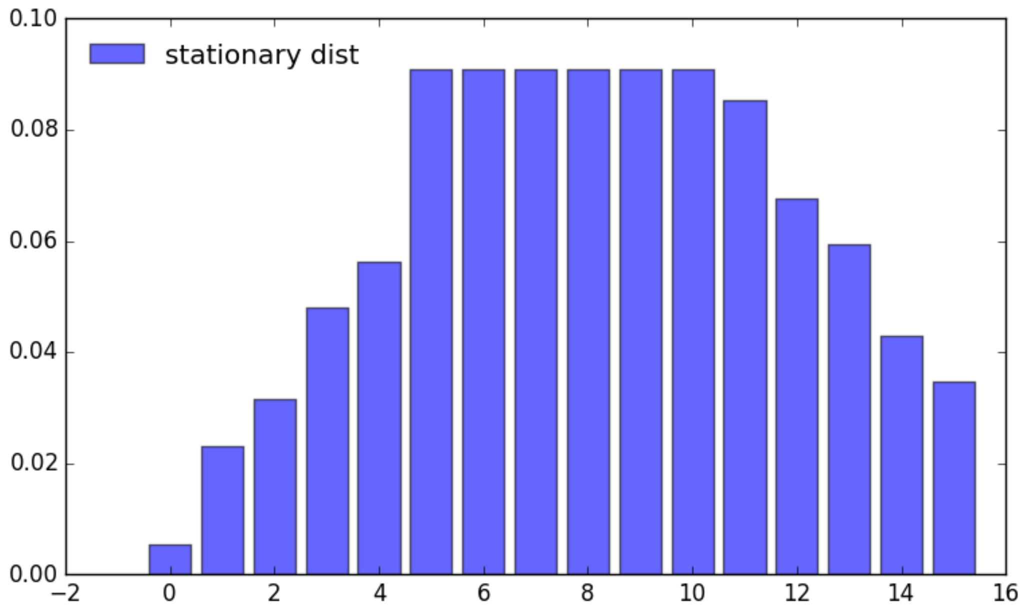
What happens if the agent is more patient?

```
ddp = qe.markov.DiscreteDP(g.R, g.Q, 0.99)  # Increase β to 0.99
results = ddp.solve(method='policy_iteration')
results.mc.stationary_distributions
```

```
array([[0.00546913, 0.02321342, 0.03147788, 0.04800681, 0.05627127,
        0.09090909, 0.09090909, 0.09090909, 0.09090909, 0.09090909,
        0.09090909, 0.08543996, 0.06769567, 0.05943121, 0.04290228,
        0.03463782]])
```

If we look at the bar graph we can see the rightward shift in probability mass


stationary dist
0.10
0.08
0.06
0.04
0.02
0.00
−2
0
2
4
6
8
10
12
14
16


stationary dist
0.10
0.08
0.06
0.04
0.02
0.00
−2
0
2
4
6
8
10
12
14
16

11.4.3 State-Action Pair Formulation

The `DiscreteDP` class in fact, provides a second interface to set up an instance.

One of the advantages of this alternative set up is that it permits the use of a sparse matrix for `Q`.

(An example of using sparse matrices is given in the exercises below)

The call signature of the second formulation is `DiscreteDP(R, Q, β, s_indices, a_indices)` where

- `s_indices` and `a_indices` are arrays of equal length `L` enumerating all feasible state-action pairs
- `R` is an array of length `L` giving corresponding rewards
- `Q` is an `L x n` transition probability array

Here's how we could set up these objects for the preceding example

```
B, M, α, β = 10, 5, 0.5, 0.9
n = B + M + 1
m = M + 1

def u(c):
    return c**α

s_indices = []
a_indices = []
Q = []
R = []
b = 1.0 / (B + 1)

for s in range(n):
    for a in range(min(M, s) + 1):  # All feasible a at this s
        s_indices.append(s)
        a_indices.append(a)
        q = np.zeros(n)
        q[a:(a + B + 1)] = b        # b on these values, otherwise 0
        Q.append(q)
        R.append(u(s - a))

ddp = qe.markov.DiscreteDP(R, Q, β, s_indices, a_indices)
```

For larger problems, you might need to write this code more efficiently by vectorizing or using Numba.

11.5 Exercises

In the stochastic optimal growth lecture from our introductory lecture series, we solve a benchmark model that has an analytical solution.

The exercise is to replicate this solution using `DiscreteDP`.

11.6 Solutions

11.6.1 Setup

Details of the model can be found in the lecture on optimal growth.

We let $f(k) = k^\alpha$ with $\alpha = 0.65$, $u(c) = \log c$, and $\beta = 0.95$

```
α = 0.65
f = lambda k: k**α
u = np.log
β = 0.95
```

Here we want to solve a finite state version of the continuous state model above.

We discretize the state space into a grid of size `grid_size=500`, from 10^{-6} to `grid_max=2`

```
grid_max = 2
grid_size = 500
grid = np.linspace(1e-6, grid_max, grid_size)
```

We choose the action to be the amount of capital to save for the next period (the state is the capital stock at the beginning of the period).

Thus the state indices and the action indices are both `0`, …, `grid_size-1`.

Action (indexed by) `a` is feasible at state (indexed by) `s` if and only if `grid[a] < f([grid[s])` (zero consumption is not allowed because of the log utility).

Thus the Bellman equation is:

$$v(k) = \max_{0<k'<f(k)} u(f(k) - k') + \beta v(k'),$$

where k' is the capital stock in the next period.

The transition probability array `Q` will be highly sparse (in fact it is degenerate as the model is deterministic), so we formulate the problem with state-action pairs, to represent `Q` in scipy sparse matrix format.

We first construct indices for state-action pairs:

```
# Consumption matrix, with nonpositive consumption included
C = f(grid).reshape(grid_size, 1) - grid.reshape(1, grid_size)

# State-action indices
s_indices, a_indices = np.where(C > 0)

# Number of state-action pairs
L = len(s_indices)

print(L)
print(s_indices)
print(a_indices)
```

```
118841
[  0   1   1 ... 499 499 499]
[  0   0   1 ... 389 390 391]
```

Reward vector `R` (of length `L`):

```
R = u(C[s_indices, a_indices])
```

(Degenerate) transition probability matrix `Q` (of shape `(L, grid_size)`), where we choose the scipy.sparse.lil_matrix format, while any format will do (internally it will be converted to the csr format):

```
Q = sparse.lil_matrix((L, grid_size))
Q[np.arange(L), a_indices] = 1
```

(If you are familiar with the data structure of scipy.sparse.csr_matrix, the following is the most efficient way to create the `Q` matrix in the current case)

```
# data = np.ones(L)
# indptr = np.arange(L+1)
# Q = sparse.csr_matrix((data, a_indices, indptr), shape=(L, grid_size))
```

Discrete growth model:

```
ddp = DiscreteDP(R, Q, β, s_indices, a_indices)
```

Notes

Here we intensively vectorized the operations on arrays to simplify the code.

As noted, however, vectorization is memory consumptive, and it can be prohibitively so for grids with large size.

11.6.2 Solving the Model

Solve the dynamic optimization problem:

```
res = ddp.solve(method='policy_iteration')
v, σ, num_iter = res.v, res.sigma, res.num_iter
num_iter
```

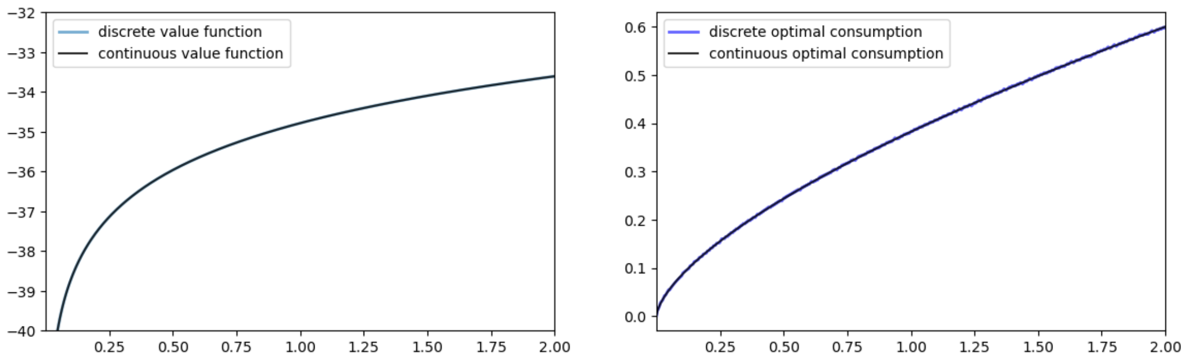
```
10
```

Note that `sigma` contains the *indices* of the optimal *capital stocks* to save for the next period. The following translates `sigma` to the corresponding consumption vector.

```
# Optimal consumption in the discrete version
c = f(grid) - grid[σ]

# Exact solution of the continuous version
ab = α * β
c1 = (np.log(1 - ab) + np.log(ab) * ab / (1 - ab)) / (1 - β)
c2 = α / (1 - ab)

def v_star(k):
    return c1 + c2 * np.log(k)

def c_star(k):
    return (1 - ab) * k**α
```

Let us compare the solution of the discrete model with that of the original continuous model

```
fig, ax = plt.subplots(1, 2, figsize=(14, 4))
ax[0].set_ylim(-40, -32)
ax[0].set_xlim(grid[0], grid[-1])
ax[1].set_xlim(grid[0], grid[-1])

lb0 = 'discrete value function'
ax[0].plot(grid, v, lw=2, alpha=0.6, label=lb0)

lb0 = 'continuous value function'
ax[0].plot(grid, v_star(grid), 'k-', lw=1.5, alpha=0.8, label=lb0)
ax[0].legend(loc='upper left')

lb1 = 'discrete optimal consumption'
ax[1].plot(grid, c, 'b-', lw=2, alpha=0.6, label=lb1)

lb1 = 'continuous optimal consumption'
ax[1].plot(grid, c_star(grid), 'k-', lw=1.5, alpha=0.8, label=lb1)
ax[1].legend(loc='upper left')
plt.show()
```



The outcomes appear very close to those of the continuous version.

Except for the “boundary” point, the value functions are very close:

```
np.abs(v - v_star(grid)).max()
```

```
121.49819147053378
```

```
np.abs(v - v_star(grid))[1:].max()
```

```
0.012681735127500815
```

The optimal consumption functions are close as well:

```
np.abs(c - c_star(grid)).max()
```

```
0.003826523100010082
```

In fact, the optimal consumption obtained in the discrete version is not really monotone, but the decrements are quite small:

```
diff = np.diff(c)
(diff >= 0).all()
```

```
False
```

```
dec_ind = np.where(diff < 0)[0]
len(dec_ind)
```

```
174
```

```
np.abs(diff[dec_ind]).max()
```

```
0.0019618533397668 39
```

The value function is monotone:

```
(np.diff(v) > 0).all()
```

```
True
```

11.6.3 Comparison of the Solution Methods

Let us solve the problem with the other two methods.

Value Iteration

```
ddp.epsilon = 1e-4
ddp.max_iter = 500
res1 = ddp.solve(method='value_iteration')
res1.num_iter
```

```
294
```

```
np.array_equal(σ, res1.sigma)
```

```
True
```

Modified Policy Iteration

```
res2 = ddp.solve(method='modified_policy_iteration')
res2.num_iter
```

```
16
```

```
np.array_equal(σ, res2.sigma)
```

```
True
```

Speed Comparison

```
%timeit ddp.solve(method='value_iteration')
%timeit ddp.solve(method='policy_iteration')
%timeit ddp.solve(method='modified_policy_iteration')
```

```
92.1 ms ± 138 μs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
9.27 ms ± 25.8 μs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
10.8 ms ± 123 μs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

As is often the case, policy iteration and modified policy iteration are much faster than value iteration.

11.6.4 Replication of the Figures

Using `DiscreteDP` we replicate the figures shown in the lecture.

Convergence of Value Iteration

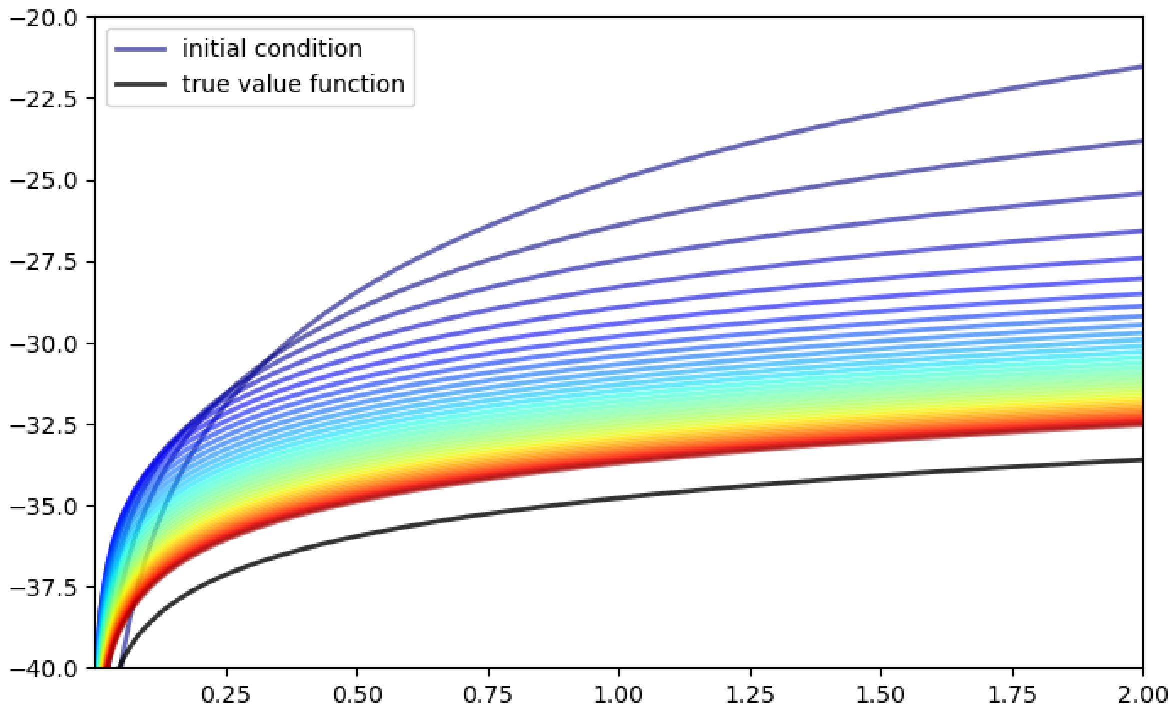
Let us first visualize the convergence of the value iteration algorithm as in the lecture, where we use `ddp.bellman_operator` implemented as a method of `DiscreteDP`

```
w = 5 * np.log(grid) - 25  # Initial condition
n = 35
fig, ax = plt.subplots(figsize=(8,5))
ax.set_ylim(-40, -20)
ax.set_xlim(np.min(grid), np.max(grid))
lb = 'initial condition'
ax.plot(grid, w, color=plt.cm.jet(0), lw=2, alpha=0.6, label=lb)
for i in range(n):
    w = ddp.bellman_operator(w)
    ax.plot(grid, w, color=plt.cm.jet(i / n), lw=2, alpha=0.6)
lb = 'true value function'
ax.plot(grid, v_star(grid), 'k-', lw=2, alpha=0.8, label=lb)
ax.legend(loc='upper left')
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

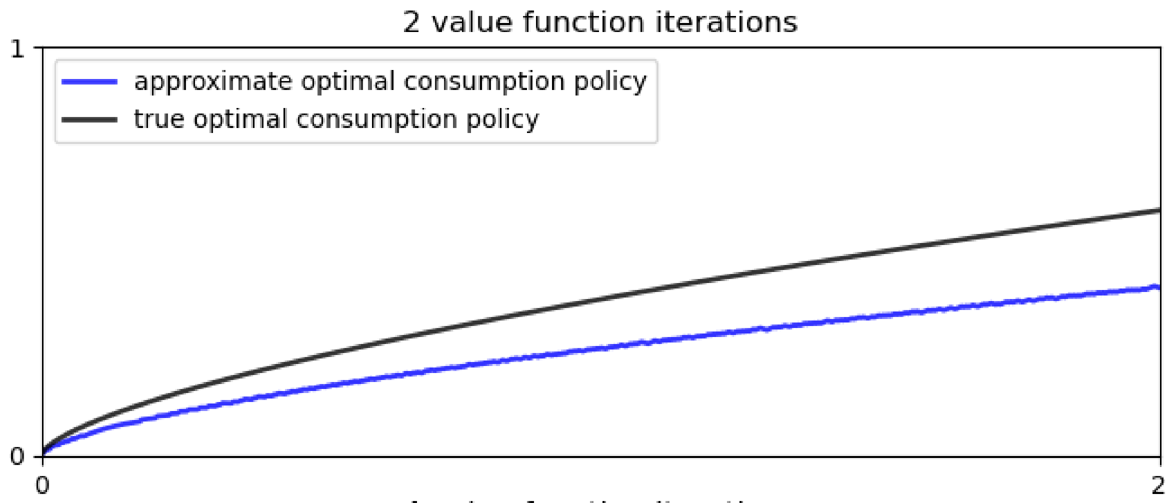


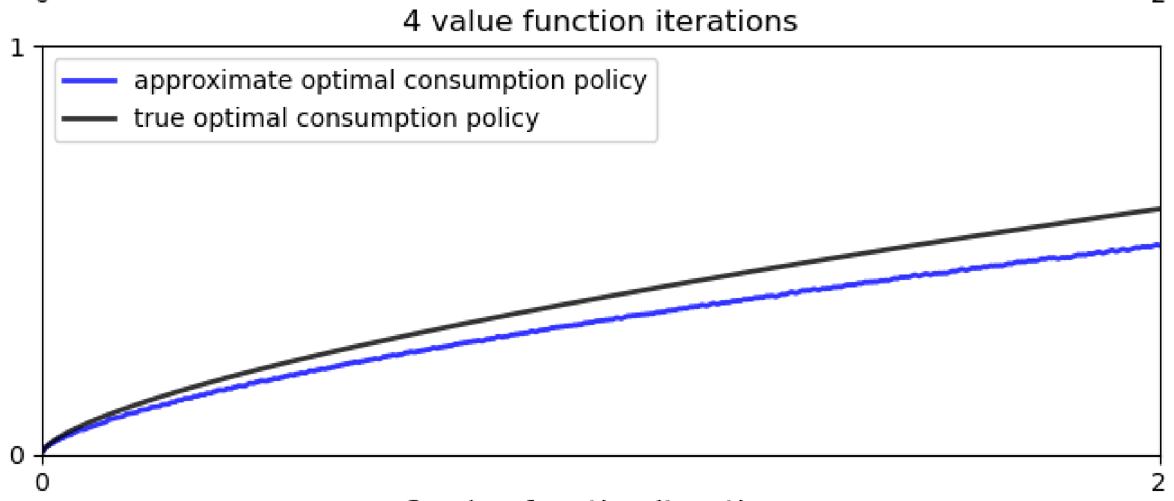
We next plot the consumption policies along with the value iteration

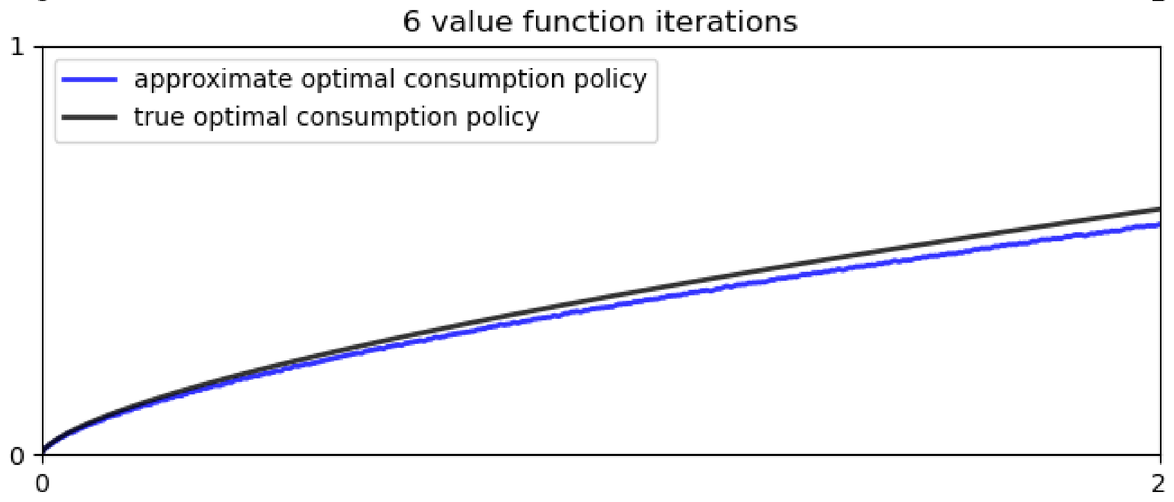
```
w = 5 * u(grid) - 25           # Initial condition

fig, ax = plt.subplots(3, 1, figsize=(8, 10))
true_c = c_star(grid)

for i, n in enumerate((2, 4, 6)):
    ax[i].set_ylim(0, 1)
    ax[i].set_xlim(0, 2)
    ax[i].set_yticks((0, 1))
    ax[i].set_xticks((0, 2))

    w = 5 * u(grid) - 25       # Initial condition
    compute_fixed_point(ddp.bellman_operator, w, max_iter=n, print_skip=1)
    σ = ddp.compute_greedy(w)  # Policy indices
    c_policy = f(grid) - grid[σ]

    ax[i].plot(grid, c_policy, 'b-', lw=2, alpha=0.8,
               label='approximate optimal consumption policy')
    ax[i].plot(grid, true_c, 'k-', lw=2, alpha=0.8,
               label='true optimal consumption policy')
    ax[i].legend(loc='upper left')
    ax[i].set_title(f'{n} value function iterations')
plt.show()
```

```
Iteration    Distance       Elapsed (seconds)
---------------------------------------------
1            5.518e+00      5.000e-04
2            4.070e+00      8.597e-04
Iteration    Distance       Elapsed (seconds)
---------------------------------------------
1            5.518e+00      3.660e-04
2            4.070e+00      7.148e-04
3            3.866e+00      1.073e-03
4            3.673e+00      1.412e-03
Iteration    Distance       Elapsed (seconds)
---------------------------------------------
1            5.518e+00      3.633e-04
2            4.070e+00      7.191e-04
3            3.866e+00      1.063e-03
4            3.673e+00      1.401e-03
5            3.489e+00      1.739e-03
6            3.315e+00      2.074e-03
```

```
/home/runner/miniconda3/envs/quantecon/lib/python3.11/site-packages/quantecon/_
↪compute_fp.py:152: RuntimeWarning: max_iter attained before convergence in␣
↪compute_fixed_point
  warnings.warn(_non_convergence_msg, RuntimeWarning)
```


2 value function iterations
approximate optimal consumption policy
true optimal consumption policy
1
0
0
2

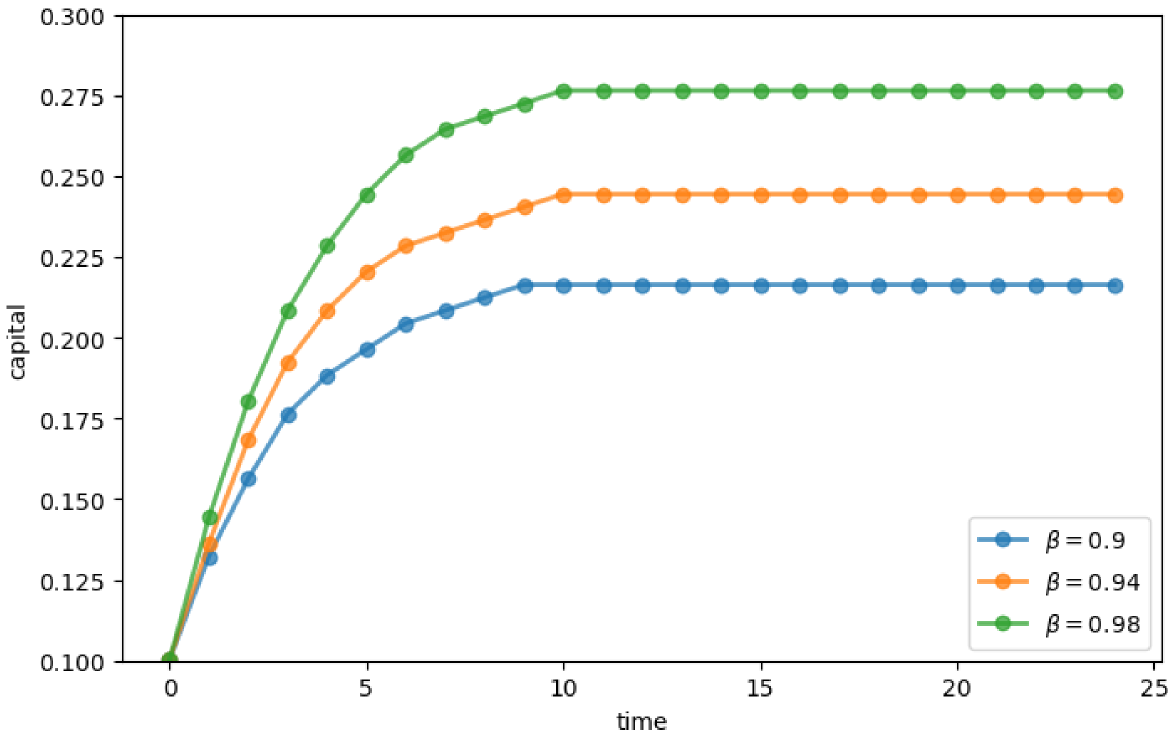

4 value function iterations
approximate optimal consumption policy
true optimal consumption policy
1
0
0
2


6 value function iterations
approximate optimal consumption policy
true optimal consumption policy
1
0
0
2

Dynamics of the Capital Stock

Finally, let us work on Exercise 2, where we plot the trajectories of the capital stock for three different discount factors, 0.9, 0.94, and 0.98, with initial condition $k_0 = 0.1$.

```
discount_factors = (0.9, 0.94, 0.98)
k_init = 0.1

# Search for the index corresponding to k_init
k_init_ind = np.searchsorted(grid, k_init)

sample_size = 25

fig, ax = plt.subplots(figsize=(8,5))
ax.set_xlabel("time")
ax.set_ylabel("capital")
ax.set_ylim(0.10, 0.30)

# Create a new instance, not to modify the one used above
ddp0 = DiscreteDP(R, Q, β, s_indices, a_indices)

for beta in discount_factors:
    ddp0.beta = beta
    res0 = ddp0.solve()
    k_path_ind = res0.mc.simulate(init=k_init_ind, ts_length=sample_size)
    k_path = grid[k_path_ind]
    ax.plot(k_path, 'o-', lw=2, alpha=0.75, label=f'$\\beta = {beta}$')

ax.legend(loc='lower right')
plt.show()
```



11.7 Appendix: Algorithms

This appendix covers the details of the solution algorithms implemented for `DiscreteDP`.

We will make use of the following notions of approximate optimality:

- For $\varepsilon > 0$, v is called an ε-approximation of v^* if $\|v - v^*\| < \varepsilon$.
- A policy $\sigma \in \Sigma$ is called ε-optimal if v_σ is an ε-approximation of v^*.

11.7.1 Value Iteration

The `DiscreteDP` value iteration method implements value function iteration as follows

1. Choose any $v^0 \in \mathbb{R}^n$, and specify $\varepsilon > 0$; set $i = 0$.
2. Compute $v^{i+1} = Tv^i$.
3. If $\|v^{i+1} - v^i\| < [(1 - \beta)/(2\beta)]\varepsilon$, then go to step 4; otherwise, set $i = i + 1$ and go to step 2.
4. Compute a v^{i+1}-greedy policy σ, and return v^{i+1} and σ.

Given $\varepsilon > 0$, the value iteration algorithm

- terminates in a finite number of iterations
- returns an $\varepsilon/2$-approximation of the optimal value function and an ε-optimal policy function (unless `iter_max` is reached)

(While not explicit, in the actual implementation each algorithm is terminated if the number of iterations reaches `iter_max`)

11.7.2 Policy Iteration

The `DiscreteDP` policy iteration method runs as follows

1. Choose any $v^0 \in \mathbb{R}^n$ and compute a v^0-greedy policy σ^0; set $i = 0$.
2. Compute the value v_{σ^i} by solving the equation $v = T_{\sigma^i} v$.
3. Compute a v_{σ^i}-greedy policy σ^{i+1}; let $\sigma^{i+1} = \sigma^i$ if possible.
4. If $\sigma^{i+1} = \sigma^i$, then return v_{σ^i} and σ^{i+1}; otherwise, set $i = i + 1$ and go to step 2.

The policy iteration algorithm terminates in a finite number of iterations.

It returns an optimal value function and an optimal policy function (unless `iter_max` is reached).

11.7.3 Modified Policy Iteration

The `DiscreteDP` modified policy iteration method runs as follows:

1. Choose any $v^0 \in \mathbb{R}^n$, and specify $\varepsilon > 0$ and $k \geq 0$; set $i = 0$.
2. Compute a v^i-greedy policy σ^{i+1}; let $\sigma^{i+1} = \sigma^i$ if possible (for $i \geq 1$).
3. Compute $u = Tv^i$ $(= T_{\sigma^{i+1}} v^i)$. If $\mathrm{span}(u - v^i) < [(1 - \beta)/\beta]\varepsilon$, then go to step 5; otherwise go to step 4.
    - Span is defined by $\mathrm{span}(z) = \max(z) - \min(z)$.
4. Compute $v^{i+1} = (T_{\sigma^{i+1}})^k u$ $(= (T_{\sigma^{i+1}})^{k+1} v^i)$; set $i = i + 1$ and go to step 2.

5. Return $v = u + [\beta/(1-\beta)][(\min(u - v^i) + \max(u - v^i))/2]\mathbf{1}$ and σ_{i+1}.

Given $\varepsilon > 0$, provided that v^0 is such that $Tv^0 \geq v^0$, the modified policy iteration algorithm terminates in a finite number of iterations.

It returns an $\varepsilon/2$-approximation of the optimal value function and an ε-optimal policy function (unless `iter_max` is reached).

See also the documentation for `DiscreteDP`.

Part IV

Time Series Models

CHAPTER

TWELVE

VARS AND DMDS

This lecture applies computational methods that we learned about in this lecture *Singular Value Decomposition* to

- first-order vector autoregressions (VARs)
- dynamic mode decompositions (DMDs)
- connections between DMDs and first-order VARs

12.1 First-Order Vector Autoregressions

We want to fit a **first-order vector autoregression**

$$X_{t+1} = AX_t + C\epsilon_{t+1}, \quad \epsilon_{t+1} \perp X_t \tag{12.1}$$

where ϵ_{t+1} is the time $t+1$ component of a sequence of i.i.d. $m \times 1$ random vectors with mean vector zero and identity covariance matrix and where the $m \times 1$ vector X_t is

$$X_t = \begin{bmatrix} X_{1,t} & X_{2,t} & \cdots & X_{m,t} \end{bmatrix}^\top \tag{12.2}$$

and where $\cdot^\top$ again denotes complex transposition and $X_{i,t}$ is variable i at time t.

We want to fit equation (12.1).

Our data are organized in an $m \times (n+1)$ matrix $\tilde{X}$

$$\tilde{X} = [X_1 \mid X_2 \mid \cdots \mid X_n \mid X_{n+1}]$$

where for $t = 1, \ldots, n+1$, the $m \times 1$ vector X_t is given by (12.2).

Thus, we want to estimate a system (12.1) that consists of m least squares regressions of **everything** on one lagged value of **everything**.

The i'th equation of (12.1) is a regression of $X_{i,t+1}$ on the vector X_t.

We proceed as follows.

From $\tilde{X}$, we form two $m \times n$ matrices

$$X = [X_1 \mid X_2 \mid \cdots \mid X_n]$$

and

$$X' = [X_2 \mid X_3 \mid \cdots \mid X_{n+1}]$$

Here $'$ is part of the name of the matrix X' and does not indicate matrix transposition.

We use $\cdot^\top$ to denote matrix transposition or its extension to complex matrices.

In forming X and X', we have in each case dropped a column from $\tilde{X}$, the last column in the case of X, and the first column in the case of X'.

Evidently, X and X' are both $m \times n$ matrices.

We denote the rank of X as $p \leq \min(m, n)$.

Two cases that interest us are

- $n >> m$, so that we have many more time series observations n than variables m
- $m >> n$, so that we have many more variables m than time series observations n

At a general level that includes both of these special cases, a common formula describes the least squares estimator $\hat{A}$ of A.

But important details differ.

The common formula is

$$\hat{A} = X' X^+ \tag{12.3}$$

where X^+ is the pseudo-inverse of X.

To read about the **Moore-Penrose pseudo-inverse** please see Moore-Penrose pseudo-inverse

Applicable formulas for the pseudo-inverse differ for our two cases.

Short-Fat Case:

When $n >> m$, so that we have many more time series observations n than variables m and when X has linearly independent **rows**, $XX^\top$ has an inverse and the pseudo-inverse X^+ is

$$X^+ = X^\top (XX^\top)^{-1}$$

Here X^+ is a **right-inverse** that verifies $XX^+ = I_{m \times m}$.

In this case, our formula (12.3) for the least-squares estimator of the population matrix of regression coefficients A becomes

$$\hat{A} = X' X^\top (XX^\top)^{-1} \tag{12.4}$$

This formula for least-squares regression coefficients is widely used in econometrics.

It is used to estimate vector autorgressions.

The right side of formula (12.4) is proportional to the empirical cross second moment matrix of X_{t+1} and X_t times the inverse of the second moment matrix of X_t.

Tall-Skinny Case:

When $m >> n$, so that we have many more attributes m than time series observations n and when X has linearly independent **columns**, $X^\top X$ has an inverse and the pseudo-inverse X^+ is

$$X^+ = (X^\top X)^{-1} X^\top$$

Here X^+ is a **left-inverse** that verifies $X^+ X = I_{n \times n}$.

In this case, our formula (12.3) for a least-squares estimator of A becomes

$$\hat{A} = X' (X^\top X)^{-1} X^\top \tag{12.5}$$

Please compare formulas (12.4) and (12.5) for $\hat{A}$.

Here we are especially interested in formula (12.5).

The ith row of $\hat{A}$ is an $m \times 1$ vector of regression coefficients of $X_{i,t+1}$ on $X_{j,t}, j = 1, \dots, m$.

If we use formula (12.5) to calculate $\hat{A}X$ we find that

$$\hat{A}X = X'$$

so that the regression equation **fits perfectly**.

This is a typical outcome in an **underdetermined least-squares** model.

To reiterate, in the **tall-skinny** case (described in *Singular Value Decomposition*) in which we have a number n of observations that is small relative to the number m of attributes that appear in the vector X_t, we want to fit equation (12.1).

We confront the facts that the least squares estimator is underdetermined and that the regression equation fits perfectly.

To proceed, we'll want efficiently to calculate the pseudo-inverse X^+.

The pseudo-inverse X^+ will be a component of our estimator of A.

As our estimator $\hat{A}$ of A we want to form an $m \times m$ matrix that solves the least-squares best-fit problem

$$\hat{A} = \text{argmin}_{\check{A}} ||X' - \check{A}X||_F \tag{12.6}$$

where $|| \cdot ||_F$ denotes the Frobenius (or Euclidean) norm of a matrix.

The Frobenius norm is defined as

$$||A||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{m} |A_{ij}|^2}$$

The minimizer of the right side of equation (12.6) is

$$\hat{A} = X'X^+ \tag{12.7}$$

where the (possibly huge) $n \times m$ matrix $X^+ = (X^\top X)^{-1} X^\top$ is again a pseudo-inverse of X.

For some situations that we are interested in, $X^\top X$ can be close to singular, a situation that makes some numerical algorithms be inaccurate.

To acknowledge that possibility, we'll use efficient algorithms to constructing a **reduced-rank approximation** of $\hat{A}$ in formula (12.5).

Such an approximation to our vector autoregression will no longer fit perfectly.

The ith row of $\hat{A}$ is an $m \times 1$ vector of regression coefficients of $X_{i,t+1}$ on $X_{j,t}, j = 1, \dots, m$.

An efficient way to compute the pseudo-inverse X^+ is to start with a singular value decomposition

$$X = U \Sigma V^\top \tag{12.8}$$

where we remind ourselves that for a **reduced** SVD, X is an $m \times n$ matrix of data, U is an $m \times p$ matrix, Σ is a $p \times p$ matrix, and V is an $n \times p$ matrix.

We can efficiently construct the pertinent pseudo-inverse X^+ by recognizing the following string of equalities.

$$\begin{aligned} X^+ &= (X^\top X)^{-1} X^\top \\ &= (V \Sigma U^\top U \Sigma V^\top)^{-1} V \Sigma U^\top \\ &= (V \Sigma \Sigma V^\top)^{-1} V \Sigma U^\top \\ &= V \Sigma^{-1} \Sigma^{-1} V^\top V \Sigma U^\top \\ &= V \Sigma^{-1} U^\top \end{aligned} \tag{12.9}$$

(Since we are in the $m >> n$ case in which $V^\top V = I_{p\times p}$ in a reduced SVD, we can use the preceding string of equalities for a reduced SVD as well as for a full SVD.)

Thus, we shall construct a pseudo-inverse X^+ of X by using a singular value decomposition of X in equation (12.8) to compute

$$X^+ = V\Sigma^{-1}U^\top \tag{12.10}$$

where the matrix Σ^{-1} is constructed by replacing each non-zero element of Σ with σ_j^{-1}.

We can use formula (12.10) together with formula (12.7) to compute the matrix $\hat{A}$ of regression coefficients.

Thus, our estimator $\hat{A} = X'X^+$ of the $m \times m$ matrix of coefficients A is

$$\hat{A} = X'V\Sigma^{-1}U^\top \tag{12.11}$$

12.2 Dynamic Mode Decomposition (DMD)

We turn to the $m >> n$ **tall and skinny** case associated with **Dynamic Mode Decomposition**.

Here an $m \times n + 1$ data matrix $\tilde{X}$ contains many more attributes (or variables) m than time periods $n + 1$.

Dynamic mode decomposition was introduced by [Schmid, 2010],

You can read about Dynamic Mode Decomposition here [Kutz *et al.*, 2016] and here [Brunton and Kutz, 2019] (section 7.2).

Dynamic Mode Decomposition (DMD) computes a rank $r < p$ approximation to the least squares regression coefficients $\hat{A}$ described by formula (12.11).

We'll build up gradually to a formulation that is useful in applications.

We'll do this by describing three alternative representations of our first-order linear dynamic system, i.e., our vector autoregression.

Guide to three representations: In practice, we'll mainly be interested in Representation 3.

We use the first two representations to present some useful intermediate steps that help us to appreciate what is under the hood of Representation 3.

In applications, we'll use only a small subset of **DMD modes** to approximate dynamics.

We use such a small subset of DMD modes to construct a reduced-rank approximation to A.

To do that, we'll want to use the **reduced** SVD's affiliated with representation 3, not the **full** SVD's affiliated with representations 1 and 2.

Guide to impatient reader: In our applications, we'll be using Representation 3.

You might want to skip the stage-setting representations 1 and 2 on first reading.

12.3 Representation 1

In this representation, we shall use a **full** SVD of X.

We use the m **columns** of U, and thus the m **rows** of $U^\top$, to define a $m \times 1$ vector $\tilde{b}_t$ as

$$\tilde{b}_t = U^\top X_t. \tag{12.12}$$

The original data X_t can be represented as

$$X_t = U\tilde{b}_t \tag{12.13}$$

(Here we use b to remind ourselves that we are creating a **basis** vector.)

Since we are now using a **full** SVD, $UU^\top = I_{m\times m}$.

So it follows from equation (12.12) that we can reconstruct X_t from $\tilde{b}_t$.

In particular,

- Equation (12.12) serves as an **encoder** that **rotates** the $m \times 1$ vector X_t to become an $m \times 1$ vector $\tilde{b}_t$
- Equation (12.13) serves as a **decoder** that **reconstructs** the $m \times 1$ vector X_t by rotating the $m \times 1$ vector $\tilde{b}_t$

Define a transition matrix for an $m \times 1$ basis vector $\tilde{b}_t$ by

$$\tilde{A} = U^\top \hat{A} U \tag{12.14}$$

We can recover $\hat{A}$ from

$$\hat{A} = U\tilde{A}U^\top$$

Dynamics of the $m \times 1$ basis vector $\tilde{b}_t$ are governed by

$$\tilde{b}_{t+1} = \tilde{A}\tilde{b}_t$$

To construct forecasts $\overline{X}_t$ of future values of X_t conditional on X_1, we can apply decoders (i.e., rotators) to both sides of this equation and deduce

$$\overline{X}_{t+1} = U\tilde{A}^t U^\top X_1$$

where we use $\overline{X}_{t+1}, t \geq 1$ to denote a forecast.

12.4 Representation 2

This representation is related to one originally proposed by [Schmid, 2010].

It can be regarded as an intermediate step on the way to obtaining a related representation 3 to be presented later

As with Representation 1, we continue to

- use a **full** SVD and **not** a reduced SVD

As we observed and illustrated in a lecture about the *Singular Value Decomposition*

- (a) for a full SVD $UU^\top = I_{m\times m}$ and $U^\top U = I_{p\times p}$ are both identity matrices
- (b) for a reduced SVD of X, $U^\top U$ is not an identity matrix.

As we shall see later, a full SVD is too confining for what we ultimately want to do, namely, cope with situations in which $U^\top U$ is **not** an identity matrix because we use a reduced SVD of X.

But for now, let's proceed under the assumption that we are using a full SVD so that requirements (a) and (b) are both satisfied.

Form an eigendecomposition of the $m \times m$ matrix $\tilde{A} = U^\top \hat{A} U$ defined in equation (12.14):

$$\tilde{A} = W \Lambda W^{-1} \tag{12.15}$$

where Λ is a diagonal matrix of eigenvalues and W is an $m \times m$ matrix whose columns are eigenvectors corresponding to rows (eigenvalues) in Λ.

When $UU^\top = I_{m \times m}$, as is true with a full SVD of X, it follows that

$$\hat{A} = U \tilde{A} U^\top = U W \Lambda W^{-1} U^\top \tag{12.16}$$

According to equation (12.16), the diagonal matrix Λ contains eigenvalues of $\hat{A}$ and corresponding eigenvectors of $\hat{A}$ are columns of the matrix UW.

It follows that the systematic (i.e., not random) parts of the X_t dynamics captured by our first-order vector autoregressions are described by

$$X_{t+1} = U W \Lambda W^{-1} U^\top X_t$$

Multiplying both sides of the above equation by $W^{-1} U^\top$ gives

$$W^{-1} U^\top X_{t+1} = \Lambda W^{-1} U^\top X_t$$

or

$$\hat{b}_{t+1} = \Lambda \hat{b}_t$$

where our **encoder** is

$$\hat{b}_t = W^{-1} U^\top X_t$$

and our **decoder** is

$$X_t = U W \hat{b}_t$$

We can use this representation to construct a predictor $\overline{X}_{t+1}$ of X_{t+1} conditional on X_1 via:

$$\overline{X}_{t+1} = U W \Lambda^t W^{-1} U^\top X_1 \tag{12.17}$$

In effect, [Schmid, 2010] defined an $m \times m$ matrix Φ_s as

$$\Phi_s = UW \tag{12.18}$$

and a generalized inverse

$$\Phi_s^+ = W^{-1} U^\top \tag{12.19}$$

[Schmid, 2010] then represented equation (12.17) as

$$\overline{X}_{t+1} = \Phi_s \Lambda^t \Phi_s^+ X_1 \tag{12.20}$$

Components of the basis vector $\hat{b}_t = W^{-1} U^\top X_t \equiv \Phi_s^+ X_t$ are
DMD **projected modes**.

To understand why they are called **projected modes**, notice that

$$\Phi_s^+ = (\Phi_s^\top \Phi_s)^{-1}\Phi_s^\top$$

so that the $m \times p$ matrix

$$\hat{b} = \Phi_s^+ X$$

is a matrix of regression coefficients of the $m \times n$ matrix X on the $m \times p$ matrix Φ_s.

We'll say more about this interpretation in a related context when we discuss representation 3, which was suggested by Tu et al. [Tu *et al.*, 2014].

It is more appropriate to use representation 3 when, as is often the case in practice, we want to use a reduced SVD.

12.5 Representation 3

Departing from the procedures used to construct Representations 1 and 2, each of which deployed a **full** SVD, we now use a **reduced** SVD.

Again, we let $p \leq \min(m, n)$ be the rank of X.

Construct a **reduced** SVD

$$X = \tilde{U}\tilde{\Sigma}\tilde{V}^\top,$$

where now $\tilde{U}$ is $m \times p$, $\tilde{\Sigma}$ is $p \times p$, and $\tilde{V}^\top$ is $p \times n$.

Our minimum-norm least-squares approximator of A now has representation

$$\hat{A} = X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^\top \tag{12.21}$$

Computing Dominant Eigenvectors of $\hat{A}$

We begin by paralleling a step used to construct Representation 1, define a transition matrix for a rotated $p \times 1$ state $\tilde{b}_t$ by

$$\tilde{A} = \tilde{U}^\top \hat{A}\tilde{U} \tag{12.22}$$

Interpretation as projection coefficients

[Brunton and Kutz, 2022] remark that $\tilde{A}$ can be interpreted in terms of a projection of $\hat{A}$ onto the p modes in $\tilde{U}$.

To verify this, first note that, because $\tilde{U}^\top\tilde{U} = I$, it follows that

$$\tilde{A} = \tilde{U}^\top \hat{A}\tilde{U} = \tilde{U}^\top X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^\top\tilde{U} = \tilde{U}^\top X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^\top \tag{12.23}$$

Next, we'll just compute the regression coefficients in a projection of $\hat{A}$ on $\tilde{U}$ using a standard least-squares formula

$$(\tilde{U}^\top\tilde{U})^{-1}\tilde{U}^\top\hat{A} = (\tilde{U}^\top\tilde{U})^{-1}\tilde{U}^\top X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^\top = \tilde{U}^\top X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^\top = \tilde{A}.$$

Thus, we have verified that $\tilde{A}$ is a least-squares projection of $\hat{A}$ onto $\tilde{U}$.

An Inverse Challenge

Because we are using a reduced SVD, $\tilde{U}\tilde{U}^\top \neq I$.

Consequently,

$$\hat{A} \neq \tilde{U}\tilde{A}\tilde{U}^\top,$$

so we can't simply recover $\hat{A}$ from $\tilde{A}$ and $\tilde{U}$.

A Blind Alley

We can start by hoping for the best and proceeding to construct an eigendecomposition of the $p \times p$ matrix $\tilde{A}$:

$$\tilde{A} = \tilde{W}\Lambda\tilde{W}^{-1} \tag{12.24}$$

where Λ is a diagonal matrix of p eigenvalues and the columns of $\tilde{W}$ are corresponding eigenvectors.

Mimicking our procedure in Representation 2, we cross our fingers and compute an $m \times p$ matrix

$$\tilde{\Phi}_s = \tilde{U}\tilde{W} \tag{12.25}$$

that corresponds to (12.18) for a full SVD.

At this point, where $\hat{A}$ is given by formula (12.21) it is interesting to compute $\hat{A}\tilde{\Phi}_s$:

$$\begin{aligned}\hat{A}\tilde{\Phi}_s &= (X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^\top)(\tilde{U}\tilde{W}) \\ &= X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{W} \\ &\neq (\tilde{U}\tilde{W})\Lambda \\ &= \tilde{\Phi}_s\Lambda\end{aligned}$$

That $\hat{A}\tilde{\Phi}_s \neq \tilde{\Phi}_s\Lambda$ means that, unlike the corresponding situation in Representation 2, columns of $\tilde{\Phi}_s = \tilde{U}\tilde{W}$ are **not** eigenvectors of $\hat{A}$ corresponding to eigenvalues on the diagonal of matix Λ.

An Approach That Works

Continuing our quest for eigenvectors of $\hat{A}$ that we **can** compute with a reduced SVD, let's define an $m \times p$ matrix Φ as

$$\Phi \equiv \hat{A}\tilde{\Phi}_s = X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{W} \tag{12.26}$$

It turns out that columns of Φ **are** eigenvectors of $\hat{A}$.

This is a consequence of a result established by Tu et al. [Tu *et al.*, 2014] that we now present.

Proposition The p columns of Φ are eigenvectors of $\hat{A}$.

Proof: From formula (12.26) we have

$$\begin{aligned}\hat{A}\Phi &= (X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^\top)(X'\tilde{V}\Sigma^{-1}\tilde{W}) \\ &= X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{A}\tilde{W} \\ &= X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{W}\Lambda \\ &= \Phi\Lambda\end{aligned}$$

so that

$$\hat{A}\Phi = \Phi\Lambda. \tag{12.27}$$

Let ϕ_i be the ith column of Φ and λ_i be the corresponding i eigenvalue of $\tilde{A}$ from decomposition (12.24).

Equating the $m \times 1$ vectors that appear on the two sides of equation (12.27) gives

$$\hat{A}\phi_i = \lambda_i\phi_i.$$

This equation confirms that ϕ_i is an eigenvector of $\hat{A}$ that corresponds to eigenvalue λ_i of both $\tilde{A}$ and $\hat{A}$.

This concludes the proof.

Also see [Brunton and Kutz, 2022] (p. 238)

12.5.1 Decoder of $\check{b}$ as a linear projection

From eigendecomposition (12.27) we can represent $\hat{A}$ as

$$\hat{A} = \Phi \Lambda \Phi^{+}. \tag{12.28}$$

From formula (12.28) we can deduce dynamics of the $p \times 1$ vector $\check{b}_t$:

$$\check{b}_{t+1} = \Lambda \check{b}_t$$

where

$$\check{b}_t = \Phi^{+} X_t \tag{12.29}$$

Since the $m \times p$ matrix Φ has p linearly independent columns, the generalized inverse of Φ is

$$\Phi^{+} = (\Phi^\top \Phi)^{-1} \Phi^\top$$

and so

$$\check{b} = (\Phi^\top \Phi)^{-1} \Phi^\top X \tag{12.30}$$

The $p \times n$ matrix $\check{b}$ is recognizable as a matrix of least squares regression coefficients of the $m \times n$ matrix X on the $m \times p$ matrix Φ and consequently

$$\check{X} = \Phi \check{b} \tag{12.31}$$

is an $m \times n$ matrix of least squares projections of X on Φ.

Variance Decomposition of X

By virtue of the least-squares projection theory discussed in this quantecon lecture https://python-advanced.quantecon.org/orth_proj.html, we can represent X as the sum of the projection $\check{X}$ of X on Φ plus a matrix of errors.

To verify this, note that the least squares projection $\check{X}$ is related to X by

$$X = \check{X} + \epsilon$$

or

$$X = \Phi \check{b} + \epsilon \tag{12.32}$$

where ϵ is an $m \times n$ matrix of least squares errors satisfying the least squares orthogonality conditions $\epsilon^\top \Phi = 0$ or

$$(X - \Phi \check{b})^\top \Phi = 0_{m \times p} \tag{12.33}$$

Rearranging the orthogonality conditions (12.33) gives $X^\top \Phi = \check{b} \Phi^\top \Phi$, which implies formula (12.30).

12.5.2 An Approximation

We now describe a way to approximate the $p \times 1$ vector $\check{b}_t$ instead of using formula (12.29).

In particular, the following argument adapted from [Brunton and Kutz, 2022] (page 240) provides a computationally efficient way to approximate $\check{b}_t$.

For convenience, we'll apply the method at time $t = 1$.

For $t = 1$, from equation (12.32) we have

$$\check{X}_1 = \Phi \check{b}_1 \tag{12.34}$$

where $\check{b}_1$ is a $p \times 1$ vector.

Recall from representation 1 above that $X_1 = U\tilde{b}_1$, where $\tilde{b}_1$ is a time 1 basis vector for representation 1 and U is from the full SVD $X = U\Sigma V^\top$.

It then follows from equation (12.32) that

$$U\tilde{b}_1 = X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{W}\check{b}_1 + \epsilon_1$$

where ϵ_1 is a least-squares error vector from equation (12.32).

It follows that

$$\tilde{b}_1 = U^\top X'V\tilde{\Sigma}^{-1}\tilde{W}\check{b}_1 + U^\top \epsilon_1$$

Replacing the error term $U^\top \epsilon_1$ by zero, and replacing U from a **full** SVD of X with $\tilde{U}$ from a **reduced** SVD, we obtain an approximation $\hat{b}_1$ to $\tilde{b}_1$:

$$\hat{b}_1 = \tilde{U}^\top X'\tilde{V}\tilde{\Sigma}^{-1}\tilde{W}\check{b}_1$$

Recall that from equation (12.23), $\tilde{A} = \tilde{U}^\top X'\tilde{V}\tilde{\Sigma}^{-1}$.

It then follows that

$$\hat{b}_1 = \tilde{A}\tilde{W}\check{b}_1$$

and therefore, by the eigendecomposition (12.24) of $\tilde{A}$, we have

$$\hat{b}_1 = \tilde{W}\Lambda\check{b}_1$$

Consequently,

$$\hat{b}_1 = (\tilde{W}\Lambda)^{-1}\tilde{b}_1$$

or

$$\hat{b}_1 = (\tilde{W}\Lambda)^{-1}\tilde{U}^\top X_1, \tag{12.35}$$

which is a computationally efficient approximation to the following instance of equation (12.29) for the initial vector $\check{b}_1$:

$$\check{b}_1 = \Phi^+ X_1 \tag{12.36}$$

(To highlight that (12.35) is an approximation, users of DMD sometimes call components of basis vector $\check{b}_t = \Phi^+ X_t$ the **exact** DMD modes and components of $\hat{b}_t = (\tilde{W}\Lambda)^{-1}\tilde{U}^\top X_t$ the **approximate** modes.)

Conditional on X_t, we can compute a decoded $\check{X}_{t+j}, j = 1, 2, \ldots$ from the exact modes via

$$\check{X}_{t+j} = \Phi\Lambda^j\Phi^+ X_t \tag{12.37}$$

or use compute a decoded $\hat{X}_{t+j}$ from approximate modes via

$$\hat{X}_{t+j} = \Phi\Lambda^j(\tilde{W}\Lambda)^{-1}\tilde{U}^\top X_t. \tag{12.38}$$

We can then use a decoded $\check{X}_{t+j}$ or $\hat{X}_{t+j}$ to forecast X_{t+j}.

12.5.3 Using Fewer Modes

In applications, we'll actually use only a few modes, often three or less.

Some of the preceding formulas assume that we have retained all p modes associated with singular values of X.

We can adjust our formulas to describe a situation in which we instead retain only the $r < p$ largest singular values.

In that case, we simply replace $\tilde{\Sigma}$ with the appropriate $r \times r$ matrix of singular values, $\tilde{U}$ with the $m \times r$ matrix whose columns correspond to the r largest singular values, and $\tilde{V}$ with the $n \times r$ matrix whose columns correspond to the r largest singular values.

Counterparts of all of the salient formulas above then apply.

12.6 Source for Some Python Code

You can find a Python implementation of DMD here:

https://mathlab.sissa.it/pydmd

CHAPTER

THIRTEEN

FINITE MARKOV CHAINS

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install quantecon
```

13.1 Overview

Markov chains are one of the most useful classes of stochastic processes, being

- simple, flexible and supported by many elegant theoretical results
- valuable for building intuition about random dynamic models
- central to quantitative modeling in their own right

You will find them in many of the workhorse models of economics and finance.

In this lecture, we review some of the theory of Markov chains.

We will also introduce some of the high-quality routines for working with Markov chains available in QuantEcon.py.

Prerequisite knowledge is basic probability and linear algebra.

Let's start with some standard imports:

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
import quantecon as qe
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
```

13.2 Definitions

The following concepts are fundamental.

13.2.1 Stochastic Matrices

A **stochastic matrix** (or **Markov matrix**) is an $n \times n$ square matrix P such that

1. each element of P is nonnegative, and
2. each row of P sums to one

Each row of P can be regarded as a probability mass function over n possible outcomes.

It is too not difficult to check[1] that if P is a stochastic matrix, then so is the k-th power P^k for all $k \in \mathbb{N}$.

13.2.2 Markov Chains

There is a close connection between stochastic matrices and Markov chains.

To begin, let S be a finite set with n elements $\{x_1, \dots, x_n\}$.

The set S is called the **state space** and $x_1, \dots, x_n$ are the **state values**.

A **Markov chain** $\{X_t\}$ on S is a sequence of random variables on S that have the **Markov property**.

This means that, for any date t and any state $y \in S$,

$$\mathbb{P}\{X_{t+1} = y \,|\, X_t\} = \mathbb{P}\{X_{t+1} = y \,|\, X_t, X_{t-1}, \dots\} \tag{13.1}$$

In other words, knowing the current state is enough to know probabilities for future states.

In particular, the dynamics of a Markov chain are fully determined by the set of values

$$P(x, y) := \mathbb{P}\{X_{t+1} = y \,|\, X_t = x\} \qquad (x, y \in S) \tag{13.2}$$

By construction,

- $P(x, y)$ is the probability of going from x to y in one unit of time (one step)
- $P(x, \cdot)$ is the conditional distribution of X_{t+1} given $X_t = x$

We can view P as a stochastic matrix where

$$P_{ij} = P(x_i, x_j) \qquad 1 \leq i, j \leq n$$

Going the other way, if we take a stochastic matrix P, we can generate a Markov chain $\{X_t\}$ as follows:

- draw X_0 from a marginal distribution ψ
- for each $t = 0, 1, \dots$, draw X_{t+1} from $P(X_t, \cdot)$

By construction, the resulting process satisfies (13.2).

13.2.3 Example 1

Consider a worker who, at any given time t, is either unemployed (state 0) or employed (state 1).

Suppose that, over a one month period,

1. An unemployed worker finds a job with probability $\alpha \in (0, 1)$.
2. An employed worker loses her job and becomes unemployed with probability $\beta \in (0, 1)$.

[1] Hint: First show that if P and Q are stochastic matrices then so is their product — to check the row sums, try post multiplying by a column vector of ones. Finally, argue that P^n is a stochastic matrix using induction.

In terms of a Markov model, we have

- $S = \{0, 1\}$
- $P(0, 1) = \alpha$ and $P(1, 0) = \beta$

We can write out the transition probabilities in matrix form as

$$P = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix} \tag{13.3}$$

Once we have the values α and β, we can address a range of questions, such as

- What is the average duration of unemployment?
- Over the long-run, what fraction of time does a worker find herself unemployed?
- Conditional on employment, what is the probability of becoming unemployed at least once over the next 12 months?

We'll cover such applications below.

13.2.4 Example 2

From US unemployment data, Hamilton [Hamilton, 2005] estimated the stochastic matrix

$$P = \begin{pmatrix} 0.971 & 0.029 & 0 \\ 0.145 & 0.778 & 0.077 \\ 0 & 0.508 & 0.492 \end{pmatrix}$$

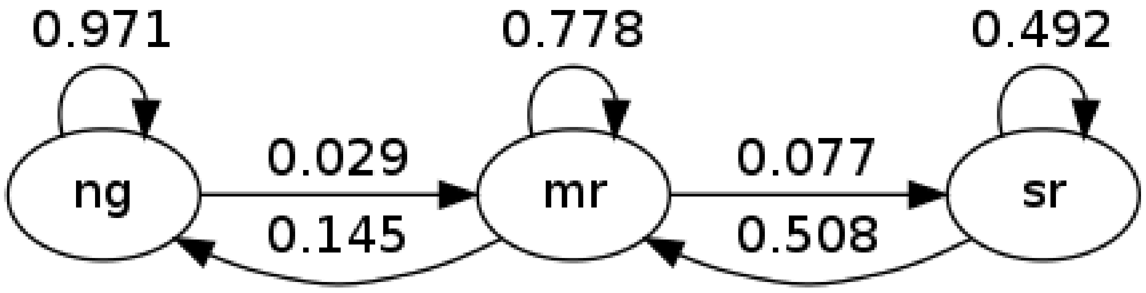
where

- the frequency is monthly
- the first state represents "normal growth"
- the second state represents "mild recession"
- the third state represents "severe recession"

For example, the matrix tells us that when the state is normal growth, the state will again be normal growth next month with probability 0.97.

In general, large values on the main diagonal indicate persistence in the process $\{X_t\}$.

This Markov process can also be represented as a directed graph, with edges labeled by transition probabilities



Here "ng" is normal growth, "mr" is mild recession, etc.

13.3 Simulation

One natural way to answer questions about Markov chains is to simulate them.

(To approximate the probability of event E, we can simulate many times and count the fraction of times that E occurs).

Nice functionality for simulating Markov chains exists in QuantEcon.py.

- Efficient, bundled with lots of other useful routines for handling Markov chains.

However, it's also a good exercise to roll our own routines — let's do that first and then come back to the methods in QuantEcon.py.

In these exercises, we'll take the state space to be $S = 0, \ldots, n - 1$.

13.3.1 Rolling Our Own

To simulate a Markov chain, we need its stochastic matrix P and a marginal probability distribution ψ from which to draw a realization of X_0.

The Markov chain is then constructed as discussed above. To repeat:

1. At time $t = 0$, draw a realization of X_0 from ψ.
2. At each subsequent time t, draw a realization of the new state X_{t+1} from $P(X_t, \cdot)$.

To implement this simulation procedure, we need a method for generating draws from a discrete distribution.

For this task, we'll use `random.draw` from QuantEcon, which works as follows:

```
ψ = (0.3, 0.7)           # probabilities over {0, 1}
cdf = np.cumsum(ψ)       # convert into cummulative distribution
qe.random.draw(cdf, 5)   # generate 5 independent draws from ψ
```

```
array([1, 1, 1, 1, 1])
```

We'll write our code as a function that accepts the following three arguments

- A stochastic matrix `P`
- An initial state `init`
- A positive integer `sample_size` representing the length of the time series the function should return

```
def mc_sample_path(P, ψ_0=None, sample_size=1_000):

    # set up
    P = np.asarray(P)
    X = np.empty(sample_size, dtype=int)

    # Convert each row of P into a cdf
    n = len(P)
    P_dist = [np.cumsum(P[i, :]) for i in range(n)]

    # draw initial state, defaulting to 0
    if ψ_0 is not None:
        X_0 = qe.random.draw(np.cumsum(ψ_0))
    else:
        X_0 = 0
```

(continues on next page)

(continued from previous page)

```
    # simulate
    X[0] = X_0
    for t in range(sample_size - 1):
        X[t+1] = qe.random.draw(P_dist[X[t]])

    return X
```

Let's see how it works using the small matrix

```
P = [[0.4, 0.6],
     [0.2, 0.8]]
```

As we'll see later, for a long series drawn from `P`, the fraction of the sample that takes value 0 will be about 0.25.

Moreover, this is true, regardless of the initial distribution from which X_0 is drawn.

The following code illustrates this

```
X = mc_sample_path(P, ψ_0=[0.1, 0.9], sample_size=100_000)
np.mean(X == 0)
```

```
0.25093
```

You can try changing the initial distribution to confirm that the output is always close to 0.25, at least for the `P` matrix above.

13.3.2 Using QuantEcon's Routines

As discussed above, QuantEcon.py has routines for handling Markov chains, including simulation.

Here's an illustration using the same P as the preceding example

```
from quantecon import MarkovChain

mc = qe.MarkovChain(P)
X = mc.simulate(ts_length=1_000_000)
np.mean(X == 0)
```

```
0.249151
```

The QuantEcon.py routine is JIT compiled and much faster.

```
%time mc_sample_path(P, sample_size=1_000_000) # Our homemade code version
```

```
CPU times: user 927 ms, sys: 462 µs, total: 927 ms
Wall time: 927 ms
```

```
array([0, 0, 0, ..., 1, 1, 0])
```

```
%time mc.simulate(ts_length=1_000_000) # qe code version
```

```
CPU times: user 14.3 ms, sys: 0 ns, total: 14.3 ms
Wall time: 14 ms
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

Adding State Values and Initial Conditions

If we wish to, we can provide a specification of state values to `MarkovChain`.

These state values can be integers, floats, or even strings.

The following code illustrates

```
mc = qe.MarkovChain(P, state_values=('unemployed', 'employed'))
mc.simulate(ts_length=4, init='employed')
```

```
array(['employed', 'employed', 'employed', 'employed'], dtype='<U10')
```

```
mc.simulate(ts_length=4, init='unemployed')
```

```
array(['unemployed', 'unemployed', 'unemployed', 'employed'], dtype='<U10')
```

```
mc.simulate(ts_length=4)  # Start at randomly chosen initial state
```

```
array(['unemployed', 'unemployed', 'employed', 'employed'], dtype='<U10')
```

If we want to see indices rather than state values as outputs as we can use

```
mc.simulate_indices(ts_length=4)
```

```
array([1, 1, 1, 0])
```

13.4 Marginal Distributions

Suppose that

1. $\{X_t\}$ is a Markov chain with stochastic matrix P
2. the marginal distribution of X_t is known to be ψ_t

What then is the marginal distribution of X_{t+1}, or, more generally, of X_{t+m}?

To answer this, we let ψ_t be the marginal distribution of X_t for $t = 0, 1, 2, \ldots$.

Our first aim is to find ψ_{t+1} given ψ_t and P.

To begin, pick any $y \in S$.

Using the law of total probability, we can decompose the probability that $X_{t+1} = y$ as follows:

$$\mathbb{P}\{X_{t+1} = y\} = \sum_{x \in S} \mathbb{P}\{X_{t+1} = y \,|\, X_t = x\} \cdot \mathbb{P}\{X_t = x\}$$

In words, to get the probability of being at y tomorrow, we account for all ways this can happen and sum their probabilities.

Rewriting this statement in terms of marginal and conditional probabilities gives

$$\psi_{t+1}(y) = \sum_{x \in S} P(x, y) \psi_t(x)$$

There are n such equations, one for each $y \in S$.

If we think of ψ_{t+1} and ψ_t as *row vectors*, these n equations are summarized by the matrix expression

$$\psi_{t+1} = \psi_t P \tag{13.4}$$

Thus, to move a marginal distribution forward one unit of time, we postmultiply by P.

By postmultiplying m times, we move a marginal distribution forward m steps into the future.

Hence, iterating on (13.4), the expression $\psi_{t+m} = \psi_t P^m$ is also valid — here P^m is the m-th power of P.

As a special case, we see that if ψ_0 is the initial distribution from which X_0 is drawn, then $\psi_0 P^m$ is the distribution of X_m.

This is very important, so let's repeat it

$$X_0 \sim \psi_0 \quad \implies \quad X_m \sim \psi_0 P^m \tag{13.5}$$

and, more generally,

$$X_t \sim \psi_t \quad \implies \quad X_{t+m} \sim \psi_t P^m \tag{13.6}$$

13.4.1 Multiple Step Transition Probabilities

We know that the probability of transitioning from x to y in one step is $P(x, y)$.

It turns out that the probability of transitioning from x to y in m steps is $P^m(x, y)$, the (x, y)-th element of the m-th power of P.

To see why, consider again (13.6), but now with a ψ_t that puts all probability on state x so that the transition probabilities are

- 1 in the x-th position and zero elsewhere

Inserting this into (13.6), we see that, conditional on $X_t = x$, the distribution of X_{t+m} is the x-th row of P^m.

In particular

$$\mathbb{P}\{X_{t+m} = y \,|\, X_t = x\} = P^m(x, y) = (x, y)\text{-th element of } P^m$$

13.4.2 Example: Probability of Recession

Recall the stochastic matrix P for recession and growth *considered above*.

Suppose that the current state is unknown — perhaps statistics are available only at the *end* of the current month.

We guess that the probability that the economy is in state x is $\psi(x)$.

The probability of being in recession (either mild or severe) in 6 months time is given by the inner product

$$\psi P^6 \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

13.4.3 Example 2: Cross-Sectional Distributions

The marginal distributions we have been studying can be viewed either as probabilities or as cross-sectional frequencies that a Law of Large Numbers leads us to anticipate for large samples.

To illustrate, recall our model of employment/unemployment dynamics for a given worker *discussed above*.

Consider a large population of workers, each of whose lifetime experience is described by the specified dynamics, with each worker's outcomes being realizations of processes that are statistically independent of all other workers' processes.

Let ψ be the current *cross-sectional* distribution over $\{0, 1\}$.

The cross-sectional distribution records fractions of workers employed and unemployed at a given moment.

- For example, $\psi(0)$ is the unemployment rate.

What will the cross-sectional distribution be in 10 periods hence?

The answer is ψP^{10}, where P is the stochastic matrix in (13.3).

This is because each worker's state evolves according to P, so ψP^{10} is a marginal distibution for a single randomly selected worker.

But when the sample is large, outcomes and probabilities are roughly equal (by an application of the Law of Large Numbers).

So for a very large (tending to infinite) population, ψP^{10} also represents fractions of workers in each state.

This is exactly the cross-sectional distribution.

13.5 Irreducibility and Aperiodicity

Irreducibility and aperiodicity are central concepts of modern Markov chain theory.

Let's see what they're about.

13.5.1 Irreducibility

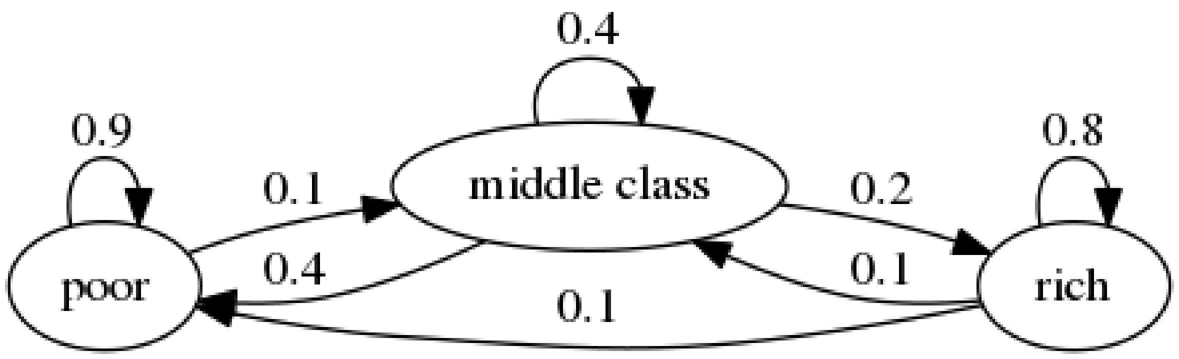
Let P be a fixed stochastic matrix.

Two states x and y are said to **communicate** with each other if there exist positive integers j and k such that

$$P^j(x, y) > 0 \quad \text{and} \quad P^k(y, x) > 0$$

In view of our discussion *above*, this means precisely that

- state x can eventually be reached from state y, and
- state y can eventually be reached from state x



The stochastic matrix P is called **irreducible** if all states communicate; that is, if x and y communicate for all (x, y) in $S \times S$.

For example, consider the following transition probabilities for wealth of a fictitious set of households

We can translate this into a stochastic matrix, putting zeros where there's no edge between nodes

$$P := \left(\begin{array}{ccc} 0.9 & 0.1 & 0 \\ 0.4 & 0.4 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{array} \right)$$

It's clear from the graph that this stochastic matrix is irreducible: we can eventually reach any state from any other state.

We can also test this using QuantEcon.py's MarkovChain class

```
P = [[0.9, 0.1, 0.0],
     [0.4, 0.4, 0.2],
     [0.1, 0.1, 0.8]]

mc = qe.MarkovChain(P, ('poor', 'middle', 'rich'))
mc.is_irreducible
```

```
True
```

Here's a more pessimistic scenario in which poor people remain poor forever

This stochastic matrix is not irreducible, since, for example, rich is not accessible from poor.

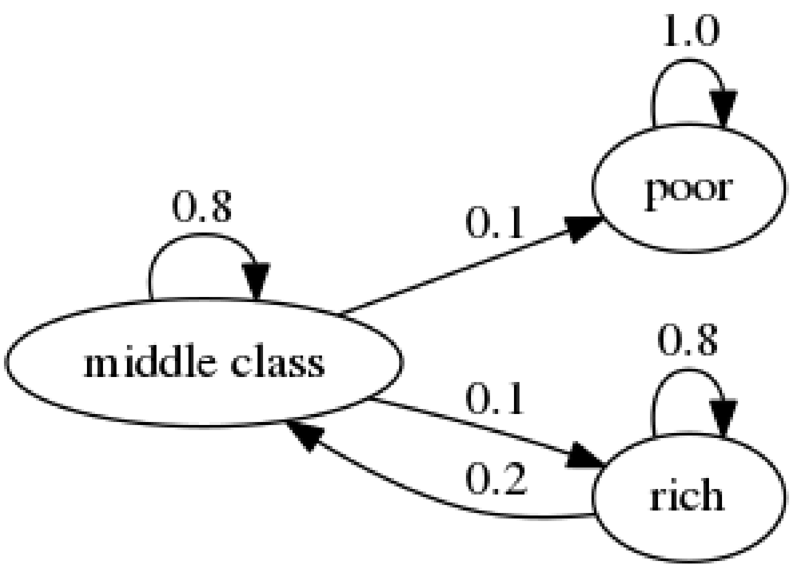
Let's confirm this

```
P = [[1.0, 0.0, 0.0],
     [0.1, 0.8, 0.1],
     [0.0, 0.2, 0.8]]

mc = qe.MarkovChain(P, ('poor', 'middle', 'rich'))
mc.is_irreducible
```

```
False
```

We can also determine the "communication classes"

```
mc.communication_classes
```



```
[array(['poor'], dtype='<U6'), array(['middle', 'rich'], dtype='<U6')]
```

It might be clear to you already that irreducibility is going to be important in terms of long run outcomes.

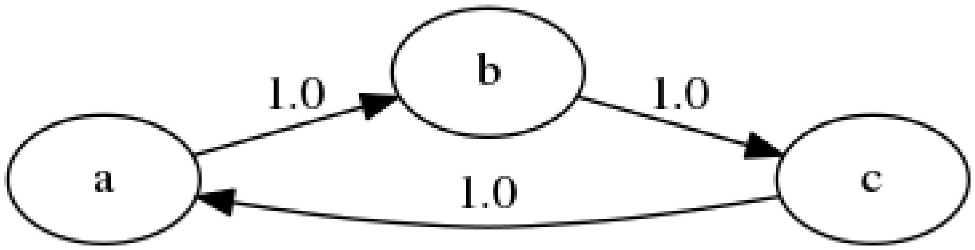
For example, poverty is a life sentence in the second graph but not the first.

We'll come back to this a bit later.

13.5.2 Aperiodicity

Loosely speaking, a Markov chain is called **periodic** if it cycles in a predictable way, and **aperiodic** otherwise.

Here's a trivial example with three states



The chain cycles with period 3:

```
P = [[0, 1, 0],
     [0, 0, 1],
     [1, 0, 0]]

mc = qe.MarkovChain(P)
mc.period
```

```
3
```

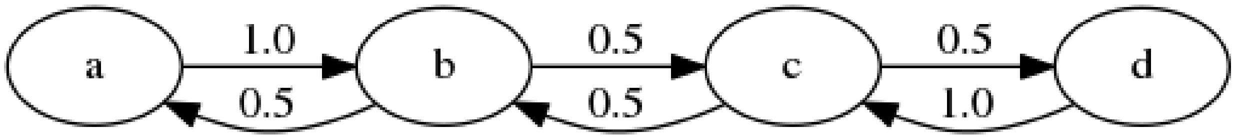
More formally, the **period** of a state x is the largest common divisor of a set of integers

$$D(x) := \{j \geq 1 : P^j(x, x) > 0\}$$

In the last example, $D(x) = \{3, 6, 9, ...\}$ for every state x, so the period is 3.

A stochastic matrix is called **aperiodic** if the period of every state is 1, and **periodic** otherwise.

For example, the stochastic matrix associated with the transition probabilities below is periodic because, for example, state a has period 2



We can confirm that the stochastic matrix is periodic with the following code

```
P = [[0.0, 1.0, 0.0, 0.0],
     [0.5, 0.0, 0.5, 0.0],
     [0.0, 0.5, 0.0, 0.5],
     [0.0, 0.0, 1.0, 0.0]]

mc = qe.MarkovChain(P)
mc.period
```

```
2
```

```
mc.is_aperiodic
```

```
False
```

13.6 Stationary Distributions

As seen in (13.4), we can shift a marginal distribution forward one unit of time via postmultiplication by P.

Some distributions are invariant under this updating process — for example,

```
P = np.array([[0.4, 0.6],
              [0.2, 0.8]])
ψ = (0.25, 0.75)
ψ @ P
```

```
array([0.25, 0.75])
```

Such distributions are called **stationary** or **invariant**.

Formally, a marginal distribution ψ^* on S is called **stationary** for P if $\psi^* = \psi^* P$.

(This is the same notion of stationarity that we learned about in the lecture on AR(1) processes applied to a different setting.)

From this equality, we immediately get $\psi^* = \psi^* P^t$ for all t.

This tells us an important fact: If the distribution of X_0 is a stationary distribution, then X_t will have this same distribution for all t.

Hence stationary distributions have a natural interpretation as **stochastic steady states** — we'll discuss this more soon.

Mathematically, a stationary distribution is a fixed point of P when P is thought of as the map $\psi \mapsto \psi P$ from (row) vectors to (row) vectors.

Theorem. Every stochastic matrix P has at least one stationary distribution.

(We are assuming here that the state space S is finite; if not more assumptions are required)

For proof of this result, you can apply Brouwer's fixed point theorem, or see EDTC, theorem 4.3.5.

There can be many stationary distributions corresponding to a given stochastic matrix P.

- For example, if P is the identity matrix, then all marginal distributions are stationary.

To get uniqueness an invariant distribution, the transition matrix P must have the property that no nontrivial subsets of the state space are **infinitely persistent**.

A subset of the state space is infinitely persistent if other parts of the state space cannot be accessed from it.

Thus, infinite persistence of a non-trivial subset is the opposite of irreducibility.

This gives some intuition for the following fundamental theorem.

Theorem. If P is both aperiodic and irreducible, then

1. P has exactly one stationary distribution ψ^*.
2. For any initial marginal distribution ψ_0, we have $\| \psi_0 P^t - \psi^* \| \to 0$ as $t \to \infty$.

For a proof, see, for example, theorem 5.2 of [Häggström, 2002].

(Note that part 1 of the theorem only requires irreducibility, whereas part 2 requires both irreducibility and aperiodicity)

A stochastic matrix that satisfies the conditions of the theorem is sometimes called **uniformly ergodic**.

A sufficient condition for aperiodicity and irreducibility is that every element of P is strictly positive.

- Try to convince yourself of this.

13.6.1 Example

Recall our model of the employment/unemployment dynamics of a particular worker *discussed above*.

Assuming $\alpha \in (0, 1)$ and $\beta \in (0, 1)$, the uniform ergodicity condition is satisfied.

Let $\psi^* = (p, 1 - p)$ be the stationary distribution, so that p corresponds to unemployment (state 0).

Using $\psi^* = \psi^* P$ and a bit of algebra yields

$$p = \frac{\beta}{\alpha + \beta}$$

This is, in some sense, a steady state probability of unemployment — more about the interpretation of this below.

Not surprisingly it tends to zero as $\beta \to 0$, and to one as $\alpha \to 0$.

13.6.2 Calculating Stationary Distributions

As discussed above, a particular Markov matrix P can have many stationary distributions.

That is, there can be many row vectors ψ such that $\psi = \psi P$.

In fact if P has two distinct stationary distributions ψ_1, ψ_2 then it has infinitely many, since in this case, as you can verify, for any $\lambda \in [0, 1]$

$$\psi_3 := \lambda \psi_1 + (1 - \lambda) \psi_2$$

is a stationary distribution for P.

If we restrict attention to the case in which only one stationary distribution exists, one way to finding it is to solve the system

$$\psi(I_n - P) = 0 \tag{13.7}$$

for ψ, where I_n is the $n \times n$ identity.

But the zero vector solves system (13.7), so we must proceed cautiously.

We want to impose the restriction that ψ is a probability distribution.

There are various ways to do this.

One option is to regard solving system (13.7) as an eigenvector problem: a vector ψ such that $\psi = \psi P$ is a left eigenvector associated with the unit eigenvalue $\lambda = 1$.

A stable and sophisticated algorithm specialized for stochastic matrices is implemented in QuantEcon.py.

This is the one we recommend:

```
P = [[0.4, 0.6],
     [0.2, 0.8]]

mc = qe.MarkovChain(P)
mc.stationary_distributions  # Show all stationary distributions
```

```
array([[0.25, 0.75]])
```

13.6.3 Convergence to Stationarity

Part 2 of the Markov chain convergence theorem *stated above* tells us that the marginal distribution of X_t converges to the stationary distribution regardless of where we begin.

This adds considerable authority to our interpretation of ψ^* as a stochastic steady state.

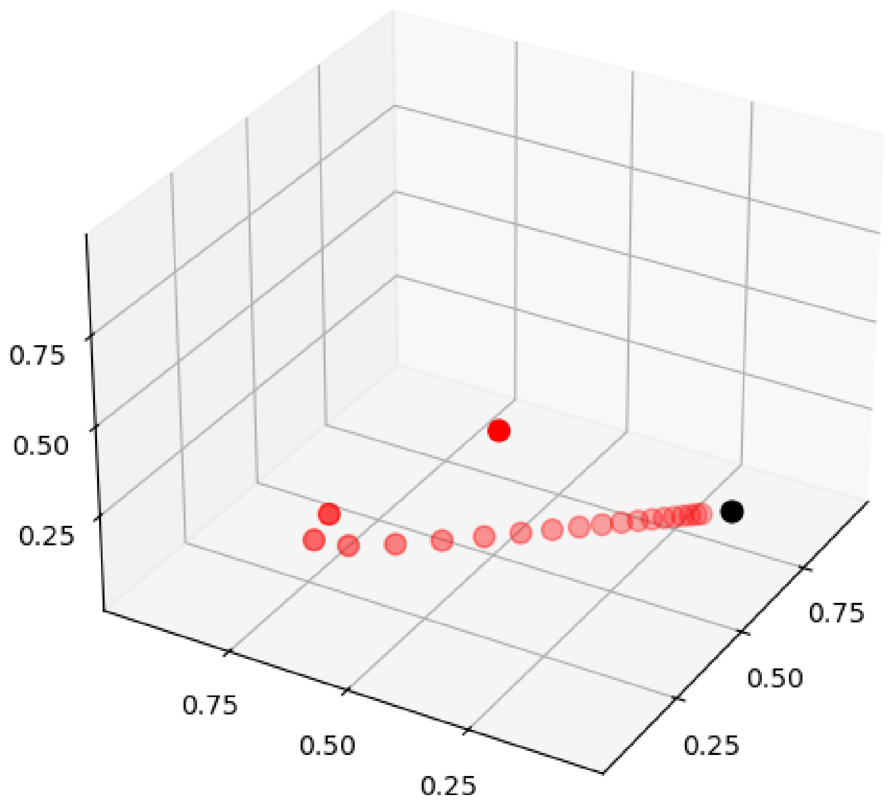
The convergence in the theorem is illustrated in the next figure

```
P = ((0.971, 0.029, 0.000),
     (0.145, 0.778, 0.077),
     (0.000, 0.508, 0.492))
P = np.array(P)

ψ = (0.0, 0.2, 0.8)        # Initial condition

fig = plt.figure(figsize=(8, 6))
```

(continues on next page)

(continued from previous page)

```
ax = fig.add_subplot(111, projection='3d')

ax.set(xlim=(0, 1), ylim=(0, 1), zlim=(0, 1),
       xticks=(0.25, 0.5, 0.75),
       yticks=(0.25, 0.5, 0.75),
       zticks=(0.25, 0.5, 0.75))

x_vals, y_vals, z_vals = [], [], []
for t in range(20):
    x_vals.append(ψ[0])
    y_vals.append(ψ[1])
    z_vals.append(ψ[2])
    ψ = ψ @ P

ax.scatter(x_vals, y_vals, z_vals, c='r', s=60)
ax.view_init(30, 210)

mc = qe.MarkovChain(P)
ψ_star = mc.stationary_distributions[0]
ax.scatter(ψ_star[0], ψ_star[1], ψ_star[2], c='k', s=60)

plt.show()
```



Here

- P is the stochastic matrix for recession and growth *considered above*.
- The highest red dot is an arbitrarily chosen initial marginal probability distribution ψ, represented as a vector in $\mathbb{R}^3$.
- The other red dots are the marginal distributions ψP^t for $t = 1, 2, \ldots$.
- The black dot is ψ^*.

You might like to try experimenting with different initial conditions.

13.7 Ergodicity

Under irreducibility, yet another important result obtains: for all $x \in S$,

$$\frac{1}{m} \sum_{t=1}^{m} \mathbf{1}\{X_t = x\} \to \psi^*(x) \quad \text{as } m \to \infty \tag{13.8}$$

Here

- $\mathbf{1}\{X_t = x\} = 1$ if $X_t = x$ and zero otherwise
- convergence is with probability one
- the result does not depend on the marginal distribution of X_0

The result tells us that the fraction of time the chain spends at state x converges to $\psi^*(x)$ as time goes to infinity.

This gives us another way to interpret the stationary distribution — provided that the convergence result in (13.8) is valid.

The convergence asserted in (13.8) is a special case of a law of large numbers result for Markov chains — see EDTC, section 4.3.4 for some additional information.

13.7.1 Example

Recall our cross-sectional interpretation of the employment/unemployment model *discussed above*.

Assume that $\alpha \in (0, 1)$ and $\beta \in (0, 1)$, so that irreducibility and aperiodicity both hold.

We saw that the stationary distribution is $(p, 1 - p)$, where

$$p = \frac{\beta}{\alpha + \beta}$$

In the cross-sectional interpretation, this is the fraction of people unemployed.

In view of our latest (ergodicity) result, it is also the fraction of time that a single worker can expect to spend unemployed.

Thus, in the long-run, cross-sectional averages for a population and time-series averages for a given person coincide.

This is one aspect of the concept of ergodicity.

13.8 Computing Expectations

We sometimes want to compute mathematical expectations of functions of X_t of the form

$$\mathbb{E}[h(X_t)] \tag{13.9}$$

and conditional expectations such as

$$\mathbb{E}[h(X_{t+k}) \mid X_t = x] \tag{13.10}$$

where

- $\{X_t\}$ is a Markov chain generated by $n \times n$ stochastic matrix P
- h is a given function, which, in terms of matrix algebra, we'll think of as the column vector

$$h = \begin{pmatrix} h(x_1) \\ \vdots \\ h(x_n) \end{pmatrix}$$

Computing the unconditional expectation (13.9) is easy.

We just sum over the marginal distribution of X_t to get

$$\mathbb{E}[h(X_t)] = \sum_{x \in S} (\psi P^t)(x) h(x)$$

Here ψ is the distribution of X_0.

Since ψ and hence ψP^t are row vectors, we can also write this as

$$\mathbb{E}[h(X_t)] = \psi P^t h$$

For the conditional expectation (13.10), we need to sum over the conditional distribution of X_{t+k} given $X_t = x$.

We already know that this is $P^k(x, \cdot)$, so

$$\mathbb{E}[h(X_{t+k}) \mid X_t = x] = (P^k h)(x) \tag{13.11}$$

The vector $P^k h$ stores the conditional expectation $\mathbb{E}[h(X_{t+k}) \mid X_t = x]$ over all x.

13.8.1 Iterated Expectations

The **law of iterated expectations** states that

$$\mathbb{E}\left[\mathbb{E}[h(X_{t+k}) \mid X_t = x]\right] = \mathbb{E}[h(X_{t+k})]$$

where the outer $\mathbb{E}$ on the left side is an unconditional distribution taken with respect to the marginal distribution ψ_t of X_t (again see equation (13.6)).

To verify the law of iterated expectations, use equation (13.11) to substitute $(P^k h)(x)$ for $E[h(X_{t+k}) \mid X_t = x]$, write

$$\mathbb{E}\left[\mathbb{E}[h(X_{t+k}) \mid X_t = x]\right] = \psi_t P^k h,$$

and note $\psi_t P^k h = \psi_{t+k} h = \mathbb{E}[h(X_{t+k})]$.

13.8.2 Expectations of Geometric Sums

Sometimes we want to compute the mathematical expectation of a geometric sum, such as $\sum_t \beta^t h(X_t)$.

In view of the preceding discussion, this is

$$\mathbb{E}[\sum_{j=0}^{\infty} \beta^j h(X_{t+j}) \mid X_t = x\Big] = [(I - \beta P)^{-1} h](x)$$

where

$$(I - \beta P)^{-1} = I + \beta P + \beta^2 P^2 + \cdots$$

Premultiplication by $(I - \beta P)^{-1}$ amounts to "applying the **resolvent operator**".

13.9 Exercises

Exercise 13.9.1

According to the discussion *above*, if a worker's employment dynamics obey the stochastic matrix

$$P = \begin{pmatrix} 1 - \alpha & \alpha \\ \beta & 1 - \beta \end{pmatrix}$$

with $\alpha \in (0, 1)$ and $\beta \in (0, 1)$, then, in the long-run, the fraction of time spent unemployed will be

$$p := \frac{\beta}{\alpha + \beta}$$

In other words, if $\{X_t\}$ represents the Markov chain for employment, then $\bar X_m \to p$ as $m \to \infty$, where

$$\bar X_m := \frac{1}{m} \sum_{t=1}^{m} \mathbf{1}\{X_t = 0\}$$

This exercise asks you to illustrate convergence by computing $\bar X_m$ for large m and checking that it is close to p.

You will see that this statement is true regardless of the choice of initial condition or the values of α, β, provided both lie in $(0, 1)$.

Solution to Exercise 13.9.1

We will address this exercise graphically.

The plots show the time series of $\bar X_m - p$ for two initial conditions.

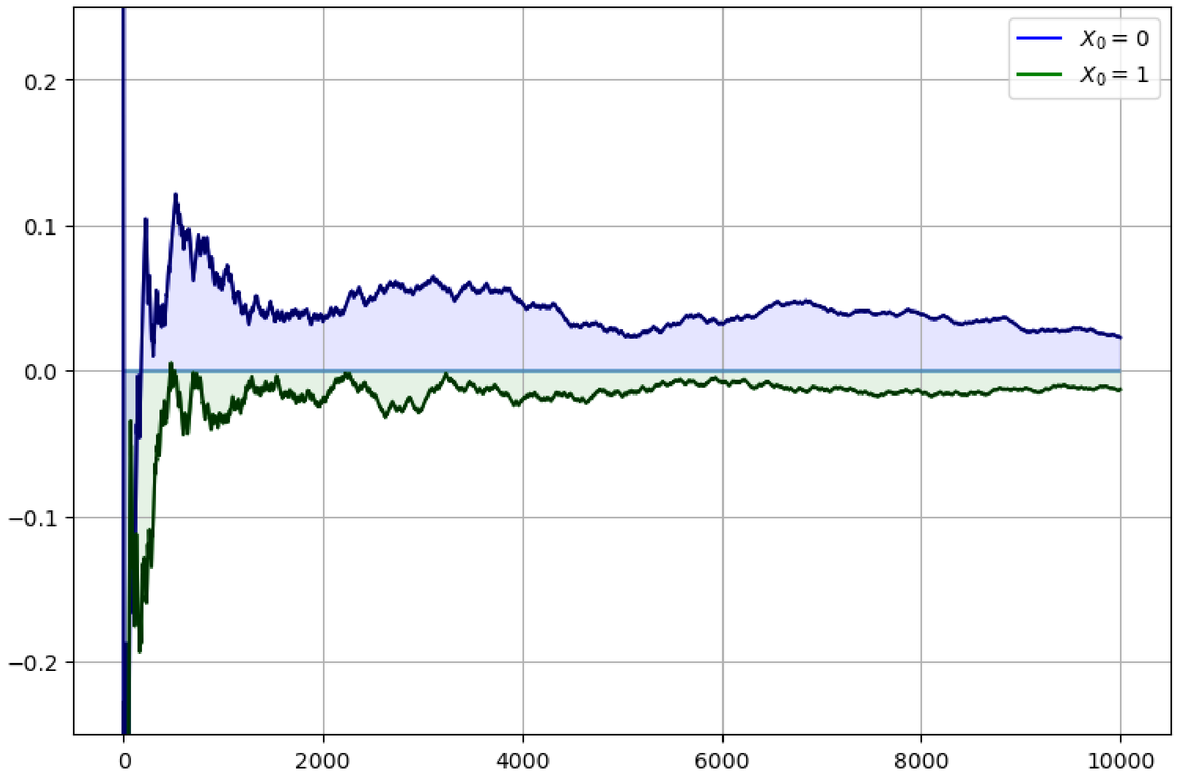
As m gets large, both series converge to zero.

```
α = β = 0.1
N = 10000
p = β / (α + β)

P = ((1 - α,       α),               # Careful: P and p are distinct
     (    β,   1 - β))
```

(continues on next page)

(continued from previous page)

```
mc = MarkovChain(P)

fig, ax = plt.subplots(figsize=(9, 6))
ax.set_ylim(-0.25, 0.25)
ax.grid()
ax.hlines(0, 0, N, lw=2, alpha=0.6)   # Horizonal line at zero

for x0, col in ((0, 'blue'), (1, 'green')):
    # Generate time series for worker that starts at x0
    X = mc.simulate(N, init=x0)
    # Compute fraction of time spent unemployed, for each n
    X_bar = (X == 0).cumsum() / (1 + np.arange(N, dtype=float))
    # Plot
    ax.fill_between(range(N), np.zeros(N), X_bar - p, color=col, alpha=0.1)
    ax.plot(X_bar - p, color=col, label=f'$X_0 = \, {x0} $')
    # Overlay in black--make lines clearer
    ax.plot(X_bar - p, 'k-', alpha=0.6)

ax.legend(loc='upper right')
plt.show()
```



Exercise 13.9.2

A topic of interest for economics and many other disciplines is *ranking*.

Let's now consider one of the most practical and important ranking problems — the rank assigned to web pages by search engines.

(Although the problem is motivated from outside of economics, there is in fact a deep connection between search ranking systems and prices in certain competitive equilibria — see [Du *et al.*, 2013].)

To understand the issue, consider the set of results returned by a query to a web search engine.

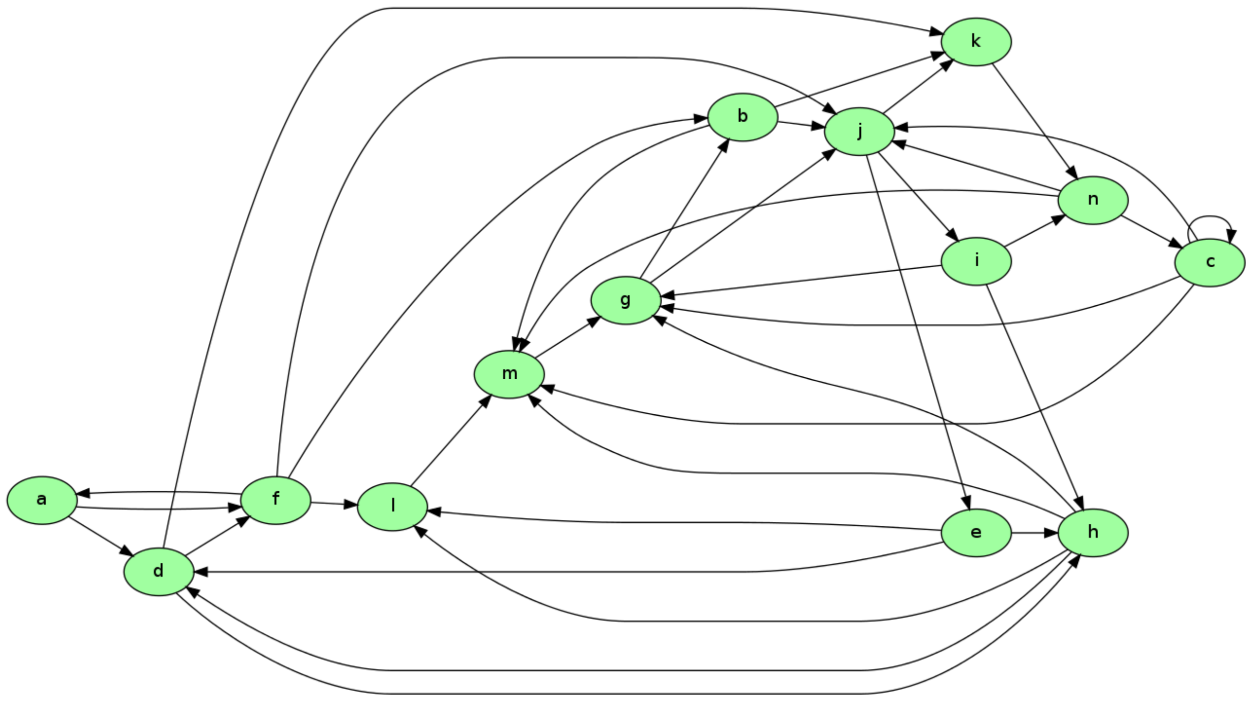
For the user, it is desirable to

1. receive a large set of accurate matches
2. have the matches returned in order, where the order corresponds to some measure of "importance"

Ranking according to a measure of importance is the problem we now consider.

The methodology developed to solve this problem by Google founders Larry Page and Sergey Brin is known as PageRank.

To illustrate the idea, consider the following diagram



Imagine that this is a miniature version of the WWW, with

- each node representing a web page
- each arrow representing the existence of a link from one page to another

Now let's think about which pages are likely to be important, in the sense of being valuable to a search engine user.

One possible criterion for the importance of a page is the number of inbound links — an indication of popularity.

By this measure, `m` and `j` are the most important pages, with 5 inbound links each.

However, what if the pages linking to `m`, say, are not themselves important?

Thinking this way, it seems appropriate to weight the inbound nodes by relative importance.

The PageRank algorithm does precisely this.

A slightly simplified presentation that captures the basic idea is as follows.

Letting j be (the integer index of) a typical page and r_j be its ranking, we set

$$r_j = \sum_{i \in L_j} \frac{r_i}{\ell_i}$$

where

- ℓ_i is the total number of outbound links from i
- L_j is the set of all pages i such that i has a link to j

This is a measure of the number of inbound links, weighted by their own ranking (and normalized by $1/\ell_i$).

There is, however, another interpretation, and it brings us back to Markov chains.

Let P be the matrix given by $P(i,j) = \mathbf{1}\{i \to j\}/\ell_i$ where $\mathbf{1}\{i \to j\} = 1$ if i has a link to j and zero otherwise.

The matrix P is a stochastic matrix provided that each page has at least one link.

With this definition of P we have

$$r_j = \sum_{i \in L_j} \frac{r_i}{\ell_i} = \sum_{\text{all } i} \mathbf{1}\{i \to j\} \frac{r_i}{\ell_i} = \sum_{\text{all } i} P(i,j) r_i$$

Writing r for the row vector of rankings, this becomes $r = rP$.

Hence r is the stationary distribution of the stochastic matrix P.

Let's think of $P(i,j)$ as the probability of "moving" from page i to page j.

The value $P(i,j)$ has the interpretation

- $P(i,j) = 1/k$ if i has k outbound links and j is one of them
- $P(i,j) = 0$ if i has no direct link to j

Thus, motion from page to page is that of a web surfer who moves from one page to another by randomly clicking on one of the links on that page.

Here "random" means that each link is selected with equal probability.

Since r is the stationary distribution of P, assuming that the uniform ergodicity condition is valid, we *can interpret* r_j as the fraction of time that a (very persistent) random surfer spends at page j.

Your exercise is to apply this ranking algorithm to the graph pictured above and return the list of pages ordered by rank.

There is a total of 14 nodes (i.e., web pages), the first named `a` and the last named `n`.

A typical line from the file has the form

```
d -> h;
```

This should be interpreted as meaning that there exists a link from `d` to `h`.

The data for this graph is shown below, and read into a file called `web_graph_data.txt` when the cell is executed.

```
%%file web_graph_data.txt
a -> d;
a -> f;
b -> j;
b -> k;
b -> m;
c -> c;
c -> g;
```

(continues on next page)

(continued from previous page)

```
c -> j;
c -> m;
d -> f;
d -> h;
d -> k;
e -> d;
e -> h;
e -> l;
f -> a;
f -> b;
f -> j;
f -> l;
g -> b;
g -> j;
h -> d;
h -> g;
h -> l;
h -> m;
i -> g;
i -> h;
i -> n;
j -> e;
j -> i;
j -> k;
k -> n;
l -> m;
m -> g;
n -> c;
n -> j;
n -> m;
```

```
Overwriting web_graph_data.txt
```

To parse this file and extract the relevant information, you can use regular expressions.

The following code snippet provides a hint as to how you can go about this

```python
import re
re.findall('\w', 'x +++ y ****** z')  # \w matches alphanumerics
```

```
['x', 'y', 'z']
```

```python
re.findall('\w', 'a ^^ b &&& $$ c')
```

```
['a', 'b', 'c']
```

When you solve for the ranking, you will find that the highest ranked node is in fact `g`, while the lowest is `a`.

Solution to Exercise 13.9.2

Here is one solution:

```python
"""
Return list of pages, ordered by rank
"""
import re
from operator import itemgetter

infile = 'web_graph_data.txt'
alphabet = 'abcdefghijklmnopqrstuvwxyz'

n = 14 # Total number of web pages (nodes)

# Create a matrix Q indicating existence of links
#  * Q[i, j] = 1 if there is a link from i to j
#  * Q[i, j] = 0 otherwise
Q = np.zeros((n, n), dtype=int)
with open(infile) as f:
    edges = f.readlines()
for edge in edges:
    from_node, to_node = re.findall('\w', edge)
    i, j = alphabet.index(from_node), alphabet.index(to_node)
    Q[i, j] = 1
# Create the corresponding Markov matrix P
P = np.empty((n, n))
for i in range(n):
    P[i, :] = Q[i, :] / Q[i, :].sum()
mc = MarkovChain(P)
# Compute the stationary distribution r
r = mc.stationary_distributions[0]
ranked_pages = {alphabet[i] : r[i] for i in range(n)}
# Print solution, sorted from highest to lowest rank
print('Rankings\n ***')
for name, rank in sorted(ranked_pages.items(), key=itemgetter(1), reverse=1):
    print(f'{name}: {rank:.4}')
```

```
Rankings
 ***
g: 0.1607
j: 0.1594
m: 0.1195
n: 0.1088
k: 0.09106
b: 0.08326
e: 0.05312
i: 0.05312
c: 0.04834
h: 0.0456
l: 0.03202
d: 0.03056
f: 0.01164
a: 0.002911
```

Exercise 13.9.3

In numerical work, it is sometimes convenient to replace a continuous model with a discrete one.

In particular, Markov chains are routinely generated as discrete approximations to AR(1) processes of the form

$$y_{t+1} = \rho y_t + u_{t+1}$$

Here u_t is assumed to be IID and $N(0, \sigma_u^2)$.

The variance of the stationary probability distribution of $\{y_t\}$ is

$$\sigma_y^2 := \frac{\sigma_u^2}{1 - \rho^2}$$

Tauchen's method [Tauchen, 1986] is the most common method for approximating this continuous state process with a finite state Markov chain.

A routine for this already exists in QuantEcon.py but let's write our own version as an exercise.

As a first step, we choose

- n, the number of states for the discrete approximation
- m, an integer that parameterizes the width of the state space

Next, we create a state space $\{x_0, \ldots, x_{n-1}\} \subset \mathbb{R}$ and a stochastic $n \times n$ matrix P such that

- $x_0 = -m\,\sigma_y$
- $x_{n-1} = m\,\sigma_y$
- $x_{i+1} = x_i + s$ where $s = (x_{n-1} - x_0)/(n-1)$

Let F be the cumulative distribution function of the normal distribution $N(0, \sigma_u^2)$.

The values $P(x_i, x_j)$ are computed to approximate the AR(1) process — omitting the derivation, the rules are as follows:

1. If $j = 0$, then set

$$P(x_i, x_j) = P(x_i, x_0) = F(x_0 - \rho x_i + s/2)$$

2. If $j = n - 1$, then set

$$P(x_i, x_j) = P(x_i, x_{n-1}) = 1 - F(x_{n-1} - \rho x_i - s/2)$$

3. Otherwise, set

$$P(x_i, x_j) = F(x_j - \rho x_i + s/2) - F(x_j - \rho x_i - s/2)$$

The exercise is to write a function `approx_markov(rho, sigma_u, m=3, n=7)` that returns $\{x_0, \ldots, x_{n-1}\} \subset \mathbb{R}$ and $n \times n$ matrix P as described above.

- Even better, write a function that returns an instance of QuantEcon.py's MarkovChain class.

Solution to Exercise 13.9.3

A solution from the QuantEcon.py library can be found here.

CHAPTER

FOURTEEN

CONTINUOUS STATE MARKOV CHAINS

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
```

14.1 Overview

In a previous lecture, we learned about finite Markov chains, a relatively elementary class of stochastic dynamic models.

The present lecture extends this analysis to continuous (i.e., uncountable) state Markov chains.

Most stochastic dynamic models studied by economists either fit directly into this class or can be represented as continuous state Markov chains after minor modifications.

In this lecture, our focus will be on continuous Markov models that

- evolve in discrete-time
- are often nonlinear

The fact that we accommodate nonlinear models here is significant, because linear stochastic models have their own highly developed toolset, as we'll see *later on*.

The question that interests us most is: Given a particular stochastic dynamic model, how will the state of the system evolve over time?

In particular,

- What happens to the distribution of the state variables?
- Is there anything we can say about the "average behavior" of these variables?
- Is there a notion of "steady state" or "long-run equilibrium" that's applicable to the model?
  - If so, how can we compute it?

Answering these questions will lead us to revisit many of the topics that occupied us in the finite state case, such as simulation, distribution dynamics, stability, ergodicity, etc.

Note: For some people, the term "Markov chain" always refers to a process with a finite or discrete state space. We follow the mainstream mathematical literature (e.g., [Meyn and Tweedie, 2009]) in using the term to refer to any discrete **time** Markov process.

Let's begin with some imports:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import lognorm, beta
from quantecon import LAE
from scipy.stats import norm, gaussian_kde
```

14.2 The Density Case

You are probably aware that some distributions can be represented by densities and some cannot.

(For example, distributions on the real numbers $\mathbb{R}$ that put positive probability on individual points have no density representation)

We are going to start our analysis by looking at Markov chains where the one-step transition probabilities have density representations.

The benefit is that the density case offers a very direct parallel to the finite case in terms of notation and intuition.

Once we've built some intuition we'll cover the general case.

14.2.1 Definitions and Basic Properties

In our lecture on finite Markov chains, we studied discrete-time Markov chains that evolve on a finite state space S.

In this setting, the dynamics of the model are described by a stochastic matrix — a nonnegative square matrix $P = P[i, j]$ such that each row $P[i, \cdot]$ sums to one.

The interpretation of P is that $P[i, j]$ represents the probability of transitioning from state i to state j in one unit of time.

In symbols,

$$\mathbb{P}\{X_{t+1} = j \,|\, X_t = i\} = P[i, j]$$

Equivalently,

- P can be thought of as a family of distributions $P[i, \cdot]$, one for each $i \in S$
- $P[i, \cdot]$ is the distribution of X_{t+1} given $X_t = i$

(As you probably recall, when using NumPy arrays, $P[i, \cdot]$ is expressed as `P[i,:]`)

In this section, we'll allow S to be a subset of $\mathbb{R}$, such as

- $\mathbb{R}$ itself
- the positive reals $(0, \infty)$
- a bounded interval (a, b)

The family of discrete distributions $P[i, \cdot]$ will be replaced by a family of densities $p(x, \cdot)$, one for each $x \in S$.

Analogous to the finite state case, $p(x, \cdot)$ is to be understood as the distribution (density) of X_{t+1} given $X_t = x$.

More formally, a *stochastic kernel on* S is a function $p: S \times S \to \mathbb{R}$ with the property that

1. $p(x, y) \geq 0$ for all $x, y \in S$
2. $\int p(x, y) dy = 1$ for all $x \in S$

(Integrals are over the whole space unless otherwise specified)

For example, let $S = \mathbb{R}$ and consider the particular stochastic kernel p_w defined by

$$p_w(x, y) := \frac{1}{\sqrt{2\pi}} \exp\left\{ -\frac{(y-x)^2}{2} \right\} \tag{14.1}$$

What kind of model does p_w represent?

The answer is, the (normally distributed) random walk

$$X_{t+1} = X_t + \xi_{t+1} \quad \text{where} \quad \{\xi_t\} \overset{\text{IID}}{\sim} N(0,1) \tag{14.2}$$

To see this, let's find the stochastic kernel p corresponding to (14.2).

Recall that $p(x, \cdot)$ represents the distribution of X_{t+1} given $X_t = x$.

Letting $X_t = x$ in (14.2) and considering the distribution of X_{t+1}, we see that $p(x, \cdot) = N(x, 1)$.

In other words, p is exactly p_w, as defined in (14.1).

14.2.2 Connection to Stochastic Difference Equations

In the previous section, we made the connection between stochastic difference equation (14.2) and stochastic kernel (14.1).

In economics and time-series analysis we meet stochastic difference equations of all different shapes and sizes.

It will be useful for us if we have some systematic methods for converting stochastic difference equations into stochastic kernels.

To this end, consider the generic (scalar) stochastic difference equation given by

$$X_{t+1} = \mu(X_t) + \sigma(X_t)\, \xi_{t+1} \tag{14.3}$$

Here we assume that

- $\{\xi_t\} \overset{\text{IID}}{\sim} \phi$, where ϕ is a given density on $\mathbb{R}$
- μ and σ are given functions on S, with $\sigma(x) > 0$ for all x

Example 1: The random walk (14.2) is a special case of (14.3), with $\mu(x) = x$ and $\sigma(x) = 1$.

Example 2: Consider the ARCH model

$$X_{t+1} = \alpha X_t + \sigma_t\, \xi_{t+1}, \qquad \sigma_t^2 = \beta + \gamma X_t^2, \qquad \beta, \gamma > 0$$

Alternatively, we can write the model as

$$X_{t+1} = \alpha X_t + (\beta + \gamma X_t^2)^{1/2} \xi_{t+1} \tag{14.4}$$

This is a special case of (14.3) with $\mu(x) = \alpha x$ and $\sigma(x) = (\beta + \gamma x^2)^{1/2}$.

Example 3: With stochastic production and a constant savings rate, the one-sector neoclassical growth model leads to a law of motion for capital per worker such as

$$k_{t+1} = s A_{t+1} f(k_t) + (1 - \delta) k_t \tag{14.5}$$

Here

- s is the rate of savings
- A_{t+1} is a production shock

  - The $t+1$ subscript indicates that A_{t+1} is not visible at time t
- δ is a depreciation rate
- $f \colon \mathbb{R}_+ \to \mathbb{R}_+$ is a production function satisfying $f(k) > 0$ whenever $k > 0$

(The fixed savings rate can be rationalized as the optimal policy for a particular set of technologies and preferences (see [Ljungqvist and Sargent, 2018], section 3.1.2), although we omit the details here).

Equation (14.5) is a special case of (14.3) with $\mu(x) = (1-\delta)x$ and $\sigma(x) = sf(x)$.

Now let's obtain the stochastic kernel corresponding to the generic model (14.3).

To find it, note first that if U is a random variable with density f_U, and $V = a + bU$ for some constants a, b with $b > 0$, then the density of V is given by

$$f_V(v) = \frac{1}{b} f_U \left(\frac{v-a}{b} \right) \tag{14.6}$$

(The proof is *below*. For a multidimensional version see EDTC, theorem 8.1.3).

Taking (14.6) as given for the moment, we can obtain the stochastic kernel p for (14.3) by recalling that $p(x, \cdot)$ is the conditional density of X_{t+1} given $X_t = x$.

In the present case, this is equivalent to stating that $p(x, \cdot)$ is the density of $Y := \mu(x) + \sigma(x)\,\xi_{t+1}$ when $\xi_{t+1} \sim \phi$.

Hence, by (14.6),

$$p(x, y) = \frac{1}{\sigma(x)} \phi \left(\frac{y - \mu(x)}{\sigma(x)} \right) \tag{14.7}$$

For example, the growth model in (14.5) has stochastic kernel

$$p(x, y) = \frac{1}{sf(x)} \phi \left(\frac{y - (1-\delta)x}{sf(x)} \right) \tag{14.8}$$

where ϕ is the density of A_{t+1}.

(Regarding the state space S for this model, a natural choice is $(0, \infty)$ — in which case $\sigma(x) = sf(x)$ is strictly positive for all s as required)

14.2.3 Distribution Dynamics

In this section of our lecture on **finite** Markov chains, we asked the following question: If

1. $\{X_t\}$ is a Markov chain with stochastic matrix P
2. the distribution of X_t is known to be ψ_t

then what is the distribution of X_{t+1}?

Letting ψ_{t+1} denote the distribution of X_{t+1}, the answer we gave was that

$$\psi_{t+1}[j] = \sum_{i \in S} P[i,j] \psi_t[i]$$

This intuitive equality states that the probability of being at j tomorrow is the probability of visiting i today and then going on to j, summed over all possible i.

In the density case, we just replace the sum with an integral and probability mass functions with densities, yielding

$$\psi_{t+1}(y) = \int p(x,y) \psi_t(x) \, dx, \qquad \forall y \in S \tag{14.9}$$

It is convenient to think of this updating process in terms of an operator.

(An operator is just a function, but the term is usually reserved for a function that sends functions into functions)

Let $\mathscr{D}$ be the set of all densities on S, and let P be the operator from $\mathscr{D}$ to itself that takes density ψ and sends it into new density ψP, where the latter is defined by

$$(\psi P)(y) = \int p(x, y)\psi(x)dx \tag{14.10}$$

This operator is usually called the *Markov operator* corresponding to p

Note: Unlike most operators, we write P to the right of its argument, instead of to the left (i.e., ψP instead of $P\psi$). This is a common convention, with the intention being to maintain the parallel with the finite case — see here

With this notation, we can write (14.9) more succinctly as $\psi_{t+1}(y) = (\psi_t P)(y)$ for all y, or, dropping the y and letting "$=$" indicate equality of functions,

$$\psi_{t+1} = \psi_t P \tag{14.11}$$

Equation (14.11) tells us that if we specify a distribution for ψ_0, then the entire sequence of future distributions can be obtained by iterating with P.

It's interesting to note that (14.11) is a deterministic difference equation.

Thus, by converting a stochastic difference equation such as (14.3) into a stochastic kernel p and hence an operator P, we convert a stochastic difference equation into a deterministic one (albeit in a much higher dimensional space).

Note: Some people might be aware that discrete Markov chains are in fact a special case of the continuous Markov chains we have just described. The reason is that probability mass functions are densities with respect to the counting measure.

14.2.4 Computation

To learn about the dynamics of a given process, it's useful to compute and study the sequences of densities generated by the model.

One way to do this is to try to implement the iteration described by (14.10) and (14.11) using numerical integration.

However, to produce ψP from ψ via (14.10), you would need to integrate at every y, and there is a continuum of such y.

Another possibility is to discretize the model, but this introduces errors of unknown size.

A nicer alternative in the present setting is to combine simulation with an elegant estimator called the *look-ahead* estimator.

Let's go over the ideas with reference to the growth model *discussed above*, the dynamics of which we repeat here for convenience:

$$k_{t+1} = sA_{t+1}f(k_t) + (1-\delta)k_t \tag{14.12}$$

Our aim is to compute the sequence $\{\psi_t\}$ associated with this model and fixed initial condition ψ_0.

To approximate ψ_t by simulation, recall that, by definition, ψ_t is the density of k_t given $k_0 \sim \psi_0$.

If we wish to generate observations of this random variable, all we need to do is

1. draw k_0 from the specified initial condition ψ_0

2. draw the shocks $A_1, \dots, A_t$ from their specified density ϕ
3. compute k_t iteratively via (14.12)

If we repeat this n times, we get n independent observations $k_t^1, \dots, k_t^n$.

With these draws in hand, the next step is to generate some kind of representation of their distribution ψ_t.

A naive approach would be to use a histogram, or perhaps a smoothed histogram using SciPy's `gaussian_kde` function.

However, in the present setting, there is a much better way to do this, based on the look-ahead estimator.

With this estimator, to construct an estimate of ψ_t, we actually generate n observations of k_{t-1}, rather than k_t.

Now we take these n observations $k_{t-1}^1, \dots, k_{t-1}^n$ and form the estimate

$$\psi_t^n(y) = \frac{1}{n} \sum_{i=1}^n p(k_{t-1}^i, y) \tag{14.13}$$

where p is the growth model stochastic kernel in (14.8).

What is the justification for this slightly surprising estimator?

The idea is that, by the strong law of large numbers,

$$\frac{1}{n} \sum_{i=1}^n p(k_{t-1}^i, y) \to \mathbb{E} p(k_{t-1}^i, y) = \int p(x, y) \psi_{t-1}(x) \, dx = \psi_t(y)$$

with probability one as $n \to \infty$.

Here the first equality is by the definition of ψ_{t-1}, and the second is by (14.9).

We have just shown that our estimator $\psi_t^n(y)$ in (14.13) converges almost surely to $\psi_t(y)$, which is just what we want to compute.

In fact, much stronger convergence results are true (see, for example, this paper).

14.2.5 Implementation

A class called `LAE` for estimating densities by this technique can be found in lae.py.

Given our use of the `__call__` method, an instance of `LAE` acts as a callable object, which is essentially a function that can store its own data (see this discussion).

This function returns the right-hand side of (14.13) using

- the data and stochastic kernel that it stores as its instance data
- the value y as its argument

The function is vectorized, in the sense that if `psi` is such an instance and `y` is an array, then the call `psi(y)` acts elementwise.

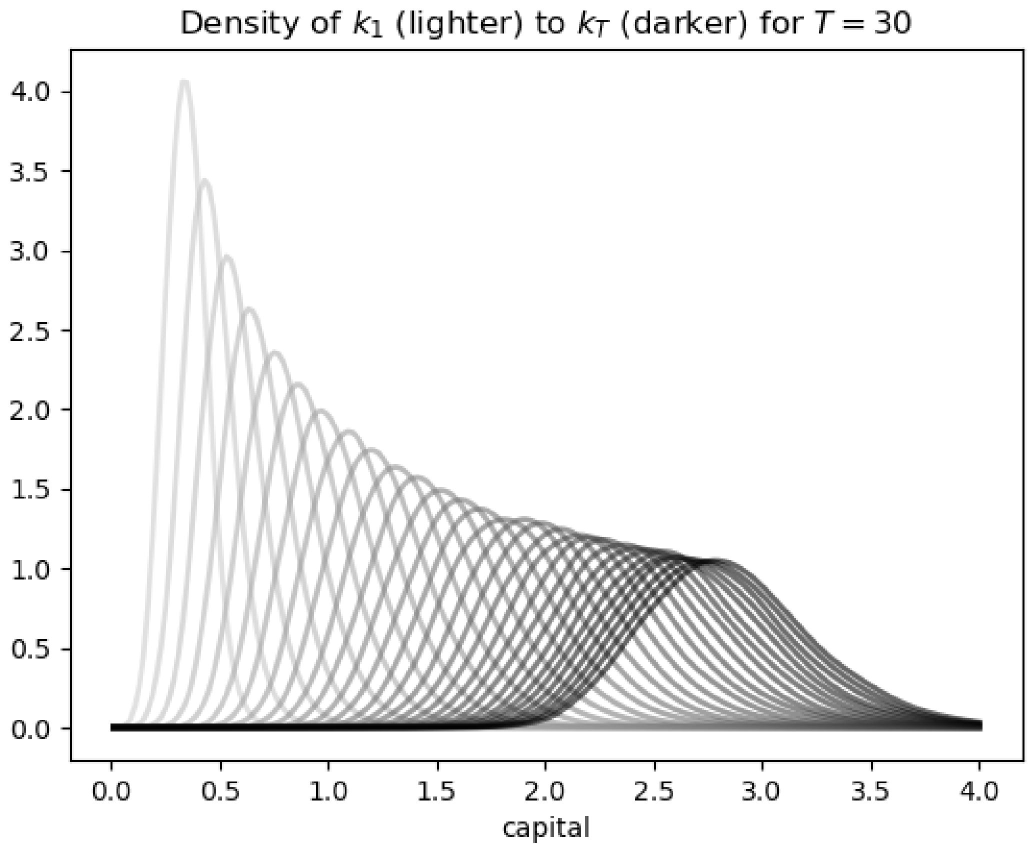
(This is the reason that we reshaped `X` and `y` inside the class — to make vectorization work)

Because the implementation is fully vectorized, it is about as efficient as it would be in C or Fortran.

14.2.6 Example

The following code is an example of usage for the stochastic growth model *described above*

```python
# == Define parameters == #
s = 0.2
δ = 0.1
a_σ = 0.4                    # A = exp(B) where B ~ N(0, a_σ)
α = 0.4                      # We set f(k) = k**α
ψ_0 = beta(5, 5, scale=0.5)  # Initial distribution
ϕ = lognorm(a_σ)


def p(x, y):
    """
    Stochastic kernel for the growth model with Cobb-Douglas production.
    Both x and y must be strictly positive.
    """
    d = s * x**α
    return ϕ.pdf((y - (1 - δ) * x) / d) / d

n = 10000    # Number of observations at each date t
T = 30       # Compute density of k_t at 1,...,T+1

# == Generate matrix s.t. t-th column is n observations of k_t == #
k = np.empty((n, T))
A = ϕ.rvs((n, T))
k[:, 0] = ψ_0.rvs(n)  # Draw first column from initial distribution
for t in range(T-1):
    k[:, t+1] = s * A[:, t] * k[:, t]**α + (1 - δ) * k[:, t]

# == Generate T instances of LAE using this data, one for each date t == #
laes = [LAE(p, k[:, t]) for t in range(T)]

# == Plot == #
fig, ax = plt.subplots()
ygrid = np.linspace(0.01, 4.0, 200)
greys = [str(g) for g in np.linspace(0.0, 0.8, T)]
greys.reverse()
for ψ, g in zip(laes, greys):
    ax.plot(ygrid, ψ(ygrid), color=g, lw=2, alpha=0.6)
ax.set_xlabel('capital')
ax.set_title(f'Density of $k_1$ (lighter) to $k_T$ (darker) for $T={T}$')
plt.show()
```



The figure shows part of the density sequence $\{\psi_t\}$, with each density computed via the look-ahead estimator.

Notice that the sequence of densities shown in the figure seems to be converging — more on this in just a moment.

Another quick comment is that each of these distributions could be interpreted as a cross-sectional distribution (recall this discussion).

14.3 Beyond Densities

Up until now, we have focused exclusively on continuous state Markov chains where all conditional distributions $p(x, \cdot)$ are densities.

As discussed above, not all distributions can be represented as densities.

If the conditional distribution of X_{t+1} given $X_t = x$ **cannot** be represented as a density for some $x \in S$, then we need a slightly different theory.

The ultimate option is to switch from densities to probability measures, but not all readers will be familiar with measure theory.

We can, however, construct a fairly general theory using distribution functions.

14.3.1 Example and Definitions

To illustrate the issues, recall that Hopenhayn and Rogerson [Hopenhayn and Rogerson, 1993] study a model of firm dynamics where individual firm productivity follows the exogenous process

$$X_{t+1} = a + \rho X_t + \xi_{t+1}, \quad \text{where} \quad \{\xi_t\} \overset{\text{IID}}{\sim} N(0, \sigma^2)$$

As is, this fits into the density case we treated above.

However, the authors wanted this process to take values in $[0, 1]$, so they added boundaries at the endpoints 0 and 1.

One way to write this is

$$X_{t+1} = h(a + \rho X_t + \xi_{t+1}) \quad \text{where} \quad h(x) := x\,\mathbf{1}\{0 \leq x \leq 1\} + \mathbf{1}\{x > 1\}$$

If you think about it, you will see that for any given $x \in [0, 1]$, the conditional distribution of X_{t+1} given $X_t = x$ puts positive probability mass on 0 and 1.

Hence it cannot be represented as a density.

What we can do instead is use cumulative distribution functions (cdfs).

To this end, set

$$G(x, y) := \mathbb{P}\{h(a + \rho x + \xi_{t+1}) \leq y\} \qquad (0 \leq x, y \leq 1)$$

This family of cdfs $G(x, \cdot)$ plays a role analogous to the stochastic kernel in the density case.

The distribution dynamics in (14.9) are then replaced by

$$F_{t+1}(y) = \int G(x, y) F_t(dx) \tag{14.14}$$

Here F_t and F_{t+1} are cdfs representing the distribution of the current state and next period state.

The intuition behind (14.14) is essentially the same as for (14.9).

14.3.2 Computation

If you wish to compute these cdfs, you cannot use the look-ahead estimator as before.

Indeed, you should not use any density estimator, since the objects you are estimating/computing are not densities.

One good option is simulation as before, combined with the empirical distribution function.

14.4 Stability

In our lecture on finite Markov chains, we also studied stationarity, stability and ergodicity.

Here we will cover the same topics for the continuous case.

We will, however, treat only the density case (as in *this section*), where the stochastic kernel is a family of densities.

The general case is relatively similar — references are given below.

14.4.1 Theoretical Results

Analogous to the finite case, given a stochastic kernel p and corresponding Markov operator as defined in (14.10), a density ψ^* on S is called *stationary* for P if it is a fixed point of the operator P.

In other words,

$$\psi^*(y) = \int p(x, y)\psi^*(x)\,dx, \qquad \forall y \in S \tag{14.15}$$

As with the finite case, if ψ^* is stationary for P, and the distribution of X_0 is ψ^*, then, in view of (14.11), X_t will have this same distribution for all t.

Hence ψ^* is the stochastic equivalent of a steady state.

In the finite case, we learned that at least one stationary distribution exists, although there may be many.

When the state space is infinite, the situation is more complicated.

Even existence can fail very easily.

For example, the random walk model has no stationary density (see, e.g., EDTC, p. 210).

However, there are well-known conditions under which a stationary density ψ^* exists.

With additional conditions, we can also get a unique stationary density ($\psi \in \mathscr{D}$ and $\psi = \psi P \implies \psi = \psi^*$), and also global convergence in the sense that

$$\forall\, \psi \in \mathscr{D}, \quad \psi P^t \to \psi^* \quad \text{as} \quad t \to \infty \tag{14.16}$$

This combination of existence, uniqueness and global convergence in the sense of (14.16) is often referred to as *global stability*.

Under very similar conditions, we get *ergodicity*, which means that

$$\frac{1}{n}\sum_{t=1}^{n} h(X_t) \to \int h(x)\psi^*(x)dx \quad \text{as } n \to \infty \tag{14.17}$$

for any (measurable) function $h\colon S \to \mathbb{R}$ such that the right-hand side is finite.

Note that the convergence in (14.17) does not depend on the distribution (or value) of X_0.

This is actually very important for simulation — it means we can learn about ψ^* (i.e., approximate the right-hand side of (14.17) via the left-hand side) without requiring any special knowledge about what to do with X_0.

So what are these conditions we require to get global stability and ergodicity?

In essence, it must be the case that

1. Probability mass does not drift off to the "edges" of the state space.
2. Sufficient "mixing" obtains.

For one such set of conditions see theorem 8.2.14 of EDTC.

In addition

- [Stokey *et al.*, 1989] contains a classic (but slightly outdated) treatment of these topics.
- From the mathematical literature, [Lasota and MacKey, 1994] and [Meyn and Tweedie, 2009] give outstanding in-depth treatments.
- Section 8.1.2 of EDTC provides detailed intuition, and section 8.3 gives additional references.
- EDTC, section 11.3.4 provides a specific treatment for the growth model we considered in this lecture.

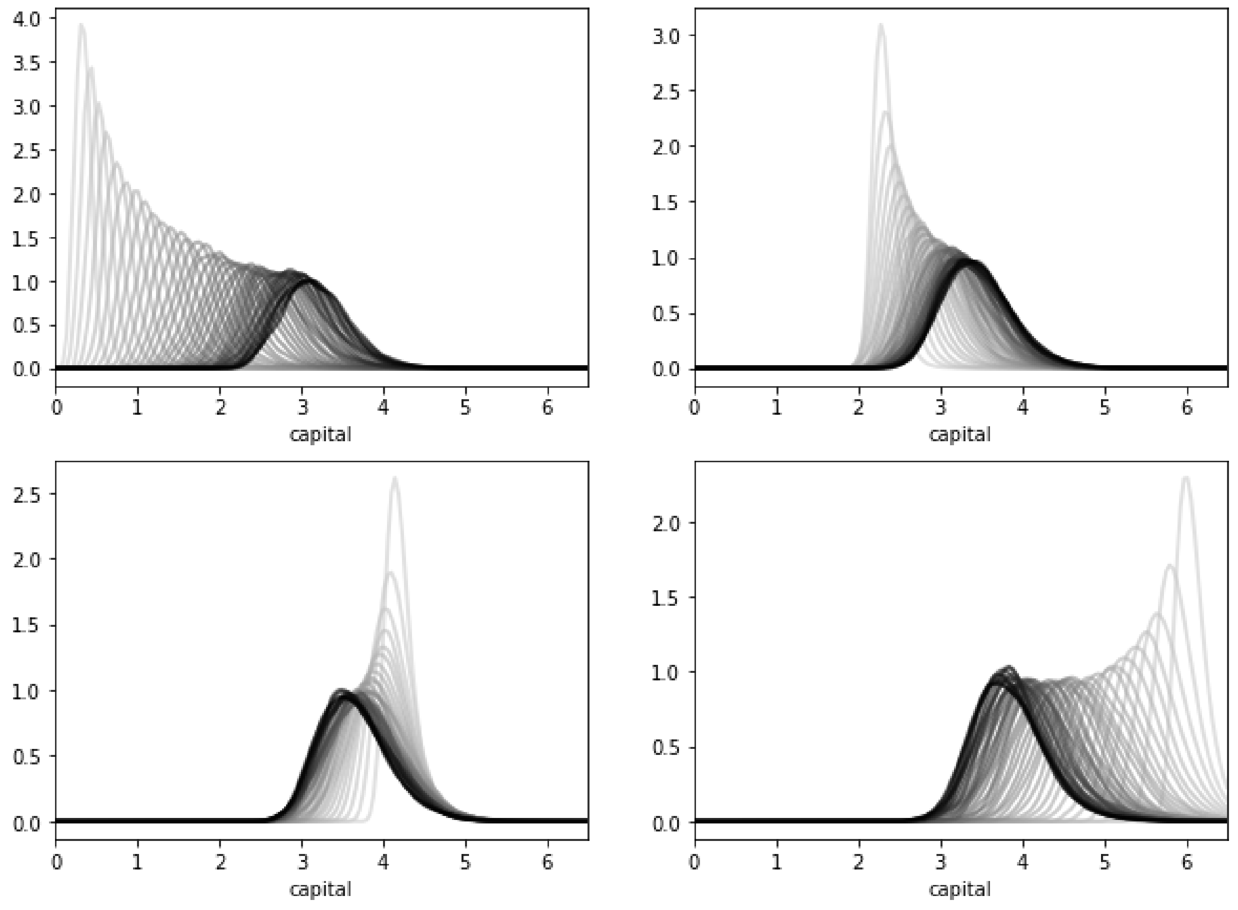
14.4.2 An Example of Stability

As stated above, the *growth model treated here* is stable under mild conditions on the primitives.

- See EDTC, section 11.3.4 for more details.

We can see this stability in action — in particular, the convergence in (14.16) — by simulating the path of densities from various initial conditions.

Here is such a figure.



All sequences are converging towards the same limit, regardless of their initial condition.

The details regarding initial conditions and so on are given in *this exercise*, where you are asked to replicate the figure.

14.4.3 Computing Stationary Densities

In the preceding figure, each sequence of densities is converging towards the unique stationary density ψ^*.

Even from this figure, we can get a fair idea what ψ^* looks like, and where its mass is located.

However, there is a much more direct way to estimate the stationary density, and it involves only a slight modification of the look-ahead estimator.

Let's say that we have a model of the form (14.3) that is stable and ergodic.

Let p be the corresponding stochastic kernel, as given in (14.7).

To approximate the stationary density ψ^*, we can simply generate a long time-series $X_0, X_1, \dots, X_n$ and estimate ψ^* via

$$\psi_n^*(y) = \frac{1}{n} \sum_{t=1}^n p(X_t, y) \tag{14.18}$$

This is essentially the same as the look-ahead estimator (14.13), except that now the observations we generate are a single time-series, rather than a cross-section.

The justification for (14.18) is that, with probability one as $n \to \infty$,

$$\frac{1}{n} \sum_{t=1}^n p(X_t, y) \to \int p(x, y) \psi^*(x) \, dx = \psi^*(y)$$

where the convergence is by (14.17) and the equality on the right is by (14.15).

The right-hand side is exactly what we want to compute.

On top of this asymptotic result, it turns out that the rate of convergence for the look-ahead estimator is very good.

The first exercise helps illustrate this point.

14.5 Exercises

Exercise 14.5.1

Consider the simple threshold autoregressive model

$$X_{t+1} = \theta |X_t| + (1 - \theta^2)^{1/2} \xi_{t+1} \qquad \text{where} \quad \{\xi_t\} \stackrel{\textrm{ IID }}{\sim} N(0, 1) \tag{14.19}$$

This is one of those rare nonlinear stochastic models where an analytical expression for the stationary density is available.

In particular, provided that $|\theta| < 1$, there is a unique stationary density ψ^* given by

$$\psi^*(y) = 2 \, \phi(y) \, \Phi \left[\frac{\theta y}{(1 - \theta^2)^{1/2}} \right] \tag{14.20}$$

Here ϕ is the standard normal density and Φ is the standard normal cdf.

As an exercise, compute the look-ahead estimate of ψ^*, as defined in (14.18), and compare it with ψ^* in (14.20) to see whether they are indeed close for large n.

In doing so, set $\theta = 0.8$ and $n = 500$.

The next figure shows the result of such a computation

The additional density (black line) is a nonparametric kernel density estimate, added to the solution for illustration.

(You can try to replicate it before looking at the solution if you want to)

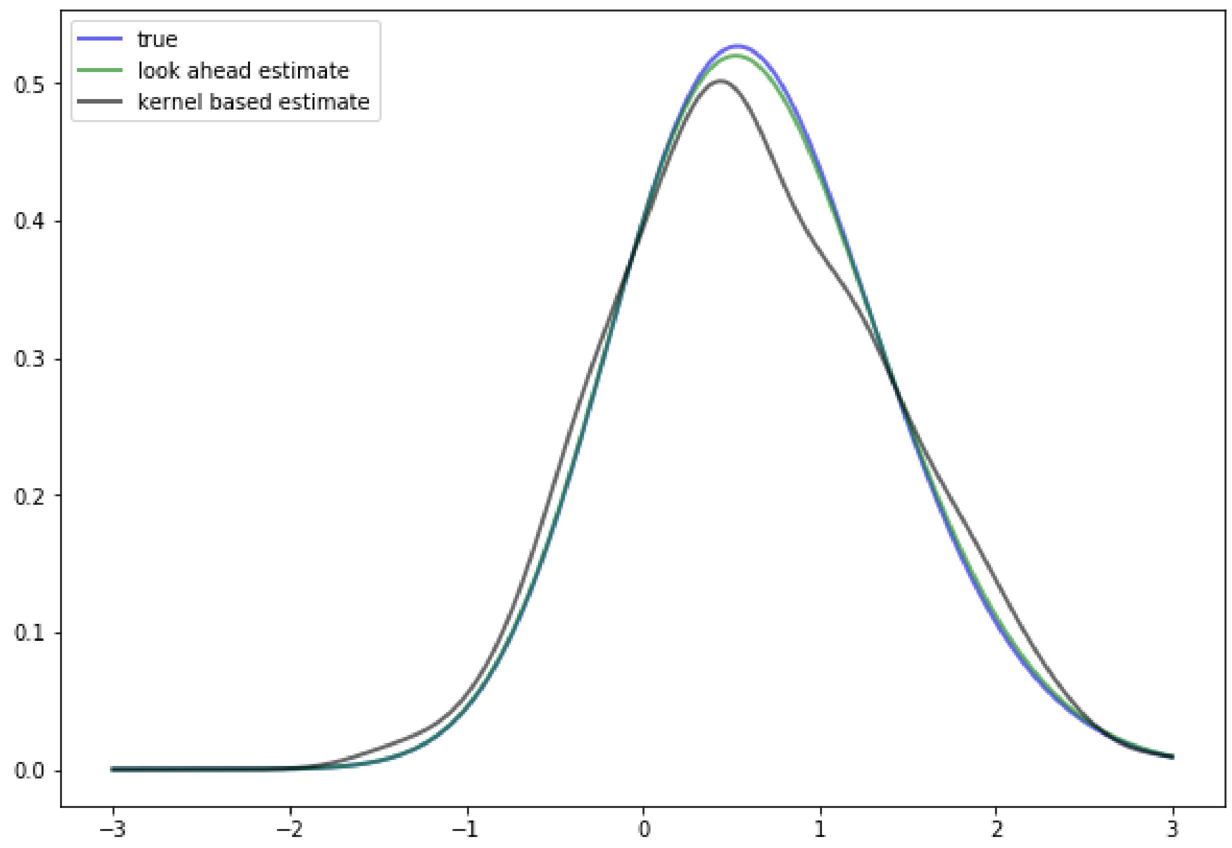
As you can see, the look-ahead estimator is a much tighter fit than the kernel density estimator.

If you repeat the simulation you will see that this is consistently the case.

Solution to Exercise 14.5.1

Look-ahead estimation of a TAR stationary density, where the TAR model is

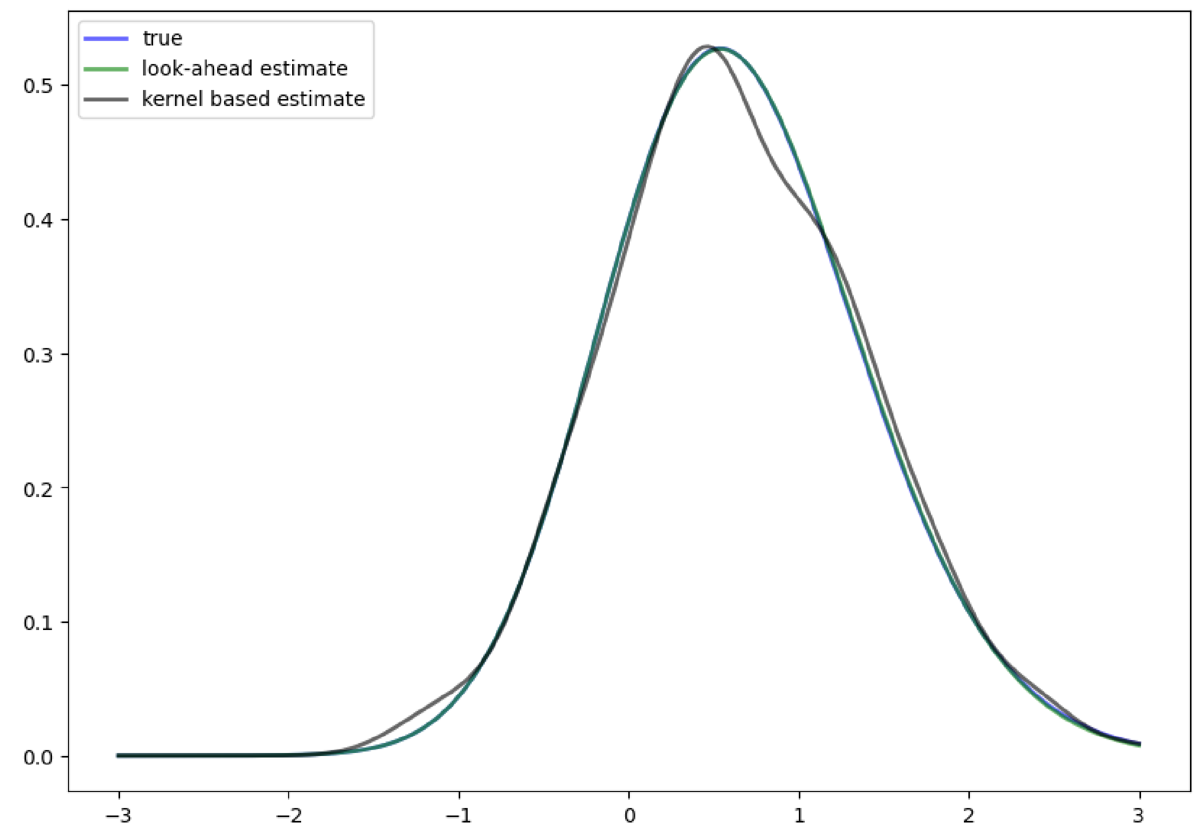
$$X_{t+1} = \theta |X_t| + (1 - \theta^2)^{1/2} \xi_{t+1}$$


true
look ahead estimate
kernel based estimate
0.0
0.1
0.2
0.3
0.4
0.5
−3
−2
−1
0
1
2
3

and $\xi_t \sim N(0, 1)$.

Try running at `n = 10, 100, 1000, 10000` to get an idea of the speed of convergence

```
ϕ = norm()
n = 500
θ = 0.8
# == Frequently used constants == #
d = np.sqrt(1 - θ**2)
δ = θ / d

def ψ_star(y):
    "True stationary density of the TAR Model"
    return 2 * norm.pdf(y) * norm.cdf(δ * y)

def p(x, y):
    "Stochastic kernel for the TAR model."
    return ϕ.pdf((y - θ * np.abs(x)) / d) / d

Z = ϕ.rvs(n)
X = np.empty(n)
for t in range(n-1):
    X[t+1] = θ * np.abs(X[t]) + d * Z[t]
ψ_est = LAE(p, X)
k_est = gaussian_kde(X)

fig, ax = plt.subplots(figsize=(10, 7))
ys = np.linspace(-3, 3, 200)
ax.plot(ys, ψ_star(ys), 'b-', lw=2, alpha=0.6, label='true')
ax.plot(ys, ψ_est(ys), 'g-', lw=2, alpha=0.6, label='look-ahead estimate')
ax.plot(ys, k_est(ys), 'k-', lw=2, alpha=0.6, label='kernel based estimate')
ax.legend(loc='upper left')
plt.show()
```



Exercise 14.5.2

Replicate the figure on global convergence *shown above*.

The densities come from the stochastic growth model treated *at the start of the lecture*.

Begin with the code found *above*.

Use the same parameters.

For the four initial distributions, use the shifted beta distributions

```
ψ_0 = beta(5, 5, scale=0.5, loc=i*2)
```

Solution to Exercise 14.5.2

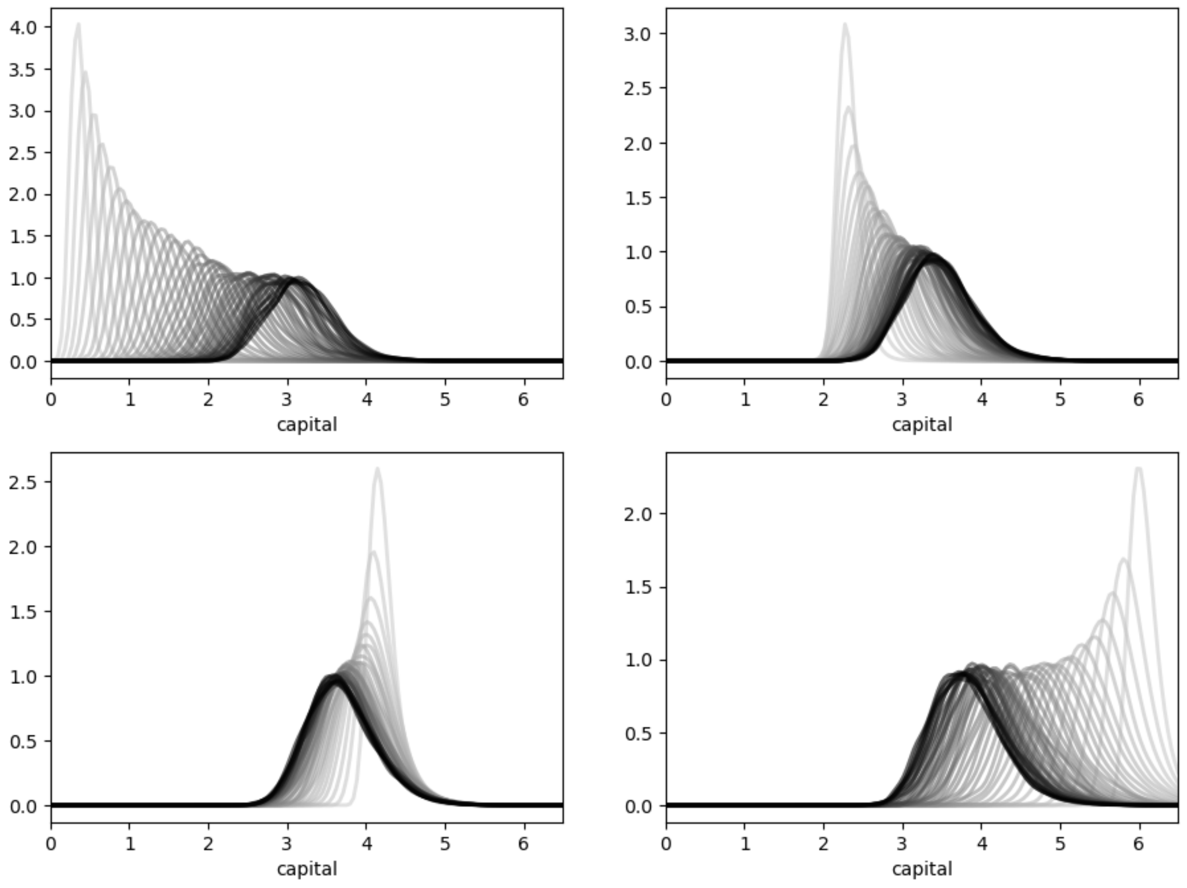
Here's one program that does the job

```
# == Define parameters == #
s = 0.2
δ = 0.1
a_σ = 0.4                                   # A = exp(B) where B ~ N(0, a_σ)
α = 0.4                                     # f(k) = k**α

ϕ = lognorm(a_σ)
```

(continues on next page)

(continued from previous page)

```
def p(x, y):
    "Stochastic kernel, vectorized in x.  Both x and y must be positive."
    d = s * x**α
    return ϕ.pdf((y - (1 - δ) * x) / d) / d

n = 1000                            # Number of observations at each date t
T = 40                              # Compute density of k_t at 1,...,T

fig, axes = plt.subplots(2, 2, figsize=(11, 8))
axes = axes.flatten()
xmax = 6.5

for i in range(4):
    ax = axes[i]
    ax.set_xlim(0, xmax)
    ψ_0 = beta(5, 5, scale=0.5, loc=i*2)  # Initial distribution

    # == Generate matrix s.t. t-th column is n observations of k_t == #
    k = np.empty((n, T))
    A = ϕ.rvs((n, T))
    k[:, 0] = ψ_0.rvs(n)
    for t in range(T-1):
        k[:, t+1] = s * A[:,t] * k[:, t]**α + (1 - δ) * k[:, t]

    # == Generate T instances of lae using this data, one for each t == #
    laes = [LAE(p, k[:, t]) for t in range(T)]

    ygrid = np.linspace(0.01, xmax, 150)
    greys = [str(g) for g in np.linspace(0.0, 0.8, T)]
    greys.reverse()
    for ψ, g in zip(laes, greys):
        ax.plot(ygrid, ψ(ygrid), color=g, lw=2, alpha=0.6)
    ax.set_xlabel('capital')
plt.show()
```



Exercise 14.5.3

A common way to compare distributions visually is with boxplots.

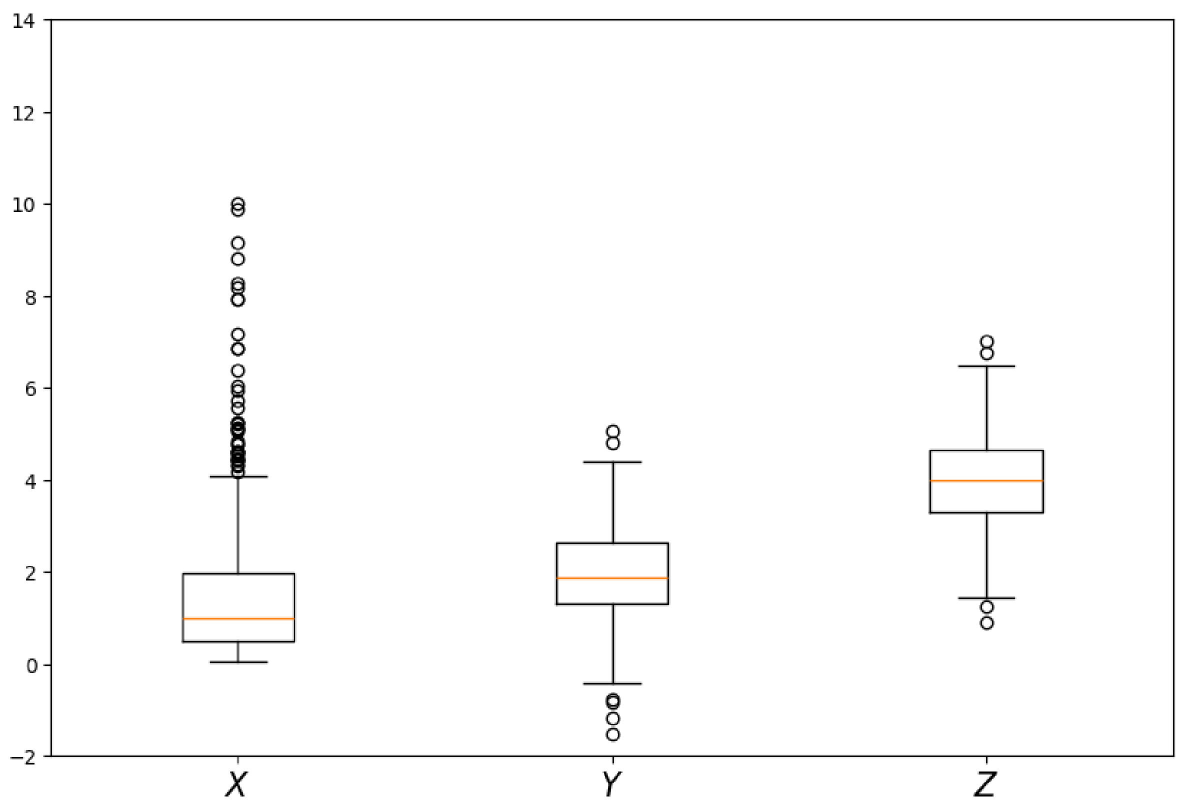
To illustrate, let's generate three artificial data sets and compare them with a boxplot.

The three data sets we will use are:

$$\{X_1, \dots, X_n\} \sim LN(0, 1), \quad \{Y_1, \dots, Y_n\} \sim N(2, 1), \quad \text{and} \quad \{Z_1, \dots, Z_n\} \sim N(4, 1),$$

Here is the code and figure:

```
n = 500
x = np.random.randn(n)        # N(0, 1)
x = np.exp(x)                 # Map x to lognormal
y = np.random.randn(n) + 2.0  # N(2, 1)
z = np.random.randn(n) + 4.0  # N(4, 1)

fig, ax = plt.subplots(figsize=(10, 6.6))
ax.boxplot([x, y, z])
ax.set_xticks((1, 2, 3))
ax.set_ylim(-2, 14)
ax.set_xticklabels(('$X$', '$Y$', '$Z$'), fontsize=16)
plt.show()
```



Each data set is represented by a box, where the top and bottom of the box are the third and first quartiles of the data, and the red line in the center is the median.

The boxes give some indication as to

- the location of probability mass for each sample
- whether the distribution is right-skewed (as is the lognormal distribution), etc

Now let's put these ideas to use in a simulation.

Consider the threshold autoregressive model in (14.19).

We know that the distribution of X_t will converge to (14.20) whenever $|\theta| < 1$.

Let's observe this convergence from different initial conditions using boxplots.

In particular, the exercise is to generate J boxplot figures, one for each initial condition X_0 in

```
initial_conditions = np.linspace(8, 0, J)
```

For each X_0 in this set,

1. Generate k time-series of length n, each starting at X_0 and obeying (14.19).
2. Create a boxplot representing n distributions, where the t-th distribution shows the k observations of X_t.

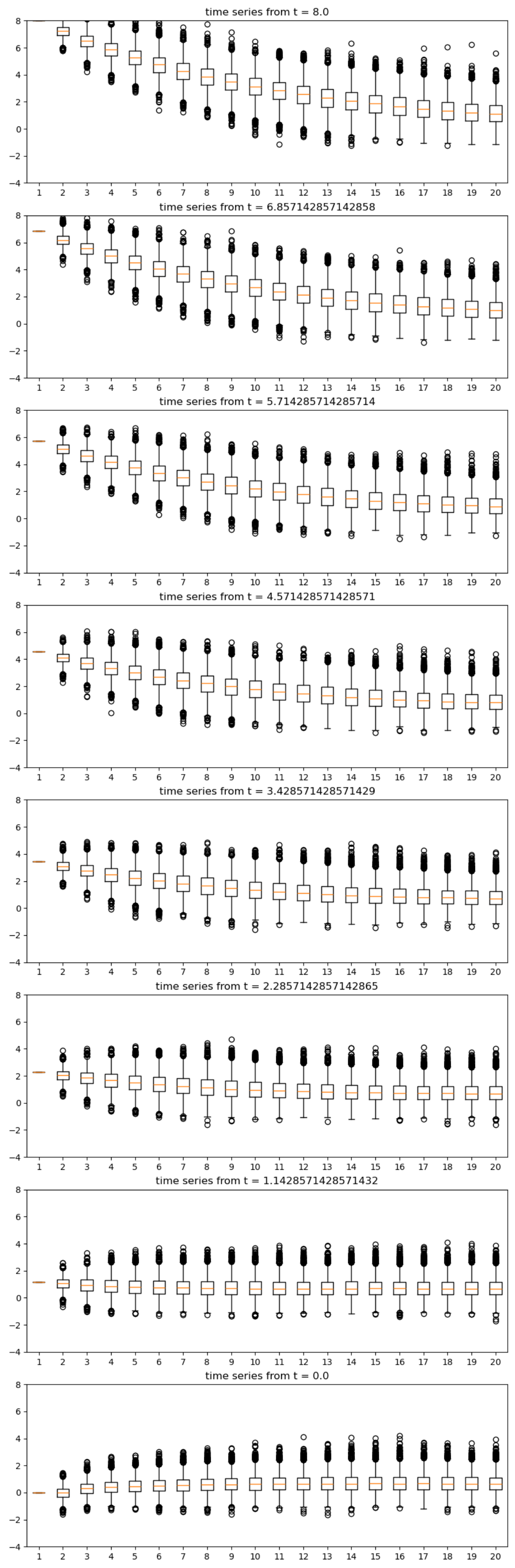
Use $\theta = 0.9, n = 20, k = 5000, J = 8$

Solution to Exercise 14.5.3

Here's a possible solution.

Note the way we use vectorized code to simulate the k time series for one boxplot all at once

```
n = 20
k = 5000
J = 8

θ = 0.9
d = np.sqrt(1 - θ**2)
δ = θ / d

fig, axes = plt.subplots(J, 1, figsize=(10, 4*J))
initial_conditions = np.linspace(8, 0, J)
X = np.empty((k, n))

for j in range(J):

    axes[j].set_ylim(-4, 8)
    axes[j].set_title(f'time series from t = {initial_conditions[j]}')

    Z = np.random.randn(k, n)
    X[:, 0] = initial_conditions[j]
    for t in range(1, n):
        X[:, t] = θ * np.abs(X[:, t-1]) + d * Z[:, t]
    axes[j].boxplot(X)

plt.show()
```


time series from t = 8.0
time series from t = 6.857142857142858
time series from t = 5.714285714285714
time series from t = 4.571428571428571
time series from t = 3.428571428571429
time series from t = 2.2857142857142865
time series from t = 1.1428571428571432
time series from t = 0.0

14.6 Appendix

Here's the proof of (14.6).

Let F_U and F_V be the cumulative distributions of U and V respectively.

By the definition of V, we have $F_V(v) = \mathbb{P}\{a + bU \leq v\} = \mathbb{P}\{U \leq (v - a)/b\}$.

In other words, $F_V(v) = F_U((v - a)/b)$.

Differentiating with respect to v yields (14.6).

CHAPTER

FIFTEEN

COVARIANCE STATIONARY PROCESSES

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
```

15.1 Overview

In this lecture we study covariance stationary linear stochastic processes, a class of models routinely used to study economic and financial time series.

This class has the advantage of being

1. simple enough to be described by an elegant and comprehensive theory
2. relatively broad in terms of the kinds of dynamics it can represent

We consider these models in both the time and frequency domain.

15.1.1 ARMA Processes

We will focus much of our attention on linear covariance stationary models with a finite number of parameters.

In particular, we will study stationary ARMA processes, which form a cornerstone of the standard theory of time series analysis.

Every ARMA process can be represented in linear state space form.

However, ARMA processes have some important structure that makes it valuable to study them separately.

15.1.2 Spectral Analysis

Analysis in the frequency domain is also called spectral analysis.

In essence, spectral analysis provides an alternative representation of the autocovariance function of a covariance stationary process.

Having a second representation of this important object

- shines a light on the dynamics of the process in question
- allows for a simpler, more tractable representation in some important cases

The famous *Fourier transform* and its inverse are used to map between the two representations.

15.1.3 Other Reading

For supplementary reading, see

- [Ljungqvist and Sargent, 2018], chapter 2
- [Sargent, 1987], chapter 11
- John Cochrane's notes on time series analysis, chapter 8
- [Shiriaev, 1995], chapter 6
- [Cryer and Chan, 2008], all

Let's start with some imports:

```
import numpy as np
import matplotlib.pyplot as plt
import quantecon as qe
```

15.2 Introduction

Consider a sequence of random variables $\{X_t\}$ indexed by $t \in \mathbb{Z}$ and taking values in $\mathbb{R}$.

Thus, $\{X_t\}$ begins in the infinite past and extends to the infinite future — a convenient and standard assumption.

As in other fields, successful economic modeling typically assumes the existence of features that are constant over time.

If these assumptions are correct, then each new observation $X_t, X_{t+1}, \ldots$ can provide additional information about the time-invariant features, allowing us to learn from as data arrive.

For this reason, we will focus in what follows on processes that are *stationary* — or become so after a transformation (see for example *this lecture*).

15.2.1 Definitions

A real-valued stochastic process $\{X_t\}$ is called *covariance stationary* if

1. Its mean $\mu := \mathbb{E} X_t$ does not depend on t.
2. For all k in $\mathbb{Z}$, the k-th autocovariance $\gamma(k) := \mathbb{E}(X_t - \mu)(X_{t+k} - \mu)$ is finite and depends only on k.

The function $\gamma \colon \mathbb{Z} \to \mathbb{R}$ is called the *autocovariance function* of the process.

Throughout this lecture, we will work exclusively with zero-mean (i.e., $\mu = 0$) covariance stationary processes.

The zero-mean assumption costs nothing in terms of generality since working with non-zero-mean processes involves no more than adding a constant.

15.2.2 Example 1: White Noise

Perhaps the simplest class of covariance stationary processes is the white noise processes.

A process $\{\epsilon_t\}$ is called a *white noise process* if

1. $\mathbb{E}\epsilon_t = 0$
2. $\gamma(k) = \sigma^2 \mathbf{1}\{k = 0\}$ for some $\sigma > 0$

(Here $\mathbf{1}\{k = 0\}$ is defined to be 1 if $k = 0$ and zero otherwise)

White noise processes play the role of **building blocks** for processes with more complicated dynamics.

15.2.3 Example 2: General Linear Processes

From the simple building block provided by white noise, we can construct a very flexible family of covariance stationary processes — the *general linear processes*

$$X_t = \sum_{j=0}^{\infty} \psi_j \epsilon_{t-j}, \qquad t \in \mathbb{Z} \tag{15.1}$$

where

- $\{\epsilon_t\}$ is white noise
- $\{\psi_t\}$ is a square summable sequence in $\mathbb{R}$ (that is, $\sum_{t=0}^{\infty} \psi_t^2 < \infty$)

The sequence $\{\psi_t\}$ is often called a *linear filter*.

Equation (15.1) is said to present a **moving average** process or a moving average representation.

With some manipulations, it is possible to confirm that the autocovariance function for (15.1) is

$$\gamma(k) = \sigma^2 \sum_{j=0}^{\infty} \psi_j \psi_{j+k} \tag{15.2}$$

By the Cauchy-Schwartz inequality, one can show that $\gamma(k)$ satisfies equation (15.2).

Evidently, $\gamma(k)$ does not depend on t.

15.2.4 Wold Representation

Remarkably, the class of general linear processes goes a long way towards describing the entire class of zero-mean covariance stationary processes.

In particular, Wold's decomposition theorem states that every zero-mean covariance stationary process $\{X_t\}$ can be written as

$$X_t = \sum_{j=0}^{\infty} \psi_j \epsilon_{t-j} + \eta_t$$

where

- $\{\epsilon_t\}$ is white noise
- $\{\psi_t\}$ is square summable
- $\psi_0 \epsilon_t$ is the one-step ahead prediction error in forecasting X_t as a linear least-squares function of the infinite history $X_{t-1}, X_{t-2}, \dots$

- η_t can be expressed as a linear function of $X_{t-1}, X_{t-2}, \ldots$ and is perfectly predictable over arbitrarily long horizons

For the method of constructing a Wold representation, intuition, and further discussion, see [Sargent, 1987], p. 286.

15.2.5 AR and MA

General linear processes are a very broad class of processes.

It often pays to specialize to those for which there exists a representation having only finitely many parameters.

(Experience and theory combine to indicate that models with a relatively small number of parameters typically perform better than larger models, especially for forecasting)

One very simple example of such a model is the first-order autoregressive or AR(1) process

$$X_t = \phi X_{t-1} + \epsilon_t \quad \text{where} \quad |\phi| < 1 \quad \text{and } \{\epsilon_t\} \text{ is white noise} \tag{15.3}$$

By direct substitution, it is easy to verify that $X_t = \sum_{j=0}^{\infty} \phi^j \epsilon_{t-j}$.

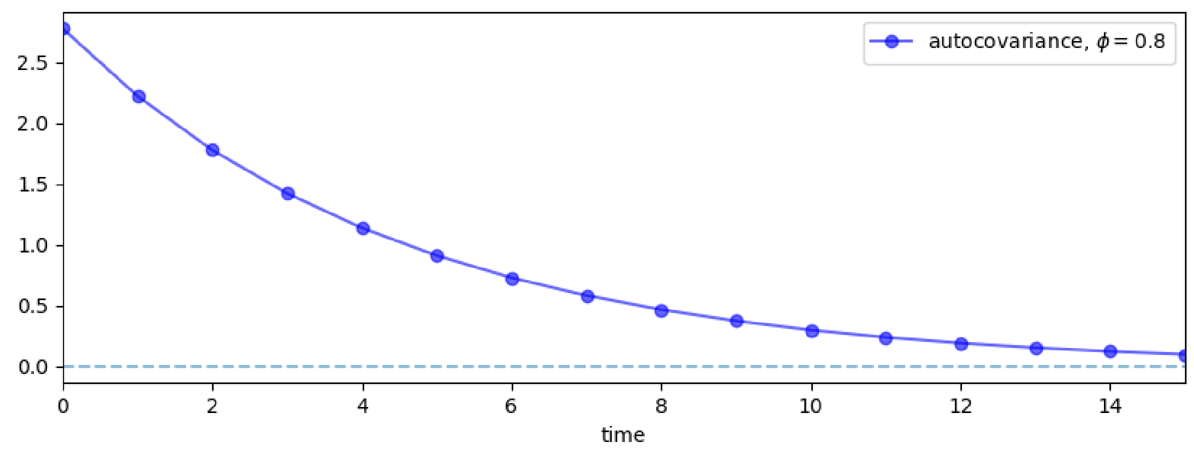
Hence $\{X_t\}$ is a general linear process.

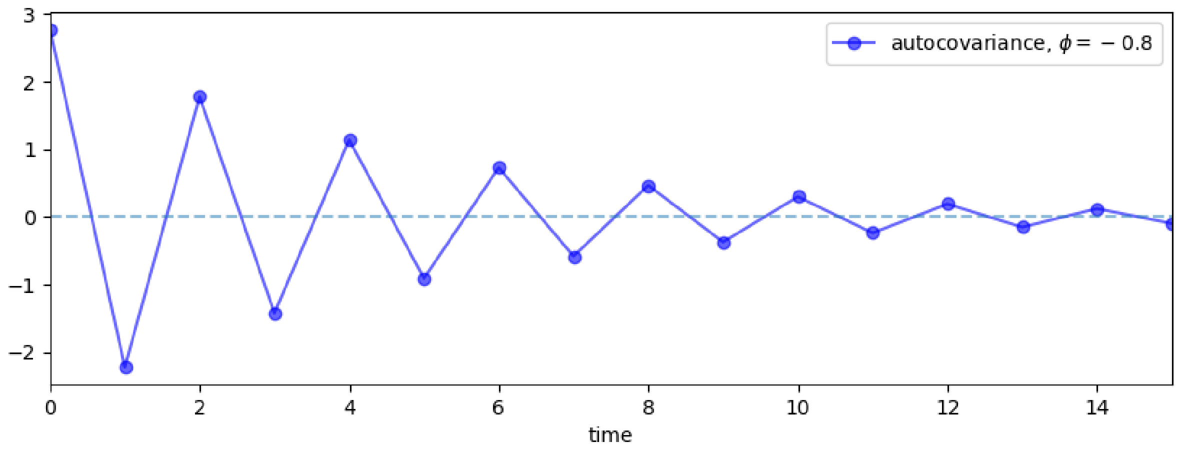
Applying (15.2) to the previous expression for X_t, we get the AR(1) autocovariance function

$$\gamma(k) = \phi^k \frac{\sigma^2}{1 - \phi^2}, \qquad k = 0, 1, \ldots \tag{15.4}$$

The next figure plots an example of this function for $\phi = 0.8$ and $\phi = -0.8$ with $\sigma = 1$.

```python
num_rows, num_cols = 2, 1
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10, 8))
plt.subplots_adjust(hspace=0.4)

for i, ϕ in enumerate((0.8, -0.8)):
    ax = axes[i]
    times = list(range(16))
    acov = [ϕ**k / (1 - ϕ**2) for k in times]
    ax.plot(times, acov, 'bo-', alpha=0.6,
            label=f'autocovariance, $\phi = {ϕ:.2}$')
    ax.legend(loc='upper right')
    ax.set(xlabel='time', xlim=(0, 15))
    ax.hlines(0, 0, 15, linestyle='--', alpha=0.5)
plt.show()
```





Another very simple process is the MA(1) process (here MA means "moving average")

$$X_t = \epsilon_t + \theta \epsilon_{t-1}$$

You will be able to verify that

$$\gamma(0) = \sigma^2 (1 + \theta^2), \quad \gamma(1) = \sigma^2 \theta, \quad \text{and} \quad \gamma(k) = 0 \quad \forall \, k > 1$$

The AR(1) can be generalized to an AR(p) and likewise for the MA(1).

Putting all of this together, we get the

15.2.6 ARMA Processes

A stochastic process $\{X_t\}$ is called an *autoregressive moving average process*, or ARMA(p, q), if it can be written as

$$X_t = \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \tag{15.5}$$

where $\{\epsilon_t\}$ is white noise.

An alternative notation for ARMA processes uses the *lag operator* L.

Def. Given arbitrary variable Y_t, let $L^k Y_t := Y_{t-k}$.

It turns out that

- lag operators facilitate succinct representations for linear stochastic processes
- algebraic manipulations that treat the lag operator as an ordinary scalar are legitimate

Using L, we can rewrite (15.5) as

$$L^0 X_t - \phi_1 L^1 X_t - \cdots - \phi_p L^p X_t = L^0 \epsilon_t + \theta_1 L^1 \epsilon_t + \cdots + \theta_q L^q \epsilon_t \tag{15.6}$$

If we let $\phi(z)$ and $\theta(z)$ be the polynomials

$$\phi(z) := 1 - \phi_1 z - \cdots - \phi_p z^p \quad \text{and} \quad \theta(z) := 1 + \theta_1 z + \cdots + \theta_q z^q \tag{15.7}$$

then (15.6) becomes

$$\phi(L) X_t = \theta(L) \epsilon_t \tag{15.8}$$

In what follows we **always assume** that the roots of the polynomial $\phi(z)$ lie outside the unit circle in the complex plane.

This condition is sufficient to guarantee that the ARMA(p, q) process is covariance stationary.

In fact, it implies that the process falls within the class of general linear processes *described above*.

That is, given an ARMA(p, q) process $\{X_t\}$ satisfying the unit circle condition, there exists a square summable sequence $\{\psi_t\}$ with $X_t = \sum_{j=0}^{\infty} \psi_j \epsilon_{t-j}$ for all t.

The sequence $\{\psi_t\}$ can be obtained by a recursive procedure outlined on page 79 of [Cryer and Chan, 2008].

The function $t \mapsto \psi_t$ is often called the *impulse response function*.

15.3 Spectral Analysis

Autocovariance functions provide a great deal of information about covariance stationary processes.

In fact, for zero-mean Gaussian processes, the autocovariance function characterizes the entire joint distribution.

Even for non-Gaussian processes, it provides a significant amount of information.

It turns out that there is an alternative representation of the autocovariance function of a covariance stationary process, called the *spectral density*.

At times, the spectral density is easier to derive, easier to manipulate, and provides additional intuition.

15.3.1 Complex Numbers

Before discussing the spectral density, we invite you to recall the main properties of complex numbers (or *skip to the next section*).

It can be helpful to remember that, in a formal sense, complex numbers are just points $(x, y) \in \mathbb{R}^2$ endowed with a specific notion of multiplication.

When (x, y) is regarded as a complex number, x is called the *real part* and y is called the *imaginary part*.

The *modulus* or *absolute value* of a complex number $z = (x, y)$ is just its Euclidean norm in $\mathbb{R}^2$, but is usually written as $|z|$ instead of $\|z\|$.

The product of two complex numbers (x, y) and (u, v) is defined to be $(xu - vy, xv + yu)$, while addition is standard pointwise vector addition.

When endowed with these notions of multiplication and addition, the set of complex numbers forms a field — addition and multiplication play well together, just as they do in $\mathbb{R}$.

The complex number (x, y) is often written as $x + iy$, where i is called the *imaginary unit* and is understood to obey $i^2 = -1$.

The $x + iy$ notation provides an easy way to remember the definition of multiplication given above, because, proceeding naively,

$$(x + iy)(u + iv) = xu - yv + i(xv + yu)$$

Converted back to our first notation, this becomes $(xu - vy, xv + yu)$ as promised.

Complex numbers can be represented in the polar form $re^{i\omega}$ where

$$re^{i\omega} := r(\cos(\omega) + i \sin(\omega)) = x + iy$$

where $x = r\cos(\omega), y = r\sin(\omega)$, and $\omega = \arctan(y/z)$ or $\tan(\omega) = y/x$.

15.3.2 Spectral Densities

Let $\{X_t\}$ be a covariance stationary process with autocovariance function γ satisfying $\sum_k \gamma(k)^2 < \infty$.

The *spectral density* f of $\{X_t\}$ is defined as the discrete time Fourier transform of its autocovariance function γ.

$$f(\omega) := \sum_{k \in \mathbb{Z}} \gamma(k) e^{-i\omega k}, \qquad \omega \in \mathbb{R}$$

(Some authors normalize the expression on the right by constants such as $1/\pi$ — the convention chosen makes little difference provided you are consistent).

Using the fact that γ is *even*, in the sense that $\gamma(t) = \gamma(-t)$ for all t, we can show that

$$f(\omega) = \gamma(0) + 2 \sum_{k \geq 1} \gamma(k) \cos(\omega k) \tag{15.9}$$

It is not difficult to confirm that f is

- real-valued
- even ($f(\omega) = f(-\omega)$), and
- 2π-periodic, in the sense that $f(2\pi + \omega) = f(\omega)$ for all ω

It follows that the values of f on $[0, \pi]$ determine the values of f on all of $\mathbb{R}$ — the proof is an exercise.

For this reason, it is standard to plot the spectral density only on the interval $[0, \pi]$.

15.3.3 Example 1: White Noise

Consider a white noise process $\{\epsilon_t\}$ with standard deviation σ.

It is easy to check that in this case $f(\omega) = \sigma^2$. So f is a constant function.

As we will see, this can be interpreted as meaning that “all frequencies are equally present”.

(White light has this property when frequency refers to the visible spectrum, a connection that provides the origins of the term “white noise”)

15.3.4 Example 2: AR and MA and ARMA

It is an exercise to show that the MA(1) process $X_t = \theta \epsilon_{t-1} + \epsilon_t$ has a spectral density

$$f(\omega) = \sigma^2 (1 + 2\theta \cos(\omega) + \theta^2) \tag{15.10}$$

With a bit more effort, it's possible to show (see, e.g., p. 261 of [Sargent, 1987]) that the spectral density of the AR(1) process $X_t = \phi X_{t-1} + \epsilon_t$ is

$$f(\omega) = \frac{\sigma^2}{1 - 2\phi \cos(\omega) + \phi^2} \tag{15.11}$$

More generally, it can be shown that the spectral density of the ARMA process (15.5) is

$$f(\omega) = \left| \frac{\theta(e^{i\omega})}{\phi(e^{i\omega})} \right|^2 \sigma^2 \tag{15.12}$$

where

- σ is the standard deviation of the white noise process $\{\epsilon_t\}$.
- the polynomials $\phi(\cdot)$ and $\theta(\cdot)$ are as defined in (15.7).

The derivation of (15.12) uses the fact that convolutions become products under Fourier transformations.

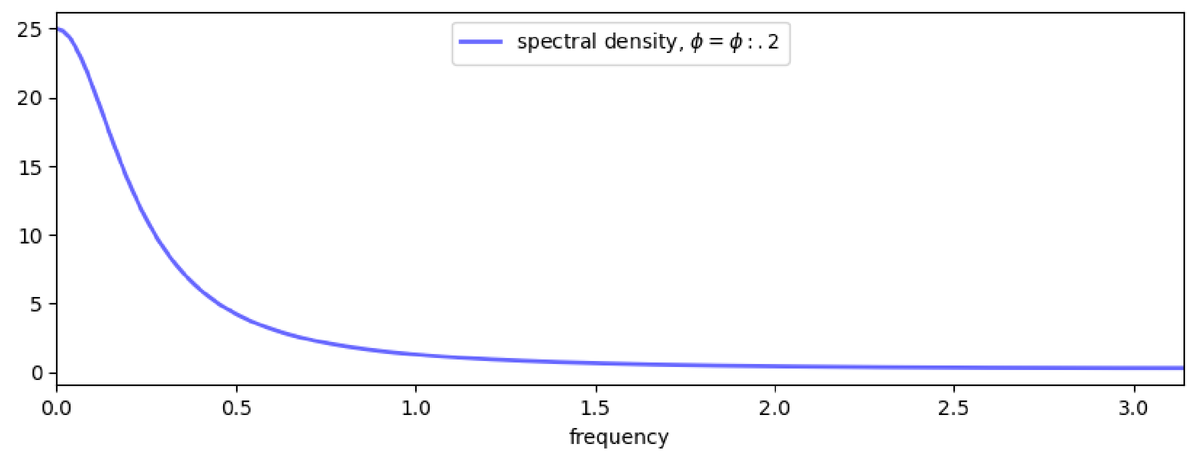
The proof is elegant and can be found in many places — see, for example, [Sargent, 1987], chapter 11, section 4.

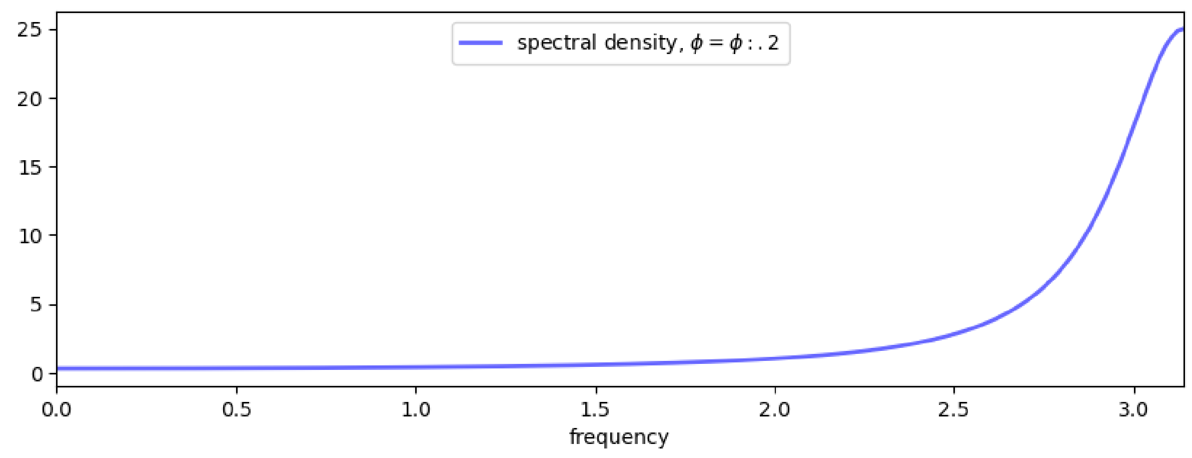
It's a nice exercise to verify that (15.10) and (15.11) are indeed special cases of (15.12).

15.3.5 Interpreting the Spectral Density

Plotting (15.11) reveals the shape of the spectral density for the AR(1) model when ϕ takes the values 0.8 and -0.8 respectively.

```python
def ar1_sd(ϕ, ω):
    return 1 / (1 - 2 * ϕ * np.cos(ω) + ϕ**2)

ωs = np.linspace(0, np.pi, 180)
num_rows, num_cols = 2, 1
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10, 8))
plt.subplots_adjust(hspace=0.4)

# Autocovariance when phi = 0.8
for i, ϕ in enumerate((0.8, -0.8)):
    ax = axes[i]
    sd = ar1_sd(ϕ, ωs)
    ax.plot(ωs, sd, 'b-', alpha=0.6, lw=2,
            label='spectral density, $\phi = {ϕ:.2}$')
    ax.legend(loc='upper center')
    ax.set(xlabel='frequency', xlim=(0, np.pi))
plt.show()
```





These spectral densities correspond to the autocovariance functions for the AR(1) process shown above.

Informally, we think of the spectral density as being large at those $\omega \in [0, \pi]$ at which the autocovariance function seems approximately to exhibit big damped cycles.

To see the idea, let's consider why, in the lower panel of the preceding figure, the spectral density for the case $\phi = -0.8$ is large at $\omega = \pi$.

Recall that the spectral density can be expressed as

$$f(\omega) = \gamma(0) + 2 \sum_{k \geq 1} \gamma(k) \cos(\omega k) = \gamma(0) + 2 \sum_{k \geq 1} (-0.8)^k \cos(\omega k) \tag{15.13}$$

When we evaluate this at $\omega = \pi$, we get a large number because $\cos(\pi k)$ is large and positive when $(-0.8)^k$ is positive, and large in absolute value and negative when $(-0.8)^k$ is negative.

Hence the product is always large and positive, and hence the sum of the products on the right-hand side of (15.13) is large.

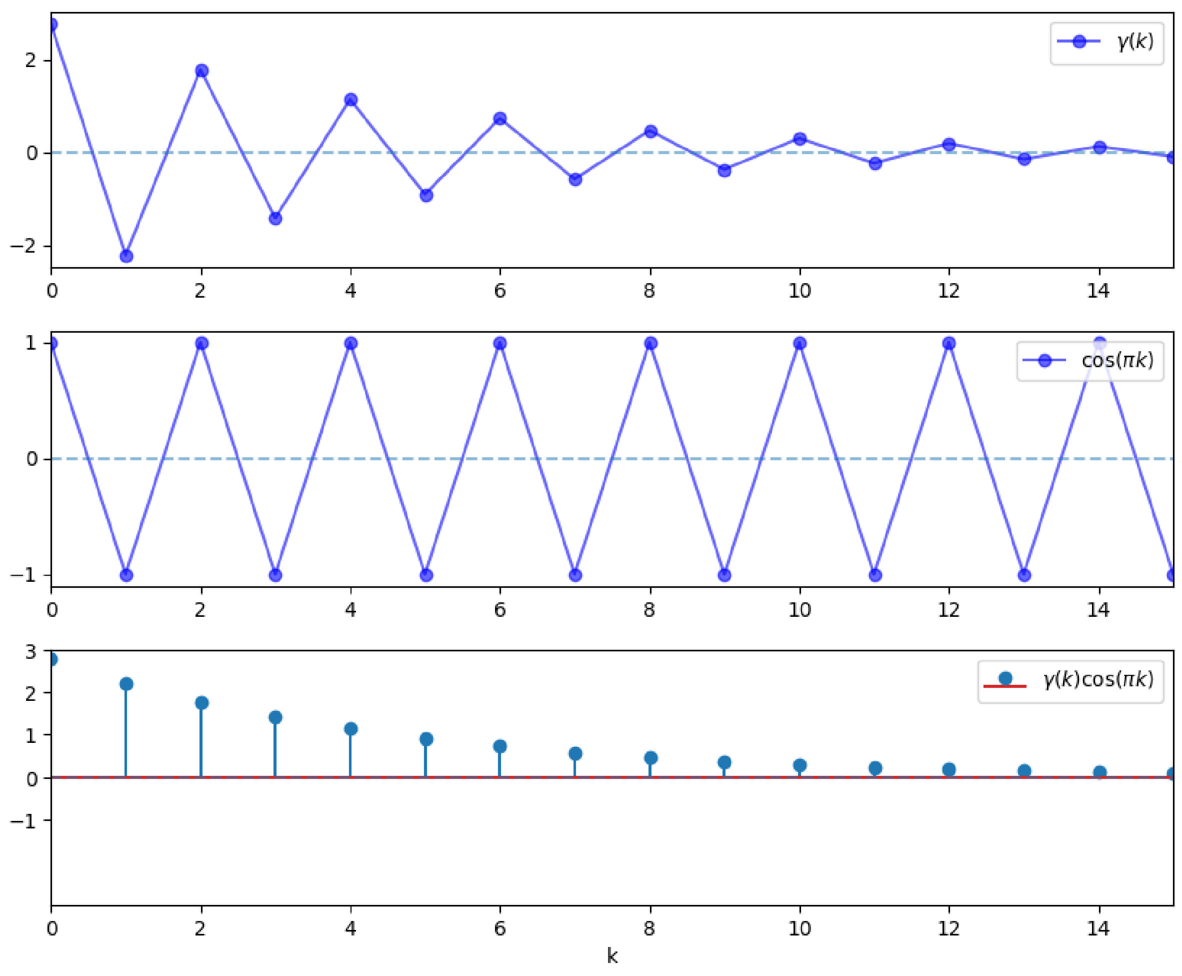
These ideas are illustrated in the next figure, which has k on the horizontal axis.

```
ϕ = -0.8
times = list(range(16))
y1 = [ϕ**k / (1 - ϕ**2) for k in times]
y2 = [np.cos(np.pi * k) for k in times]
```

(continues on next page)

(continued from previous page)

```
y3 = [a * b for a, b in zip(y1, y2)]

num_rows, num_cols = 3, 1
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10, 8))
plt.subplots_adjust(hspace=0.25)

# Autocovariance when ϕ = -0.8
ax = axes[0]
ax.plot(times, y1, 'bo-', alpha=0.6, label='$\gamma(k)$')
ax.legend(loc='upper right')
ax.set(xlim=(0, 15), yticks=(-2, 0, 2))
ax.hlines(0, 0, 15, linestyle='--', alpha=0.5)

# Cycles at frequency π
ax = axes[1]
ax.plot(times, y2, 'bo-', alpha=0.6, label='$\cos(\pi k)$')
ax.legend(loc='upper right')
ax.set(xlim=(0, 15), yticks=(-1, 0, 1))
ax.hlines(0, 0, 15, linestyle='--', alpha=0.5)

# Product
ax = axes[2]
ax.stem(times, y3, label='$\gamma(k) \cos(\pi k)$')
ax.legend(loc='upper right')
ax.set(xlim=(0, 15), ylim=(-3, 3), yticks=(-1, 0, 1, 2, 3))
ax.hlines(0, 0, 15, linestyle='--', alpha=0.5)
ax.set_xlabel("k")

plt.show()
```



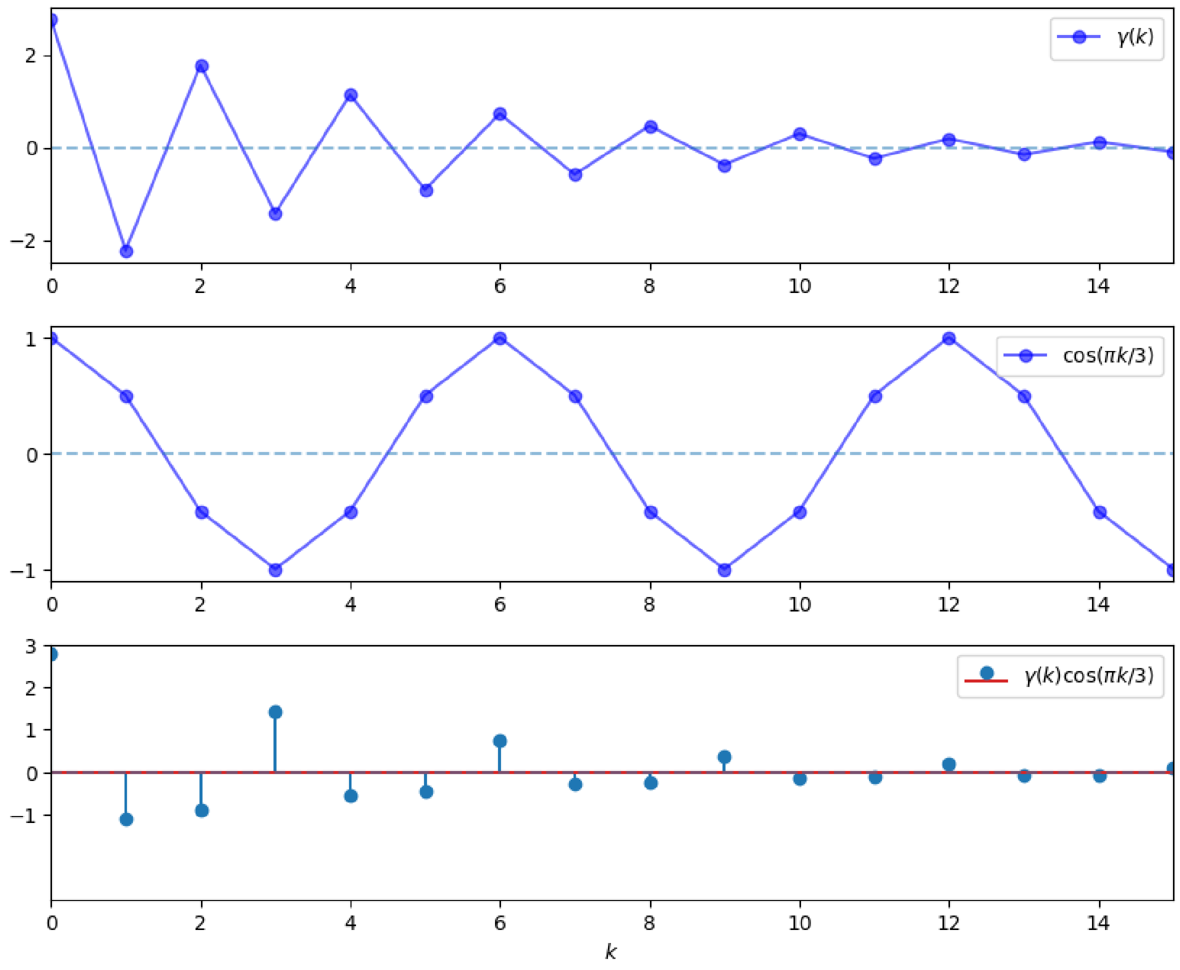
On the other hand, if we evaluate $f(\omega)$ at $\omega = \pi/3$, then the cycles are not matched, the sequence $\gamma(k)\cos(\omega k)$ contains both positive and negative terms, and hence the sum of these terms is much smaller.

```
ϕ = -0.8
times = list(range(16))
y1 = [ϕ**k / (1 - ϕ**2) for k in times]
y2 = [np.cos(np.pi * k/3) for k in times]
y3 = [a * b for a, b in zip(y1, y2)]

num_rows, num_cols = 3, 1
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10, 8))
plt.subplots_adjust(hspace=0.25)

# Autocovariance when phi = -0.8
ax = axes[0]
ax.plot(times, y1, 'bo-', alpha=0.6, label='$\gamma(k)$')
ax.legend(loc='upper right')
ax.set(xlim=(0, 15), yticks=(-2, 0, 2))
ax.hlines(0, 0, 15, linestyle='--', alpha=0.5)

# Cycles at frequency π
ax = axes[1]
ax.plot(times, y2, 'bo-', alpha=0.6, label='$\cos(\pi k/3)$')
```

(continues on next page)

(continued from previous page)

```
ax.legend(loc='upper right')
ax.set(xlim=(0, 15), yticks=(-1, 0, 1))
ax.hlines(0, 0, 15, linestyle='--', alpha=0.5)

# Product
ax = axes[2]
ax.stem(times, y3, label='$\gamma(k) \cos(\pi k/3)$')
ax.legend(loc='upper right')
ax.set(xlim=(0, 15), ylim=(-3, 3), yticks=(-1, 0, 1, 2, 3))
ax.hlines(0, 0, 15, linestyle='--', alpha=0.5)
ax.set_xlabel("$k$")

plt.show()
```



In summary, the spectral density is large at frequencies ω where the autocovariance function exhibits damped cycles.

15.3.6 Inverting the Transformation

We have just seen that the spectral density is useful in the sense that it provides a frequency-based perspective on the autocovariance structure of a covariance stationary process.

Another reason that the spectral density is useful is that it can be "inverted" to recover the autocovariance function via the *inverse Fourier transform*.

In particular, for all $k \in \mathbb{Z}$, we have

$$\gamma(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\omega) e^{i\omega k} d\omega \tag{15.14}$$

This is convenient in situations where the spectral density is easier to calculate and manipulate than the autocovariance function.

(For example, the expression (15.12) for the ARMA spectral density is much easier to work with than the expression for the ARMA autocovariance)

15.3.7 Mathematical Theory

This section is loosely based on [Sargent, 1987], p. 249-253, and included for those who

- would like a bit more insight into spectral densities
- and have at least some background in Hilbert space theory

Others should feel free to skip to the *next section* — none of this material is necessary to progress to computation.

Recall that every separable Hilbert space H has a countable orthonormal basis $\{h_k\}$.

The nice thing about such a basis is that every $f \in H$ satisfies

$$f = \sum_k \alpha_k h_k \quad \text{where} \quad \alpha_k := \langle f, h_k \rangle \tag{15.15}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in H.

Thus, f can be represented to any degree of precision by linearly combining basis vectors.

The scalar sequence $\alpha = \{\alpha_k\}$ is called the *Fourier coefficients* of f, and satisfies $\sum_k |\alpha_k|^2 < \infty$.

In other words, α is in ℓ_2, the set of square summable sequences.

Consider an operator T that maps $\alpha \in \ell_2$ into its expansion $\sum_k \alpha_k h_k \in H$.

The Fourier coefficients of $T\alpha$ are just $\alpha = \{\alpha_k\}$, as you can verify by confirming that $\langle T\alpha, h_k \rangle = \alpha_k$.

Using elementary results from Hilbert space theory, it can be shown that

- T is one-to-one — if α and β are distinct in ℓ_2, then so are their expansions in H.
- T is onto — if $f \in H$ then its preimage in ℓ_2 is the sequence α given by $\alpha_k = \langle f, h_k \rangle$.
- T is a linear isometry — in particular, $\langle \alpha, \beta \rangle = \langle T\alpha, T\beta \rangle$.

Summarizing these results, we say that any separable Hilbert space is isometrically isomorphic to ℓ_2.

In essence, this says that each separable Hilbert space we consider is just a different way of looking at the fundamental space ℓ_2.

With this in mind, let's specialize to a setting where

- $\gamma \in \ell_2$ is the autocovariance function of a covariance stationary process, and f is the spectral density.

- $H = L_2$, where L_2 is the set of square summable functions on the interval $[-\pi, \pi]$, with inner product $\langle g, h \rangle = \int_{-\pi}^{\pi} g(\omega) h(\omega) d\omega$.
- $\{h_k\}$ = the orthonormal basis for L_2 given by the set of trigonometric functions.

$$h_k(\omega) = \frac{e^{i\omega k}}{\sqrt{2\pi}}, \quad k \in \mathbb{Z}, \quad \omega \in [-\pi, \pi]$$

Using the definition of T from above and the fact that f is even, we now have

$$T\gamma = \sum_{k \in \mathbb{Z}} \gamma(k) \frac{e^{i\omega k}}{\sqrt{2\pi}} = \frac{1}{\sqrt{2\pi}} f(\omega) \tag{15.16}$$

In other words, apart from a scalar multiple, the spectral density is just a transformation of $\gamma \in \ell_2$ under a certain linear isometry — a different way to view γ.

In particular, it is an expansion of the autocovariance function with respect to the trigonometric basis functions in L_2.

As discussed above, the Fourier coefficients of $T\gamma$ are given by the sequence γ, and, in particular, $\gamma(k) = \langle T\gamma, h_k \rangle$.

Transforming this inner product into its integral expression and using (15.16) gives (15.14), justifying our earlier expression for the inverse transform.

15.4 Implementation

Most code for working with covariance stationary models deals with ARMA models.

Python code for studying ARMA models can be found in the `tsa` submodule of statsmodels.

Since this code doesn't quite cover our needs — particularly vis-a-vis spectral analysis — we've put together the module arma.py, which is part of QuantEcon.py package.

The module provides functions for mapping ARMA(p, q) models into their

1. impulse response function
2. simulated time series
3. autocovariance function
4. spectral density

15.4.1 Application

Let's use this code to replicate the plots on pages 68–69 of [Ljungqvist and Sargent, 2018].

Here are some functions to generate the plots

```python
def plot_impulse_response(arma, ax=None):
    if ax is None:
        ax = plt.gca()
    yi = arma.impulse_response()
    ax.stem(list(range(len(yi))), yi)
    ax.set(xlim=(-0.5), ylim=(min(yi)-0.1, max(yi)+0.1),
                title='Impulse response', xlabel='time', ylabel='response')
    return ax

def plot_spectral_density(arma, ax=None):
```

(continues on next page)

(continued from previous page)

```
    if ax is None:
        ax = plt.gca()
    w, spect = arma.spectral_density(two_pi=False)
    ax.semilogy(w, spect)
    ax.set(xlim=(0, np.pi), ylim=(0, np.max(spect)),
           title='Spectral density', xlabel='frequency', ylabel='spectrum')
    return ax

def plot_autocovariance(arma, ax=None):
    if ax is None:
        ax = plt.gca()
    acov = arma.autocovariance()
    ax.stem(list(range(len(acov))), acov)
    ax.set(xlim=(-0.5, len(acov) - 0.5), title='Autocovariance',
           xlabel='time', ylabel='autocovariance')
    return ax

def plot_simulation(arma, ax=None):
    if ax is None:
        ax = plt.gca()
    x_out = arma.simulation()
    ax.plot(x_out)
    ax.set(title='Sample path', xlabel='time', ylabel='state space')
    return ax

def quad_plot(arma):
    """
    Plots the impulse response, spectral_density, autocovariance,
    and one realization of the process.

    """
    num_rows, num_cols = 2, 2
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(10, 7))
    plot_functions = [plot_impulse_response,
                      plot_spectral_density,
                      plot_autocovariance,
                      plot_simulation]
    for plot_func, ax in zip(plot_functions, axes.flatten()):
        plot_func(arma, ax)
    plt.tight_layout()
    plt.show()
```

Now let's call these functions to generate plots.

As a warmup, let's make sure things look right when we for the pure white noise model $X_t = \epsilon_t$.

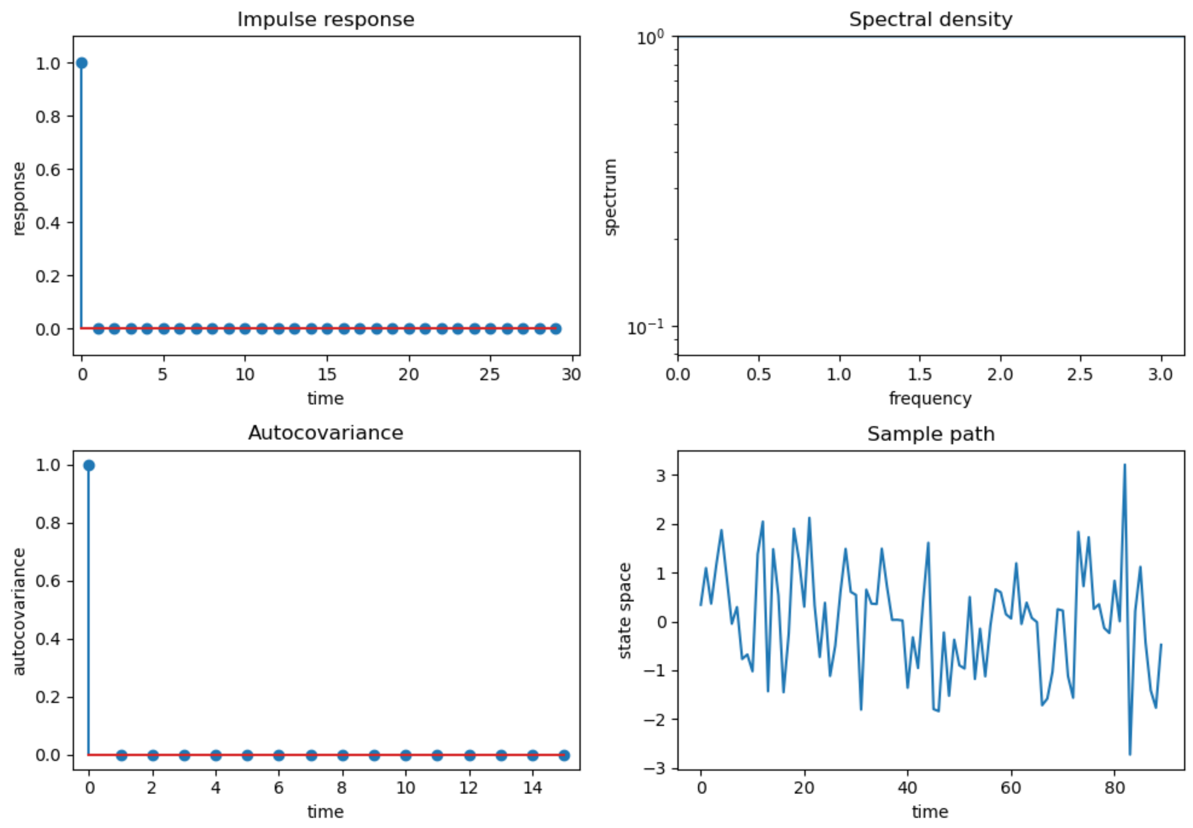
```
ϕ = 0.0
θ = 0.0
arma = qe.ARMA(ϕ, θ)
quad_plot(arma)
```

```
/home/runner/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪cbook.py:1699: ComplexWarning: Casting complex values to real discards the␣
↪imaginary part
  return math.isfinite(val)
```

(continues on next page)

(continued from previous page)

```
/home/runner/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪cbook.py:1345: ComplexWarning: Casting complex values to real discards the␣
↪imaginary part
  return np.asarray(x, float)
/tmp/ipykernel_2005/4271821819.py:15: UserWarning: Attempt to set non-positive␣
↪ylim on a log-scaled axis will be ignored.
  ax.set(xlim=(0, np.pi), ylim=(0, np.max(spect)),
/home/runner/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪transforms.py:992: ComplexWarning: Casting complex values to real discards the␣
↪imaginary part
  self._points[:, 1] = interval
```



If we look carefully, things look good: the spectrum is the flat line at 10^0 at the very top of the spectrum graphs, which is at it should be.

Also

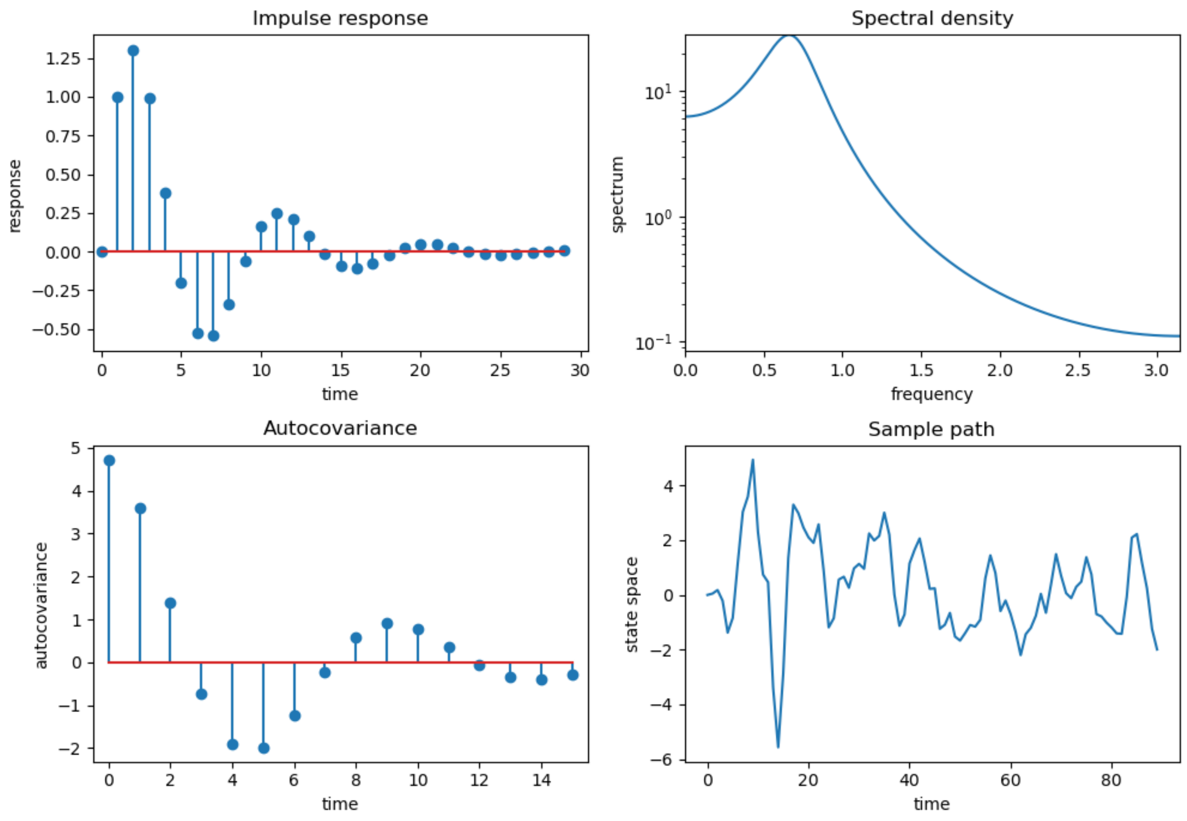
- the variance equals $1 = \frac{1}{2\pi}\int_{-\pi}^{\pi} 1 d\omega$ as it should.
- the covariogram and impulse response look as they should.
- it is actually challenging to visualize a time series realization of white noise – a sequence of surprises – but this too looks pretty good.

To get some more examples, as our laboratory we'll replicate quartets of graphs that [Ljungqvist and Sargent, 2018] use to teach "how to read spectral densities".

Ljunqvist and Sargent's first model is $X_t = 1.3X_{t-1} - .7X_{t-2} + \epsilon_t$

```
ϕ = 1.3, -.7
θ = 0.0
arma = qe.ARMA(ϕ, θ)
quad_plot(arma)
```

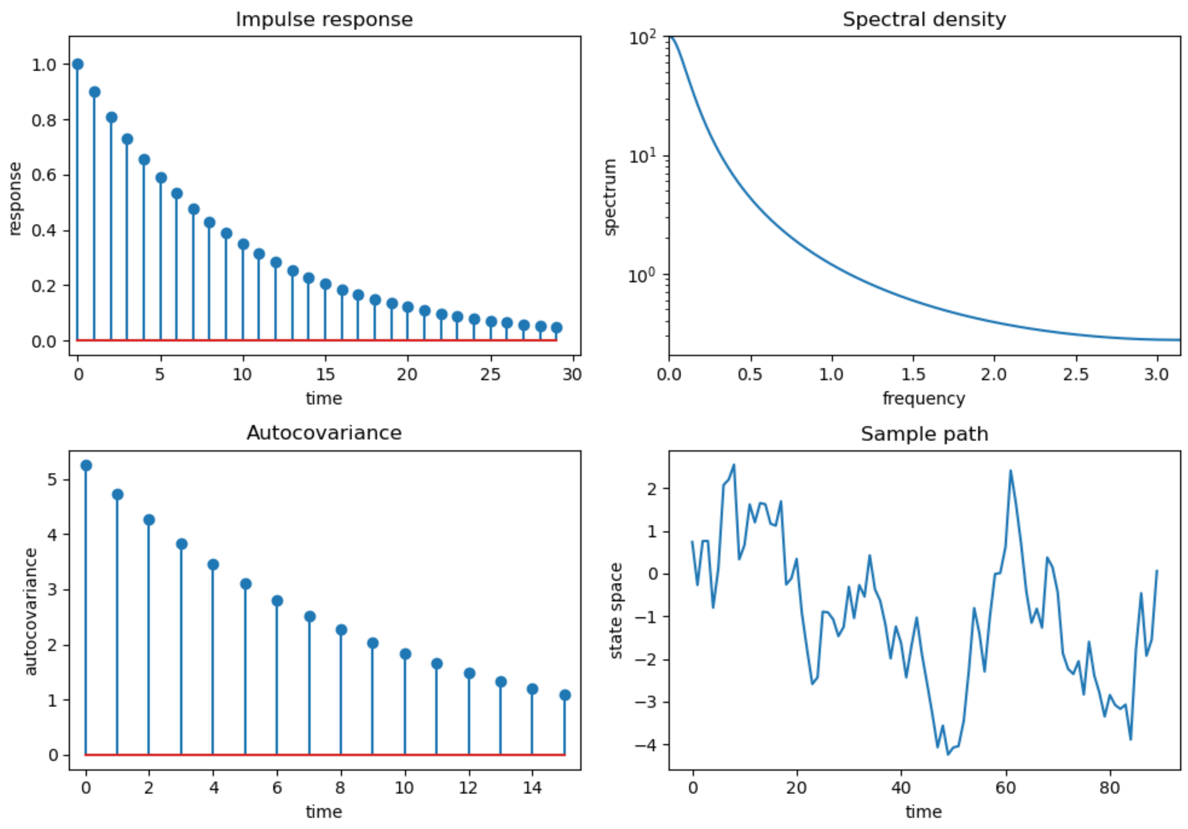
```
/tmp/ipykernel_2005/4271821819.py:15: UserWarning: Attempt to set non-positive␣
↪ylim on a log-scaled axis will be ignored.
  ax.set(xlim=(0, np.pi), ylim=(0, np.max(spect)),
```



Ljungqvist and Sargent's second model is $X_t = .9X_{t-1} + \epsilon_t$

```
ϕ = 0.9
θ = -0.0
arma = qe.ARMA(ϕ, θ)
quad_plot(arma)
```

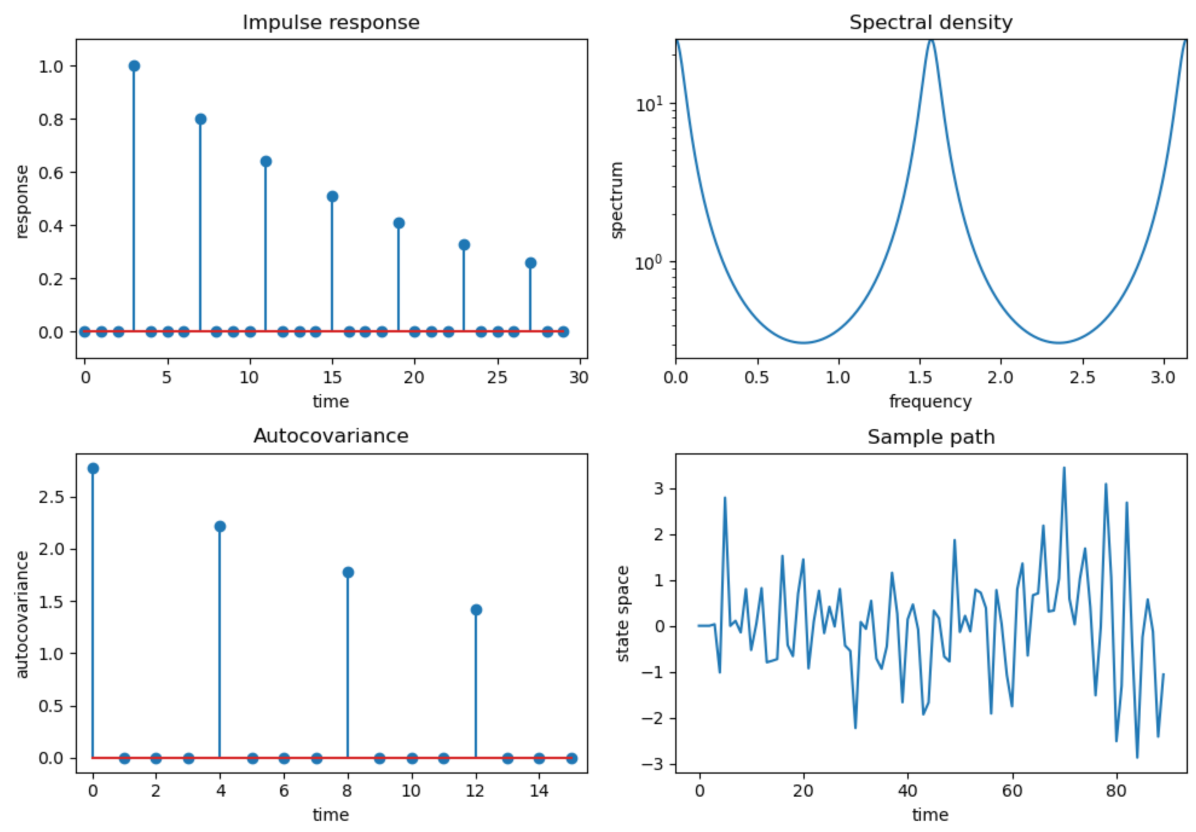
```
/tmp/ipykernel_2005/4271821819.py:15: UserWarning: Attempt to set non-positive␣
↪ylim on a log-scaled axis will be ignored.
  ax.set(xlim=(0, np.pi), ylim=(0, np.max(spect)),
```



Ljungqvist and Sargent's third model is $X_t = .8X_{t-4} + \epsilon_t$

```
ϕ = 0., 0., 0., .8
θ = -0.0
arma = qe.ARMA(ϕ, θ)
quad_plot(arma)
```

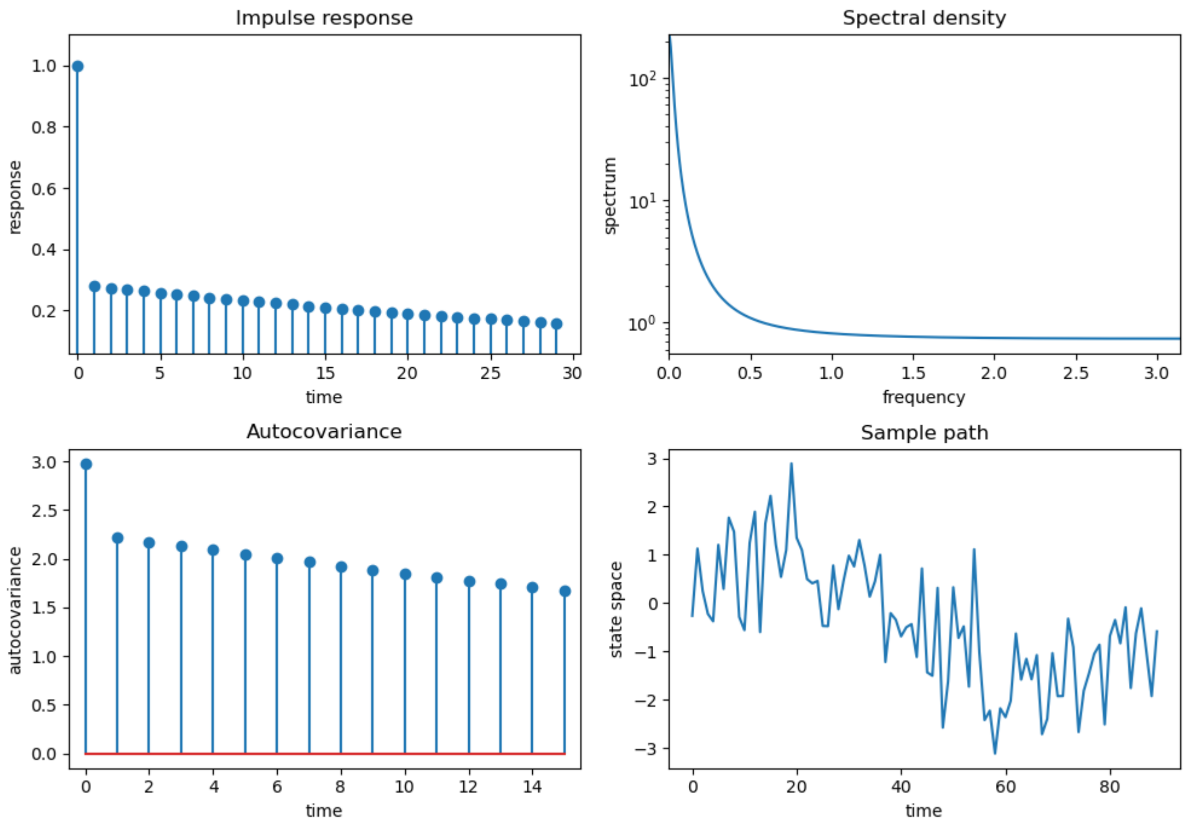
```
/tmp/ipykernel_2005/4271821819.py:15: UserWarning: Attempt to set non-positive␣
↪ylim on a log-scaled axis will be ignored.
  ax.set(xlim=(0, np.pi), ylim=(0, np.max(spect)),
```



Ljungqvist and Sargent's fourth model is $X_t = .98X_{t-1} + \epsilon_t - .7\epsilon_{t-1}$

```
ϕ = .98
θ = -0.7
arma = qe.ARMA(ϕ, θ)
quad_plot(arma)
```

```
/tmp/ipykernel_2005/4271821819.py:15: UserWarning: Attempt to set non-positive␣
↪ylim on a log-scaled axis will be ignored.
  ax.set(xlim=(0, np.pi), ylim=(0, np.max(spect)),
```



15.4.2 Explanation

The call

```
arma = ARMA(ϕ, θ, σ)
```

creates an instance `arma` that represents the ARMA(p, q) model

$$X_t = \phi_1 X_{t-1} + ... + \phi_p X_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q}$$

If `ϕ` and `θ` are arrays or sequences, then the interpretation will be

- `ϕ` holds the vector of parameters $(\phi_1, \phi_2, ..., \phi_p)$.
- `θ` holds the vector of parameters $(\theta_1, \theta_2, ..., \theta_q)$.

The parameter `σ` is always a scalar, the standard deviation of the white noise.

We also permit `ϕ` and `θ` to be scalars, in which case the model will be interpreted as

$$X_t = \phi X_{t-1} + \epsilon_t + \theta \epsilon_{t-1}$$

The two numerical packages most useful for working with ARMA models are `scipy.signal` and `numpy.fft`.

The package `scipy.signal` expects the parameters to be passed into its functions in a manner consistent with the alternative ARMA notation (15.8).

For example, the impulse response sequence $\{\psi_t\}$ discussed above can be obtained using `scipy.signal.dimpulse`, and the function call should be of the form

```
times, ψ = dimpulse((ma_poly, ar_poly, 1), n=impulse_length)
```

where `ma_poly` and `ar_poly` correspond to the polynomials in (15.7) — that is,

- `ma_poly` is the vector $(1, \theta_1, \theta_2, \ldots, \theta_q)$
- `ar_poly` is the vector $(1, -\phi_1, -\phi_2, \ldots, -\phi_p)$

To this end, we also maintain the arrays `ma_poly` and `ar_poly` as instance data, with their values computed automatically from the values of `phi` and `theta` supplied by the user.

If the user decides to change the value of either `theta` or `phi` ex-post by assignments such as `arma.phi = (0.5, 0.2)` or `arma.theta = (0, -0.1)`.

then `ma_poly` and `ar_poly` should update automatically to reflect these new parameters.

This is achieved in our implementation by using descriptors.

15.4.3 Computing the Autocovariance Function

As discussed above, for ARMA processes the spectral density has a *simple representation* that is relatively easy to calculate.

Given this fact, the easiest way to obtain the autocovariance function is to recover it from the spectral density via the inverse Fourier transform.

Here we use NumPy's Fourier transform package np.fft, which wraps a standard Fortran-based package called FFTPACK.

A look at the np.fft documentation shows that the inverse transform np.fft.ifft takes a given sequence $A_0, A_1, \ldots, A_{n-1}$ and returns the sequence $a_0, a_1, \ldots, a_{n-1}$ defined by

$$a_k = \frac{1}{n} \sum_{t=0}^{n-1} A_t e^{ik2\pi t/n}$$

Thus, if we set $A_t = f(\omega_t)$, where f is the spectral density and $\omega_t := 2\pi t/n$, then

$$a_k = \frac{1}{n} \sum_{t=0}^{n-1} f(\omega_t) e^{i\omega_t k} = \frac{1}{2\pi} \frac{2\pi}{n} \sum_{t=0}^{n-1} f(\omega_t) e^{i\omega_t k}, \qquad \omega_t := 2\pi t/n$$

For n sufficiently large, we then have

$$a_k \approx \frac{1}{2\pi} \int_0^{2\pi} f(\omega) e^{i\omega k} d\omega = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\omega) e^{i\omega k} d\omega$$

(You can check the last equality)

In view of (15.14), we have now shown that, for n sufficiently large, $a_k \approx \gamma(k)$ — which is exactly what we want to compute.

CHAPTER

SIXTEEN

ESTIMATION OF SPECTRA

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
```

16.1 Overview

In a *previous lecture*, we covered some fundamental properties of covariance stationary linear stochastic processes.

One objective for that lecture was to introduce spectral densities — a standard and very useful technique for analyzing such processes.

In this lecture, we turn to the problem of estimating spectral densities and other related quantities from data.

Estimates of the spectral density are computed using what is known as a periodogram — which in turn is computed via the famous fast Fourier transform.

Once the basic technique has been explained, we will apply it to the analysis of several key macroeconomic time series.

For supplementary reading, see [Sargent, 1987] or [Cryer and Chan, 2008].

Let's start with some standard imports:

```
import numpy as np
import matplotlib.pyplot as plt
from quantecon import ARMA, periodogram, ar_periodogram
```

16.2 Periodograms

Recall that the spectral density f of a covariance stationary process with autocorrelation function γ can be written

$$f(\omega) = \gamma(0) + 2\sum_{k\geq 1}\gamma(k)\cos(\omega k), \qquad \omega \in \mathbb{R}$$

Now consider the problem of estimating the spectral density of a given time series, when γ is unknown.

In particular, let $X_0, \dots, X_{n-1}$ be n consecutive observations of a single time series that is assumed to be covariance stationary.

The most common estimator of the spectral density of this process is the *periodogram* of $X_0, \dots, X_{n-1}$, which is defined as

$$I(\omega) := \frac{1}{n} \left| \sum_{t=0}^{n-1} X_t e^{it\omega} \right|^2, \qquad \omega \in \mathbb{R} \tag{16.1}$$

(Recall that $|z|$ denotes the modulus of complex number z)

Alternatively, $I(\omega)$ can be expressed as

$$I(\omega) = \frac{1}{n} \left\{ \left[\sum_{t=0}^{n-1} X_t \cos(\omega t) \right]^2 + \left[\sum_{t=0}^{n-1} X_t \sin(\omega t) \right]^2 \right\}$$

It is straightforward to show that the function I is even and 2π-periodic (i.e., $I(\omega) = I(-\omega)$ and $I(\omega + 2\pi) = I(\omega)$ for all $\omega \in \mathbb{R}$).

From these two results, you will be able to verify that the values of I on $[0, \pi]$ determine the values of I on all of $\mathbb{R}$.

The next section helps to explain the connection between the periodogram and the spectral density.

16.2.1 Interpretation

To interpret the periodogram, it is convenient to focus on its values at the *Fourier frequencies*

$$\omega_j := \frac{2\pi j}{n}, \quad j = 0, \dots, n-1$$

In what sense is $I(\omega_j)$ an estimate of $f(\omega_j)$?

The answer is straightforward, although it does involve some algebra.

With a bit of effort, one can show that for any integer $j > 0$,

$$\sum_{t=0}^{n-1} e^{it\omega_j} = \sum_{t=0}^{n-1} \exp\left\{ i2\pi j \frac{t}{n} \right\} = 0$$

Letting $\bar{X}$ denote the sample mean $n^{-1} \sum_{t=0}^{n-1} X_t$, we then have

$$nI(\omega_j) = \left| \sum_{t=0}^{n-1} (X_t - \bar{X}) e^{it\omega_j} \right|^2 = \sum_{t=0}^{n-1} (X_t - \bar{X}) e^{it\omega_j} \sum_{r=0}^{n-1} (X_r - \bar{X}) e^{-ir\omega_j}$$

By carefully working through the sums, one can transform this to

$$nI(\omega_j) = \sum_{t=0}^{n-1} (X_t - \bar{X})^2 + 2 \sum_{k=1}^{n-1} \sum_{t=k}^{n-1} (X_t - \bar{X})(X_{t-k} - \bar{X}) \cos(\omega_j k)$$

Now let

$$\hat{\gamma}(k) := \frac{1}{n} \sum_{t=k}^{n-1} (X_t - \bar{X})(X_{t-k} - \bar{X}), \qquad k = 0, 1, \dots, n-1$$

This is the sample autocovariance function, the natural "plug-in estimator" of the *autocovariance function* γ.

("Plug-in estimator" is an informal term for an estimator found by replacing expectations with sample means)

With this notation, we can now write

$$I(\omega_j) = \hat{\gamma}(0) + 2 \sum_{k=1}^{n-1} \hat{\gamma}(k) \cos(\omega_j k)$$

Recalling our expression for f given *above*, we see that $I(\omega_j)$ is just a sample analog of $f(\omega_j)$.

16.2.2 Calculation

Let's now consider how to compute the periodogram as defined in (16.1).

There are already functions available that will do this for us — an example is `statsmodels.tsa.stattools.periodogram` in the `statsmodels` package.

However, it is very simple to replicate their results, and this will give us a platform to make useful extensions.

The most common way to calculate the periodogram is via the discrete Fourier transform, which in turn is implemented through the fast Fourier transform algorithm.

In general, given a sequence $a_0, \dots, a_{n-1}$, the discrete Fourier transform computes the sequence

$$A_j := \sum_{t=0}^{n-1} a_t \exp \left\{ i 2 \pi \frac{tj}{n} \right\}, \qquad j = 0, \dots, n-1$$

With `numpy.fft.fft` imported as `fft` and $a_0, \dots, a_{n-1}$ stored in NumPy array `a`, the function call `fft(a)` returns the values $A_0, \dots, A_{n-1}$ as a NumPy array.

It follows that when the data $X_0, \dots, X_{n-1}$ are stored in array `X`, the values $I(\omega_j)$ at the Fourier frequencies, which are given by

$$\frac{1}{n} \left| \sum_{t=0}^{n-1} X_t \exp \left\{ i 2 \pi \frac{tj}{n} \right\} \right|^2, \qquad j = 0, \dots, n-1$$

can be computed by `np.abs(fft(X))**2 / len(X)`.

Note: The NumPy function `abs` acts elementwise, and correctly handles complex numbers (by computing their modulus, which is exactly what we need).

A function called `periodogram` that puts all this together can be found here.

Let's generate some data for this function using the `ARMA` class from QuantEcon.py (see the *lecture on linear processes* for more details).

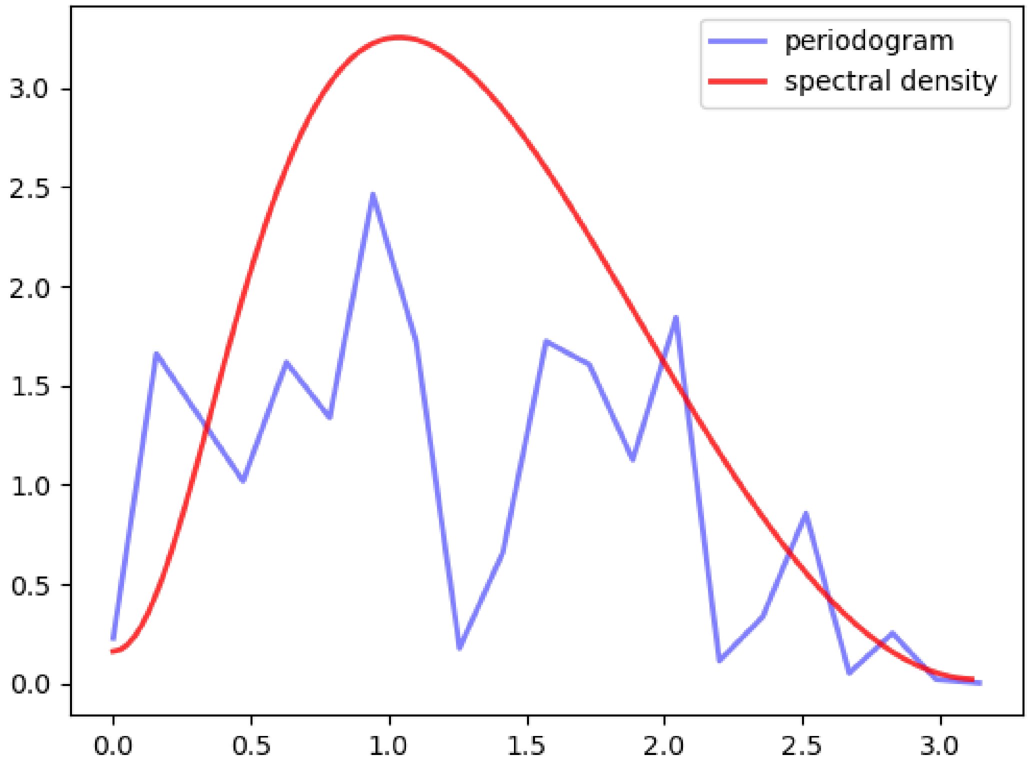
Here's a code snippet that, once the preceding code has been run, generates data from the process

$$X_t = 0.5 X_{t-1} + \epsilon_t - 0.8 \epsilon_{t-2} \tag{16.2}$$

where $\{\epsilon_t\}$ is white noise with unit variance, and compares the periodogram to the actual spectral density

```
n = 40                              # Data size
ϕ, θ = 0.5, (0, -0.8)               # AR and MA parameters
lp = ARMA(ϕ, θ)
X = lp.simulation(ts_length=n)

fig, ax = plt.subplots()
x, y = periodogram(X)
ax.plot(x, y, 'b-', lw=2, alpha=0.5, label='periodogram')
x_sd, y_sd = lp.spectral_density(two_pi=False, res=120)
ax.plot(x_sd, y_sd, 'r-', lw=2, alpha=0.8, label='spectral density')
ax.legend()
plt.show()
```

```
/home/runner/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪cbook.py:1699: ComplexWarning: Casting complex values to real discards the␣
↪imaginary part
  return math.isfinite(val)
/home/runner/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪cbook.py:1345: ComplexWarning: Casting complex values to real discards the␣
↪imaginary part
  return np.asarray(x, float)
```



This estimate looks rather disappointing, but the data size is only 40, so perhaps it's not surprising that the estimate is poor.

However, if we try again with `n = 1200` the outcome is not much better

The periodogram is far too irregular relative to the underlying spectral density.

This brings us to our next topic.

16.3 Smoothing

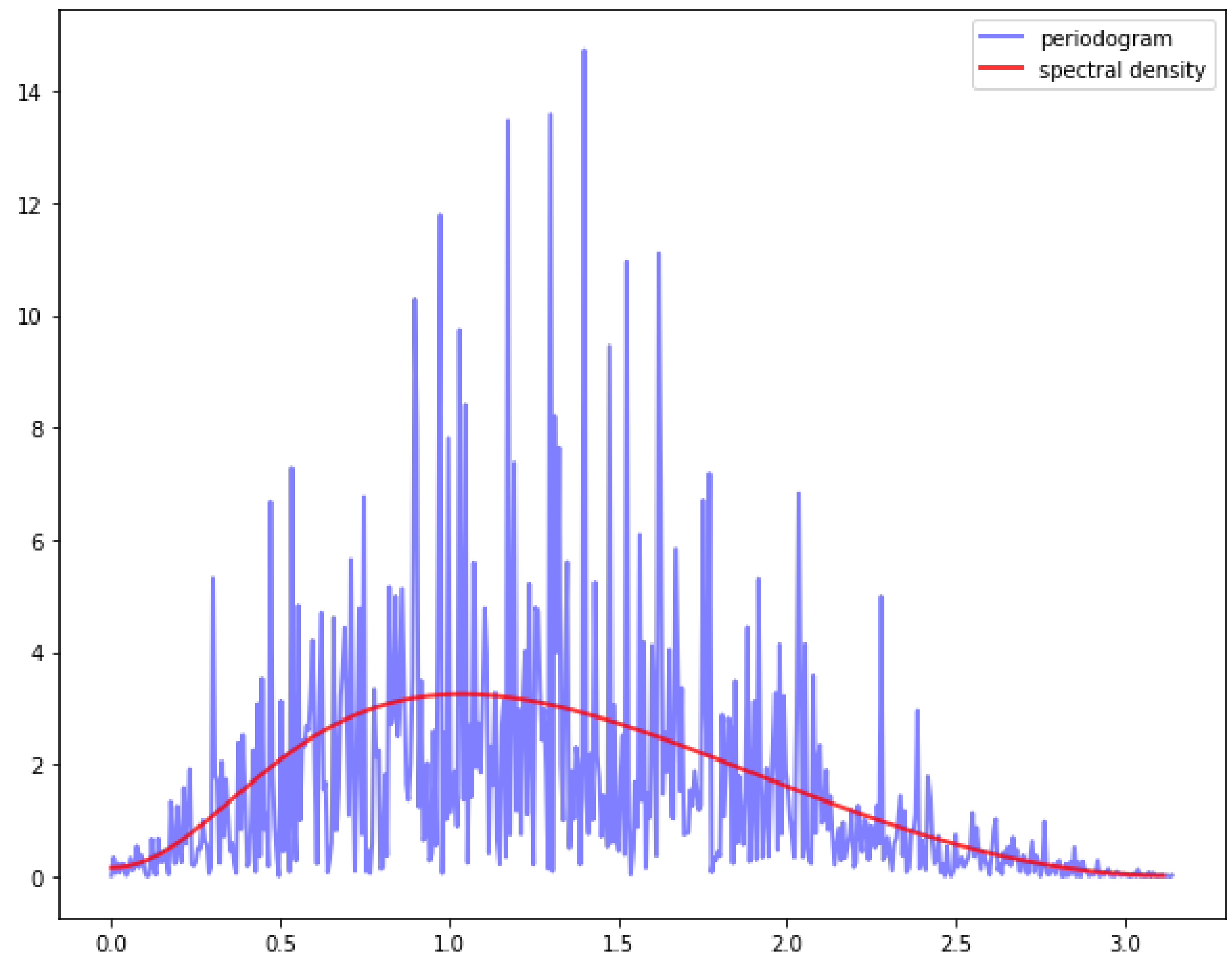
There are two related issues here.

One is that, given the way the fast Fourier transform is implemented, the number of points ω at which $I(\omega)$ is estimated increases in line with the amount of data.

In other words, although we have more data, we are also using it to estimate more values.

A second issue is that densities of all types are fundamentally hard to estimate without parametric assumptions.

Typically, nonparametric estimation of densities requires some degree of smoothing.


periodogram
spectral density
0
2
4
6
8
10
12
14
0.0
0.5
1.0
1.5
2.0
2.5
3.0

The standard way that smoothing is applied to periodograms is by taking local averages.

In other words, the value $I(\omega_j)$ is replaced with a weighted average of the adjacent values

$$I(\omega_{j-p}), I(\omega_{j-p+1}), \ldots, I(\omega_j), \ldots, I(\omega_{j+p})$$

This weighted average can be written as

$$I_S(\omega_j) := \sum_{\ell=-p}^{p} w(\ell) I(\omega_{j+\ell}) \tag{16.3}$$

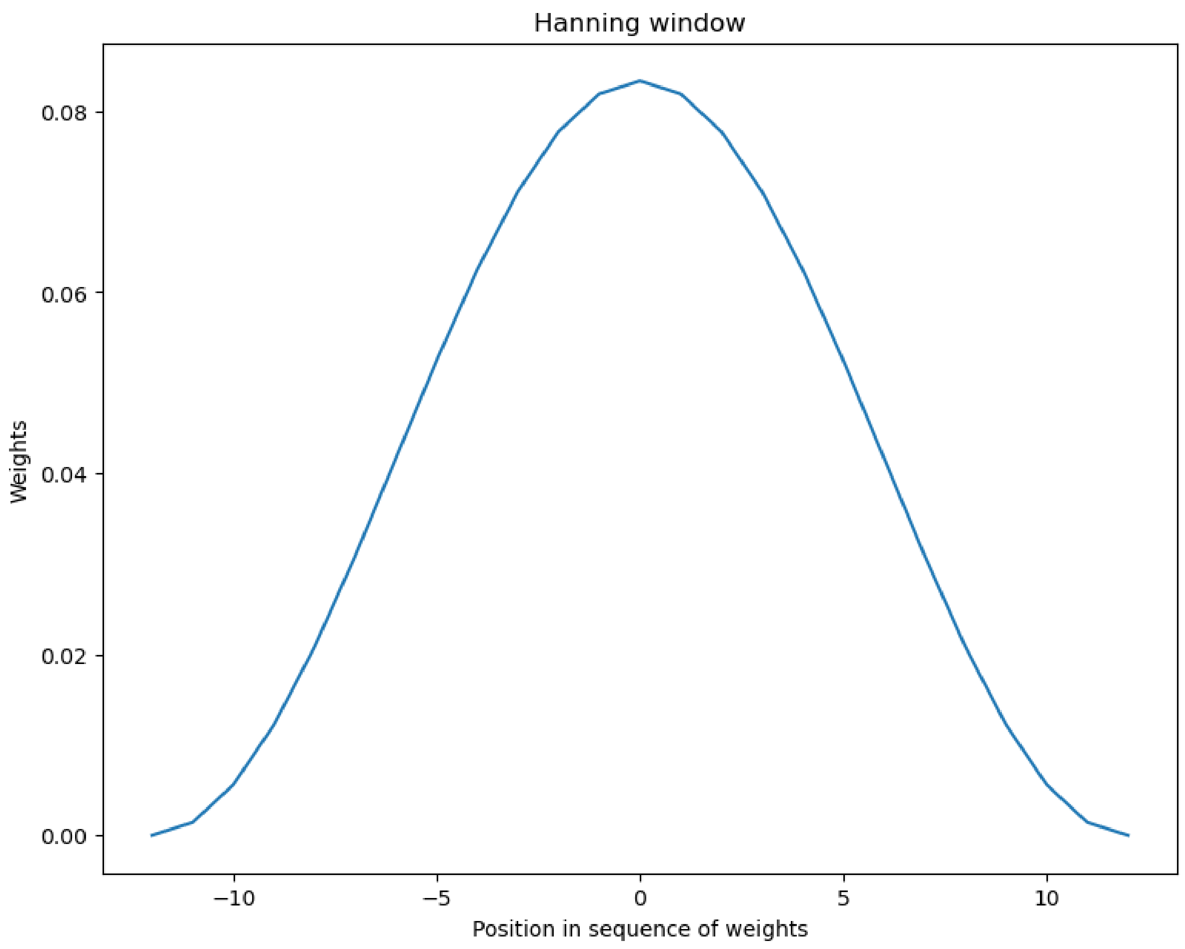
where the weights $w(-p), \ldots, w(p)$ are a sequence of $2p+1$ nonnegative values summing to one.

In general, larger values of p indicate more smoothing — more on this below.

The next figure shows the kind of sequence typically used.

Note the smaller weights towards the edges and larger weights in the center, so that more distant values from $I(\omega_j)$ have less weight than closer ones in the sum (16.3).

```
def hanning_window(M):
    w = [0.5 - 0.5 * np.cos(2 * np.pi * n/(M-1)) for n in range(M)]
    return w

window = hanning_window(25) / np.abs(sum(hanning_window(25)))
x = np.linspace(-12, 12, 25)
fig, ax = plt.subplots(figsize=(9, 7))
ax.plot(x, window)
ax.set_title("Hanning window")
ax.set_ylabel("Weights")
ax.set_xlabel("Position in sequence of weights")
plt.show()
```



16.3.1 Estimation with Smoothing

Our next step is to provide code that will not only estimate the periodogram but also provide smoothing as required.

Such functions have been written in estspec.py and are available once you've installed QuantEcon.py.

The GitHub listing displays three functions, `smooth()`, `periodogram()`, `ar_periodogram()`. We will discuss the first two here and the third one *below*.

The `periodogram()` function returns a periodogram, optionally smoothed via the `smooth()` function.

Regarding the `smooth()` function, since smoothing adds a nontrivial amount of computation, we have applied a fairly terse array-centric method based around `np.convolve`.

Readers are left either to explore or simply to use this code according to their interests.

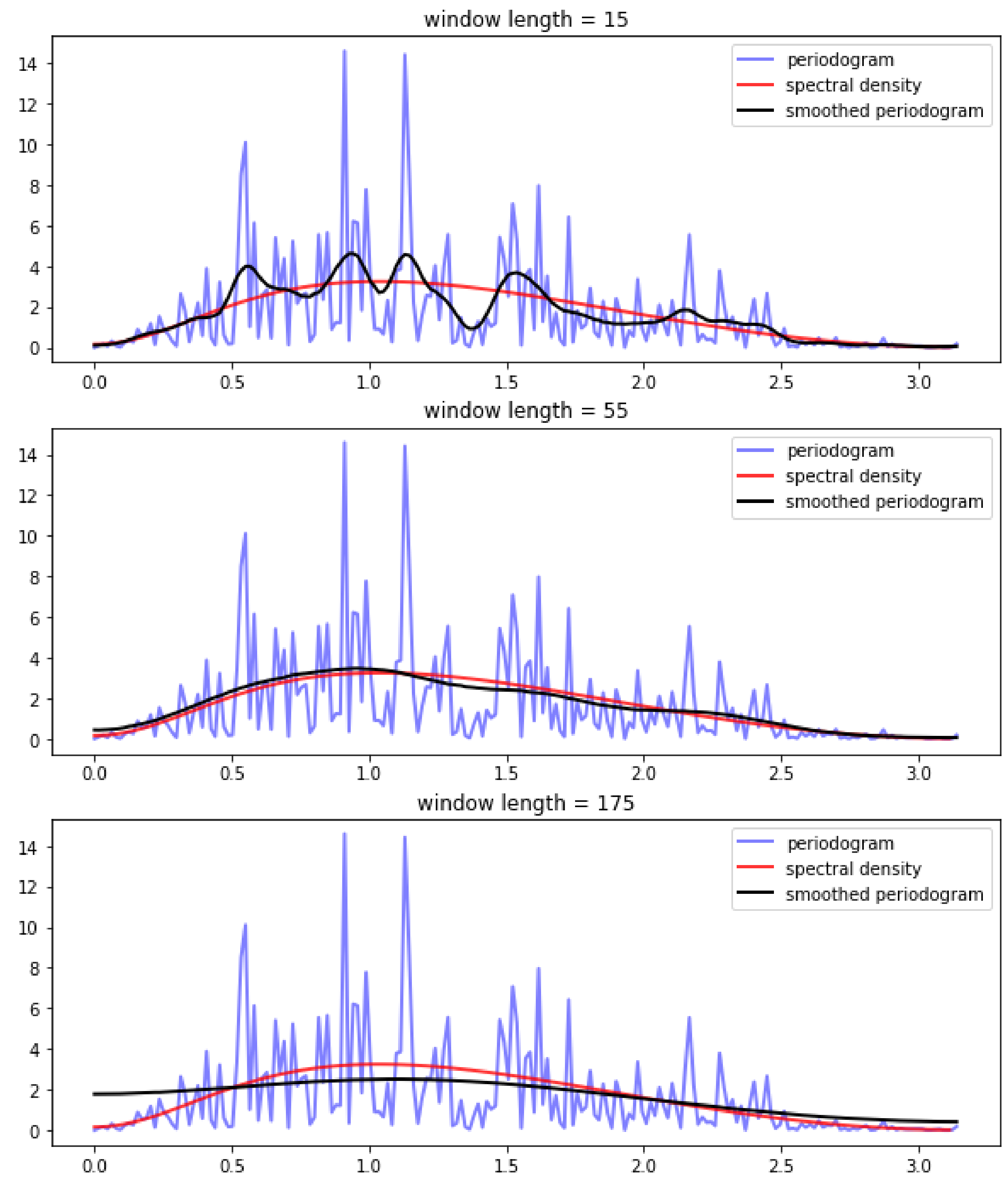
The next three figures each show smoothed and unsmoothed periodograms, as well as the population or "true" spectral density.

(The model is the same as before — see equation (16.2) — and there are 400 observations)

From the top figure to bottom, the window length is varied from small to large.

In looking at the figure, we can see that for this model and data size, the window length chosen in the middle figure provides the best fit.

Relative to this value, the first window length provides insufficient smoothing, while the third gives too much smoothing.


window length = 15
periodogram
spectral density
smoothed periodogram
window length = 55
periodogram
spectral density
smoothed periodogram
window length = 175
periodogram
spectral density
smoothed periodogram
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2
4
6
8
10
12
14

Of course in real estimation problems, the true spectral density is not visible and the choice of appropriate smoothing will have to be made based on judgement/priors or some other theory.

16.3.2 Pre-Filtering and Smoothing

In the code listing, we showed three functions from the file `estspec.py`.

The third function in the file (`ar_periodogram()`) adds a pre-processing step to periodogram smoothing.

First, we describe the basic idea, and after that we give the code.

The essential idea is to

1. Transform the data in order to make estimation of the spectral density more efficient.
2. Compute the periodogram associated with the transformed data.
3. Reverse the effect of the transformation on the periodogram, so that it now estimates the spectral density of the original process.

Step 1 is called *pre-filtering* or *pre-whitening*, while step 3 is called *recoloring*.

The first step is called pre-whitening because the transformation is usually designed to turn the data into something closer to white noise.

Why would this be desirable in terms of spectral density estimation?

The reason is that we are smoothing our estimated periodogram based on estimated values at nearby points — recall (16.3).

The underlying assumption that makes this a good idea is that the true spectral density is relatively regular — the value of $I(\omega)$ is close to that of $I(\omega')$ when ω is close to ω'.

This will not be true in all cases, but it is certainly true for white noise.

For white noise, I is as regular as possible — *it is a constant function*.

In this case, values of $I(\omega')$ at points ω' near to ω provided the maximum possible amount of information about the value $I(\omega)$.

Another way to put this is that if I is relatively constant, then we can use a large amount of smoothing without introducing too much bias.

16.3.3 The AR(1) Setting

Let's examine this idea more carefully in a particular setting — where the data are assumed to be generated by an AR(1) process.

(More general ARMA settings can be handled using similar techniques to those described below)

Suppose in particular that $\{X_t\}$ is covariance stationary and AR(1), with

$$X_{t+1} = \mu + \phi X_t + \epsilon_{t+1} \tag{16.4}$$

where μ and $\phi \in (-1, 1)$ are unknown parameters and $\{\epsilon_t\}$ is white noise.

It follows that if we regress X_{t+1} on X_t and an intercept, the residuals will approximate white noise.

Let

- g be the spectral density of $\{\epsilon_t\}$ — a constant function, as discussed above
- I_0 be the periodogram estimated from the residuals — an estimate of g

- f be the spectral density of $\{X_t\}$ — the object we are trying to estimate

In view of *an earlier result* we obtained while discussing ARMA processes, f and g are related by

$$f(\omega) = \left| \frac{1}{1 - \phi e^{i\omega}} \right|^2 g(\omega) \tag{16.5}$$

This suggests that the recoloring step, which constructs an estimate I of f from I_0, should set

$$I(\omega) = \left| \frac{1}{1 - \hat{\phi} e^{i\omega}} \right|^2 I_0(\omega)$$

where $\hat{\phi}$ is the OLS estimate of ϕ.

The code for `ar_periodogram()` — the third function in `estspec.py` — does exactly this. (See the code here).

The next figure shows realizations of the two kinds of smoothed periodograms

1. "standard smoothed periodogram", the ordinary smoothed periodogram, and
2. "AR smoothed periodogram", the pre-whitened and recolored one generated by `ar_periodogram()`

The periodograms are calculated from time series drawn from (16.4) with $\mu = 0$ and $\phi = -0.9$.

Each time series is of length 150.

The difference between the three subfigures is just randomness — each one uses a different draw of the time series.

In all cases, periodograms are fit with the "hamming" window and window length of 65.

Overall, the fit of the AR smoothed periodogram is much better, in the sense of being closer to the true spectral density.

16.4 Exercises

Exercise 16.4.1

Replicate *this figure* (modulo randomness).

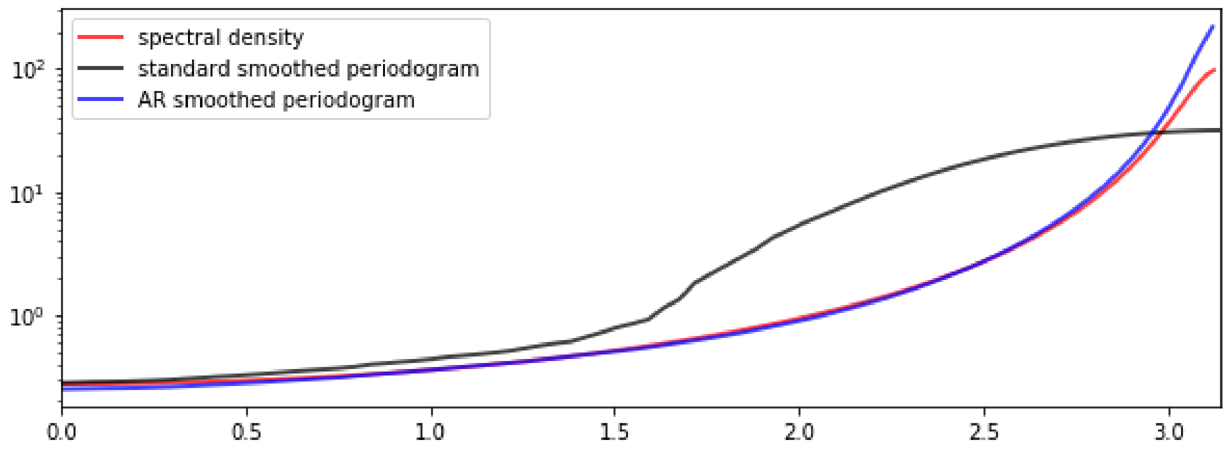
The model is as in equation (16.2) and there are 400 observations.

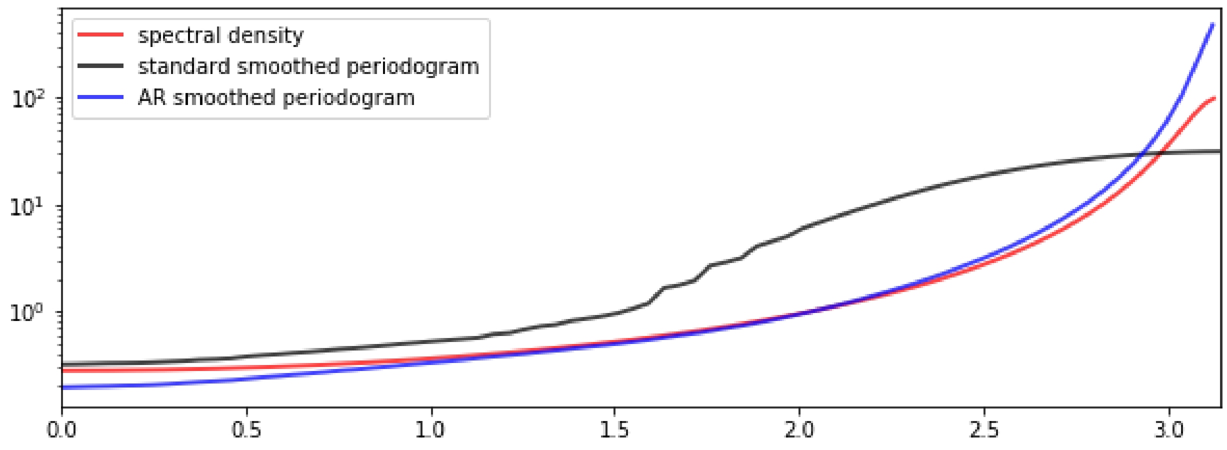
For the smoothed periodogram, the window type is "hamming".

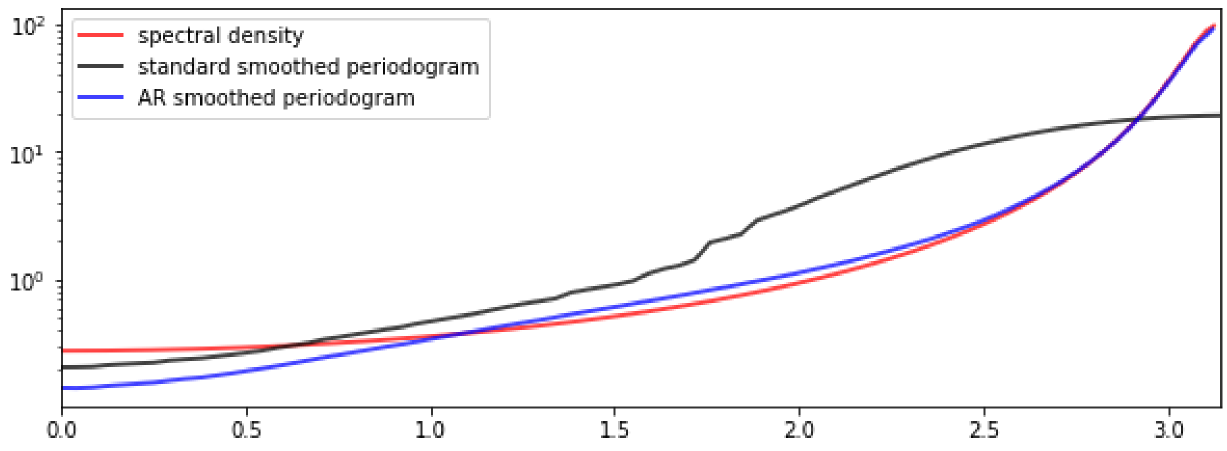
Solution to Exercise 16.4.1

```python
## Data
n = 400
ϕ = 0.5
θ = 0, -0.8
lp = ARMA(ϕ, θ)
X = lp.simulation(ts_length=n)

fig, ax = plt.subplots(3, 1, figsize=(10, 12))

for i, wl in enumerate((15, 55, 175)):  # Window lengths

    x, y = periodogram(X)
    ax[i].plot(x, y, 'b-', lw=2, alpha=0.5, label='periodogram')
```

(continues on next page)

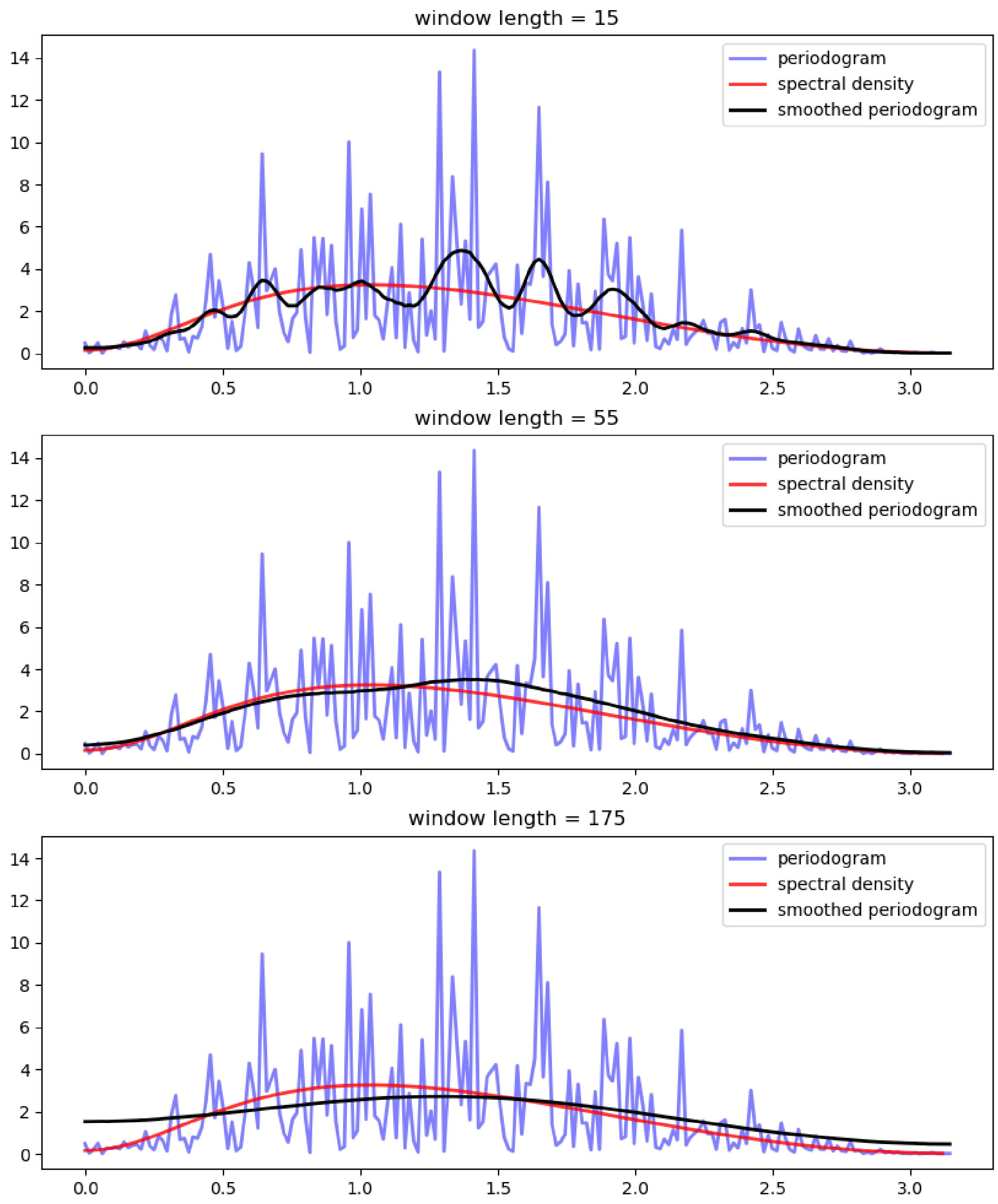

spectral density
standard smoothed periodogram
AR smoothed periodogram
10^2
10^1
10^0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


spectral density
standard smoothed periodogram
AR smoothed periodogram
10^2
10^1
10^0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


spectral density
standard smoothed periodogram
AR smoothed periodogram
10^2
10^1
10^0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

(continued from previous page)

```
    x_sd, y_sd = lp.spectral_density(two_pi=False, res=120)
    ax[i].plot(x_sd, y_sd, 'r-', lw=2, alpha=0.8, label='spectral density')

    x, y_smoothed = periodogram(X, window='hamming', window_len=wl)
    ax[i].plot(x, y_smoothed, 'k-', lw=2, label='smoothed periodogram')

    ax[i].legend()
    ax[i].set_title(f'window length = {wl}')
plt.show()
```



Exercise 16.4.2

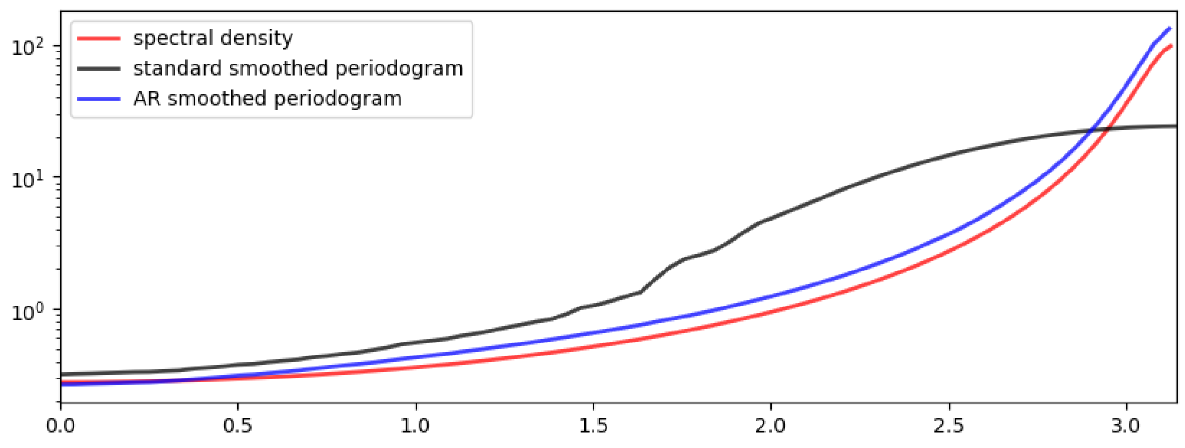
Replicate *this figure* (modulo randomness).

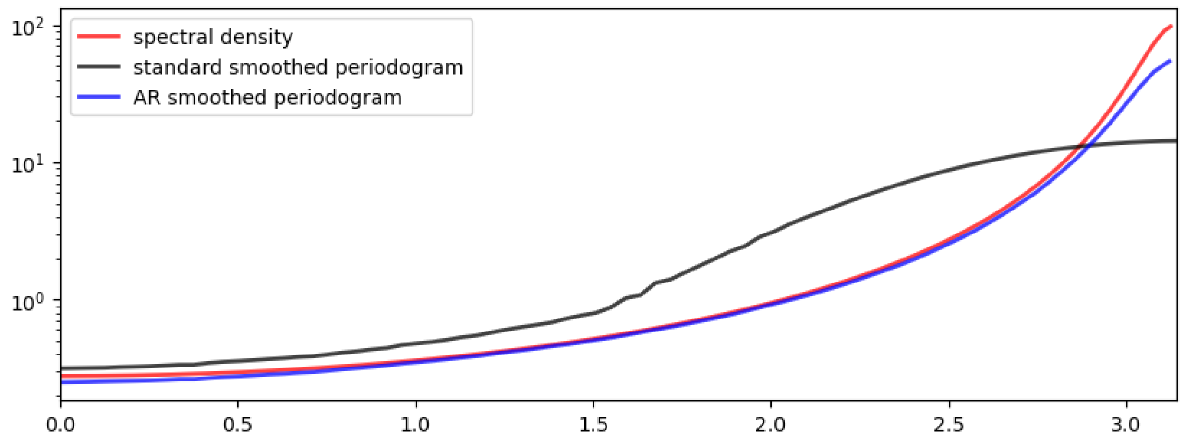
The model is as in equation (16.4), with $\mu = 0$, $\phi = -0.9$ and 150 observations in each time series.

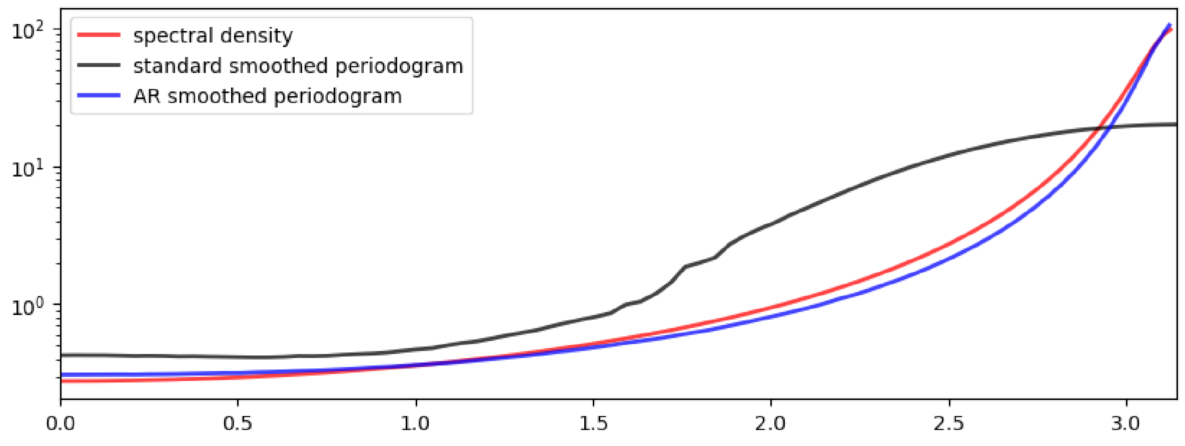
All periodograms are fit with the "hamming" window and window length of 65.

Solution to Exercise 16.4.2

```
lp = ARMA(-0.9)
wl = 65


fig, ax = plt.subplots(3, 1, figsize=(10,12))

for i in range(3):
    X = lp.simulation(ts_length=150)
    ax[i].set_xlim(0, np.pi)

    x_sd, y_sd = lp.spectral_density(two_pi=False, res=180)
    ax[i].semilogy(x_sd, y_sd, 'r-', lw=2, alpha=0.75,
        label='spectral density')

    x, y_smoothed = periodogram(X, window='hamming', window_len=wl)
    ax[i].semilogy(x, y_smoothed, 'k-', lw=2, alpha=0.75,
        label='standard smoothed periodogram')

    x, y_ar = ar_periodogram(X, window='hamming', window_len=wl)
    ax[i].semilogy(x, y_ar, 'b-', lw=2, alpha=0.75,
        label='AR smoothed periodogram')

    ax[i].legend(loc='upper left')
plt.show()
```


spectral density
standard smoothed periodogram
AR smoothed periodogram
10^2
10^1
10^0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


spectral density
standard smoothed periodogram
AR smoothed periodogram
10^2
10^1
10^0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


spectral density
standard smoothed periodogram
AR smoothed periodogram
10^2
10^1
10^0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

CHAPTER

SEVENTEEN

ADDITIVE AND MULTIPLICATIVE FUNCTIONALS

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
```

17.1 Overview

Many economic time series display persistent growth that prevents them from being asymptotically stationary and ergodic.

For example, outputs, prices, and dividends typically display irregular but persistent growth.

Asymptotic stationarity and ergodicity are key assumptions needed to make it possible to learn by applying statistical methods.

But there are good ways to model time series that have persistent growth that still enable statistical learning based on a law of large numbers for an asymptotically stationary and ergodic process.

Thus, [Hansen, 2012] described two classes of time series models that accommodate growth.

They are

1. **additive functionals** that display random "arithmetic growth"
2. **multiplicative functionals** that display random "geometric growth"

These two classes of processes are closely connected.

If a process $\{y_t\}$ is an additive functional and $\phi_t = \exp(y_t)$, then $\{\phi_t\}$ is a multiplicative functional.

In this lecture, we describe both additive functionals and multiplicative functionals.

We also describe and compute decompositions of additive and multiplicative processes into four components:

1. a **constant**
2. a **trend** component
3. an asymptotically **stationary** component
4. a **martingale**

We describe how to construct, simulate, and interpret these components.

More details about these concepts and algorithms can be found in Hansen [Hansen, 2012] and Hansen and Sargent [Hansen and Sargent, 2024].

Let's start with some imports:

```python
import numpy as np
import scipy.linalg as la
import quantecon as qe
import matplotlib.pyplot as plt
from scipy.stats import norm, lognorm
```

17.2 A Particular Additive Functional

[Hansen, 2012] describes a general class of additive functionals.

This lecture focuses on a subclass of these: a scalar process $\{y_t\}_{t=0}^{\infty}$ whose increments are driven by a Gaussian vector autoregression.

Our special additive functional displays interesting time series behavior while also being easy to construct, simulate, and analyze by using linear state-space tools.

We construct our additive functional from two pieces, the first of which is a **first-order vector autoregression** (VAR)

$$x_{t+1} = Ax_t + Bz_{t+1} \tag{17.1}$$

Here

- x_t is an $n \times 1$ vector,
- A is an $n \times n$ stable matrix (all eigenvalues lie within the open unit circle),
- $z_{t+1} \sim N(0, I)$ is an $m \times 1$ IID shock,
- B is an $n \times m$ matrix, and
- $x_0 \sim N(\mu_0, \Sigma_0)$ is a random initial condition for x

The second piece is an equation that expresses increments of $\{y_t\}_{t=0}^{\infty}$ as linear functions of

- a scalar constant ν,
- the vector x_t, and
- the same Gaussian vector z_{t+1} that appears in the VAR (17.1)

In particular,

$$y_{t+1} - y_t = \nu + Dx_t + Fz_{t+1} \tag{17.2}$$

Here $y_0 \sim N(\mu_{y0}, \Sigma_{y0})$ is a random initial condition for y.

The nonstationary random process $\{y_t\}_{t=0}^{\infty}$ displays systematic but random *arithmetic growth*.

17.2.1 Linear State-Space Representation

A convenient way to represent our additive functional is to use a linear state space system.

To do this, we set up state and observation vectors

$$\hat{x}_t = \begin{bmatrix} 1 \\ x_t \\ y_t \end{bmatrix} \quad \text{and} \quad \hat{y}_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix}$$

Next we construct a linear system

$$\begin{bmatrix} 1 \\ x_{t+1} \\ y_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & A & 0 \\ \nu & D & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_t \\ y_t \end{bmatrix} + \begin{bmatrix} 0 \\ B \\ F \end{bmatrix} z_{t+1}$$

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_t \\ y_t \end{bmatrix}$$

This can be written as

$$\begin{aligned} \hat{x}_{t+1} &= \hat{A}\hat{x}_t + \hat{B}z_{t+1} \\ \hat{y}_t &= \hat{D}\hat{x}_t \end{aligned}$$

which is a standard linear state space system.

To study it, we could map it into an instance of LinearStateSpace from QuantEcon.py.

But here we will use a different set of code for simulation, for reasons described below.

17.3 Dynamics

Let's run some simulations to build intuition.

In doing so we'll assume that z_{t+1} is scalar and that $\tilde{x}_t$ follows a 4th-order scalar autoregression.

$$\tilde{x}_{t+1} = \phi_1 \tilde{x}_t + \phi_2 \tilde{x}_{t-1} + \phi_3 \tilde{x}_{t-2} + \phi_4 \tilde{x}_{t-3} + \sigma z_{t+1} \tag{17.3}$$

in which the zeros z of the polynomial

$$\phi(z) = (1 - \phi_1 z - \phi_2 z^2 - \phi_3 z^3 - \phi_4 z^4)$$

are strictly greater than unity in absolute value.

(Being a zero of $\phi(z)$ means that $\phi(z) = 0$)

Let the increment in $\{y_t\}$ obey

$$y_{t+1} - y_t = \nu + \tilde{x}_t + \sigma z_{t+1}$$

with an initial condition for y_0.

While (17.3) is not a first order system like (17.1), we know that it can be mapped into a first order system.

- For an example of such a mapping, see this example.

In fact, this whole model can be mapped into the additive functional system definition in (17.1) – (17.2) by appropriate selection of the matrices A, B, D, F.

You can try writing these matrices down now as an exercise — correct expressions appear in the code below.

17.3.1 Simulation

When simulating we embed our variables into a bigger system.

This system also constructs the components of the decompositions of y_t and of $\exp(y_t)$ proposed by Hansen [Hansen, 2012].

All of these objects are computed using the code below

```
class AMF_LSS_VAR:
    """
    This class transforms an additive (multiplicative)
    functional into a QuantEcon linear state space system.
    """

    def __init__(self, A, B, D, F=None, ν=None):
        # Unpack required elements
        self.nx, self.nk = B.shape
        self.A, self.B = A, B

        # Checking the dimension of D (extended from the scalar case)
        if len(D.shape) > 1 and D.shape[0] != 1:
            self.nm = D.shape[0]
            self.D = D
        elif len(D.shape) > 1 and D.shape[0] == 1:
            self.nm = 1
            self.D = D
        else:
            self.nm = 1
            self.D = np.expand_dims(D, 0)

        # Create space for additive decomposition
        self.add_decomp = None
        self.mult_decomp = None

        # Set F
        if not np.any(F):
            self.F = np.zeros((self.nk, 1))
        else:
            self.F = F

        # Set ν
        if not np.any(ν):
            self.ν = np.zeros((self.nm, 1))
        elif type(ν) == float:
            self.ν = np.asarray([[ν]])
        elif len(ν.shape) == 1:
            self.ν = np.expand_dims(ν, 1)
        else:
            self.ν = ν

        if self.ν.shape[0] != self.D.shape[0]:
            raise ValueError("The dimension of ν is inconsistent with D!")

        # Construct BIG state space representation
        self.lss = self.construct_ss()

    def construct_ss(self):
```

(continues on next page)

(continued from previous page)

```
        """
        This creates the state space representation that can be passed
        into the quantecon LSS class.
        """
        # Pull out useful info
        nx, nk, nm = self.nx, self.nk, self.nm
        A, B, D, F, ν = self.A, self.B, self.D, self.F, self.ν
        if self.add_decomp:
            ν, H, g = self.add_decomp
        else:
            ν, H, g = self.additive_decomp()

        # Auxiliary blocks with 0's and 1's to fill out the lss matrices
        nx0c = np.zeros((nx, 1))
        nx0r = np.zeros(nx)
        nx1 = np.ones(nx)
        nk0 = np.zeros(nk)
        ny0c = np.zeros((nm, 1))
        ny0r = np.zeros(nm)
        ny1m = np.eye(nm)
        ny0m = np.zeros((nm, nm))
        nyx0m = np.zeros_like(D)

        # Build A matrix for LSS
        # Order of states is: [1, t, xt, yt, mt]
        A1 = np.hstack([1, 0, nx0r, ny0r, ny0r])          # Transition for 1
        A2 = np.hstack([1, 1, nx0r, ny0r, ny0r])          # Transition for t
        # Transition for x_{t+1}
        A3 = np.hstack([nx0c, nx0c, A, nyx0m.T, nyx0m.T])
        # Transition for y_{t+1}
        A4 = np.hstack([ν, ny0c, D, ny1m, ny0m])
        # Transition for m_{t+1}
        A5 = np.hstack([ny0c, ny0c, nyx0m, ny0m, ny1m])
        Abar = np.vstack([A1, A2, A3, A4, A5])

        # Build B matrix for LSS
        Bbar = np.vstack([nk0, nk0, B, F, H])

        # Build G matrix for LSS
        # Order of observation is: [xt, yt, mt, st, tt]
        # Selector for x_{t}
        G1 = np.hstack([nx0c, nx0c, np.eye(nx), nyx0m.T, nyx0m.T])
        G2 = np.hstack([ny0c, ny0c, nyx0m, ny1m, ny0m])    # Selector for y_{t}
        # Selector for martingale
        G3 = np.hstack([ny0c, ny0c, nyx0m, ny0m, ny1m])
        G4 = np.hstack([ny0c, ny0c, -g, ny0m, ny0m])  # Selector for stationary
        G5 = np.hstack([ny0c, ν, nyx0m, ny0m, ny0m])  # Selector for trend
        Gbar = np.vstack([G1, G2, G3, G4, G5])

        # Build H matrix for LSS
        Hbar = np.zeros((Gbar.shape[0], nk))

        # Build LSS type
        x0 = np.hstack([1, 0, nx0r, ny0r, ny0r])
        S0 = np.zeros((len(x0), len(x0)))
        lss = qe.LinearStateSpace(Abar, Bbar, Gbar, Hbar, mu_0=x0, Sigma_0=S0)
```

(continues on next page)

(continued from previous page)

```
        return lss

    def additive_decomp(self):
        """
        Return values for the martingale decomposition
            - ν     : unconditional mean difference in Y
            - H     : coefficient for the (linear) martingale component (κ_a)
            - g     : coefficient for the stationary component g(x)
            - Y_0   : it should be the function of X_0 (for now set it to 0.0)
        """
        I = np.identity(self.nx)
        A_res = la.solve(I - self.A, I)
        g = self.D @ A_res
        H = self.F + self.D @ A_res @ self.B

        return self.ν, H, g

    def multiplicative_decomp(self):
        """
        Return values for the multiplicative decomposition (Example 5.4.4.)
            - ν_tilde  : eigenvalue
            - H        : vector for the Jensen term
        """
        ν, H, g = self.additive_decomp()
        ν_tilde = ν + (.5)*np.expand_dims(np.diag(H @ H.T), 1)

        return ν_tilde, H, g

    def loglikelihood_path(self, x, y):
        A, B, D, F = self.A, self.B, self.D, self.F
        k, T = y.shape
        FF = F @ F.T
        FFinv = la.inv(FF)
        temp = y[:, 1:] - y[:, :-1] - D @ x[:, :-1]
        obs =  temp * FFinv * temp
        obssum = np.cumsum(obs)
        scalar = (np.log(la.det(FF)) + k*np.log(2*np.pi))*np.arange(1, T)

        return -(.5)*(obssum + scalar)

    def loglikelihood(self, x, y):
        llh = self.loglikelihood_path(x, y)

        return llh[-1]
```

Plotting

The code below adds some functions that generate plots for instances of the `AMF_LSS_VAR` *class*.

```
def plot_given_paths(amf, T, ypath, mpath, spath, tpath,
                     mbounds, sbounds, horline=0, show_trend=True):

    # Allocate space
    trange = np.arange(T)

    # Create figure
    fig, ax = plt.subplots(2, 2, sharey=True, figsize=(15, 8))

    # Plot all paths together
    ax[0, 0].plot(trange, ypath[0, :], label="$y_t$", color="k")
    ax[0, 0].plot(trange, mpath[0, :], label="$m_t$", color="m")
    ax[0, 0].plot(trange, spath[0, :], label="$s_t$", color="g")
    if show_trend:
        ax[0, 0].plot(trange, tpath[0, :], label="$t_t$", color="r")
    ax[0, 0].axhline(horline, color="k", linestyle="-.")
    ax[0, 0].set_title("One Path of All Variables")
    ax[0, 0].legend(loc="upper left")

    # Plot Martingale Component
    ax[0, 1].plot(trange, mpath[0, :], "m")
    ax[0, 1].plot(trange, mpath.T, alpha=0.45, color="m")
    ub = mbounds[1, :]
    lb = mbounds[0, :]

    ax[0, 1].fill_between(trange, lb, ub, alpha=0.25, color="m")
    ax[0, 1].set_title("Martingale Components for Many Paths")
    ax[0, 1].axhline(horline, color="k", linestyle="-.")

    # Plot Stationary Component
    ax[1, 0].plot(spath[0, :], color="g")
    ax[1, 0].plot(spath.T, alpha=0.25, color="g")
    ub = sbounds[1, :]
    lb = sbounds[0, :]
    ax[1, 0].fill_between(trange, lb, ub, alpha=0.25, color="g")
    ax[1, 0].axhline(horline, color="k", linestyle="-.")
    ax[1, 0].set_title("Stationary Components for Many Paths")

    # Plot Trend Component
    if show_trend:
        ax[1, 1].plot(tpath.T, color="r")
    ax[1, 1].set_title("Trend Components for Many Paths")
    ax[1, 1].axhline(horline, color="k", linestyle="-.")

    return fig

def plot_additive(amf, T, npaths=25, show_trend=True):
    """
    Plots for the additive decomposition.
    Acts on an instance amf of the AMF_LSS_VAR class

    """
    # Pull out right sizes so we know how to increment
```

(continues on next page)

(continued from previous page)

```
    nx, nk, nm = amf.nx, amf.nk, amf.nm

    # Allocate space (nm is the number of additive functionals -
    # we want npaths for each)
    mpath = np.empty((nm*npaths, T))
    mbounds = np.empty((nm*2, T))
    spath = np.empty((nm*npaths, T))
    sbounds = np.empty((nm*2, T))
    tpath = np.empty((nm*npaths, T))
    ypath = np.empty((nm*npaths, T))

    # Simulate for as long as we wanted
    moment_generator = amf.lss.moment_sequence()
    # Pull out population moments
    for t in range (T):
        tmoms = next(moment_generator)
        ymeans = tmoms[1]
        yvar = tmoms[3]

        # Lower and upper bounds - for each additive functional
        for ii in range(nm):
            li, ui = ii*2, (ii+1)*2
            mscale = np.sqrt(yvar[nx+nm+ii, nx+nm+ii])
            sscale = np.sqrt(yvar[nx+2*nm+ii, nx+2*nm+ii])
            if mscale == 0.0:
                mscale = 1e-12   # avoids a RuntimeWarning from calculating ppf
            if sscale == 0.0:    # of normal distribution with std dev = 0.
                sscale = 1e-12   # sets std dev to small value instead

            madd_dist = norm(ymeans[nx+nm+ii], mscale)
            sadd_dist = norm(ymeans[nx+2*nm+ii], sscale)

            mbounds[li:ui, t] = madd_dist.ppf([0.01, .99])
            sbounds[li:ui, t] = sadd_dist.ppf([0.01, .99])

    # Pull out paths
    for n in range(npaths):
        x, y = amf.lss.simulate(T)
        for ii in range(nm):
            ypath[npaths*ii+n, :] = y[nx+ii, :]
            mpath[npaths*ii+n, :] = y[nx+nm + ii, :]
            spath[npaths*ii+n, :] = y[nx+2*nm + ii, :]
            tpath[npaths*ii+n, :] = y[nx+3*nm + ii, :]

    add_figs = []

    for ii in range(nm):
        li, ui = npaths*(ii), npaths*(ii+1)
        LI, UI = 2*(ii), 2*(ii+1)
        add_figs.append(plot_given_paths(amf, T,
                                          ypath[li:ui,:],
                                          mpath[li:ui,:],
                                          spath[li:ui,:],
                                          tpath[li:ui,:],
                                          mbounds[LI:UI,:],
                                          sbounds[LI:UI,:],
```

(continues on next page)

(continued from previous page)

```
                                                    show_trend=show_trend))

        add_figs[ii].suptitle(f'Additive decomposition of $y_{ii+1}$',
                              fontsize=14)

    return add_figs


def plot_multiplicative(amf, T, npaths=25, show_trend=True):
    """
    Plots for the multiplicative decomposition

    """
    # Pull out right sizes so we know how to increment
    nx, nk, nm = amf.nx, amf.nk, amf.nm
    # Matrices for the multiplicative decomposition
    v_tilde, H, g = amf.multiplicative_decomp()

    # Allocate space (nm is the number of functionals -
    # we want npaths for each)
    mpath_mult = np.empty((nm*npaths, T))
    mbounds_mult = np.empty((nm*2, T))
    spath_mult = np.empty((nm*npaths, T))
    sbounds_mult = np.empty((nm*2, T))
    tpath_mult = np.empty((nm*npaths, T))
    ypath_mult = np.empty((nm*npaths, T))

    # Simulate for as long as we wanted
    moment_generator = amf.lss.moment_sequence()
    # Pull out population moments
    for t in range(T):
        tmoms = next(moment_generator)
        ymeans = tmoms[1]
        yvar = tmoms[3]

        # Lower and upper bounds - for each multiplicative functional
        for ii in range(nm):
            li, ui = ii*2, (ii+1)*2
            Mdist = lognorm(np.sqrt(yvar[nx+nm+ii, nx+nm+ii]).item(),
                            scale=np.exp(ymeans[nx+nm+ii] \
                                                    - t * (.5)
                                                    * np.expand_dims(
                                                        np.diag(H @ H.T),
                                                        1
                                                        )[ii]
                                                    ).item()
                                                )
            Sdist = lognorm(np.sqrt(yvar[nx+2*nm+ii, nx+2*nm+ii]).item(),
                            scale = np.exp(-ymeans[nx+2*nm+ii]).item())
            mbounds_mult[li:ui, t] = Mdist.ppf([.01, .99])
            sbounds_mult[li:ui, t] = Sdist.ppf([.01, .99])

    # Pull out paths
    for n in range(npaths):
        x, y = amf.lss.simulate(T)
        for ii in range(nm):
```

(continues on next page)

(continued from previous page)

```
            ypath_mult[npaths*ii+n, :] = np.exp(y[nx+ii, :])
            mpath_mult[npaths*ii+n, :] = np.exp(y[nx+nm + ii, :] \
                                    - np.arange(T)*(.5)
                                    * np.expand_dims(np.diag(H
                                                        @ H.T),
                                                        1)[ii]
                                    )
            spath_mult[npaths*ii+n, :] = 1/np.exp(-y[nx+2*nm + ii, :])
            tpath_mult[npaths*ii+n, :] = np.exp(y[nx+3*nm + ii, :]
                                    + np.arange(T)*(.5)
                                    * np.expand_dims(np.diag(H
                                                        @ H.T),
                                                        1)[ii]
                                    )

    mult_figs = []

    for ii in range(nm):
        li, ui = npaths*(ii), npaths*(ii+1)
        LI, UI = 2*(ii), 2*(ii+1)

        mult_figs.append(plot_given_paths(amf,T,
                                        ypath_mult[li:ui,:],
                                        mpath_mult[li:ui,:],
                                        spath_mult[li:ui,:],
                                        tpath_mult[li:ui,:],
                                        mbounds_mult[LI:UI,:],
                                        sbounds_mult[LI:UI,:],
                                        1,
                                        show_trend=show_trend))
        mult_figs[ii].suptitle(f'Multiplicative decomposition of \
                            $y_{ii+1}$', fontsize=14)

    return mult_figs

def plot_martingale_paths(amf, T, mpath, mbounds, horline=1, show_trend=False):
    # Allocate space
    trange = np.arange(T)

    # Create figure
    fig, ax = plt.subplots(1, 1, figsize=(10, 6))

    # Plot Martingale Component
    ub = mbounds[1, :]
    lb = mbounds[0, :]
    ax.fill_between(trange, lb, ub, color="#ffccff")
    ax.axhline(horline, color="k", linestyle="-.")
    ax.plot(trange, mpath.T, linewidth=0.25, color="#4c4c4c")

    return fig

def plot_martingales(amf, T, npaths=25):

    # Pull out right sizes so we know how to increment
    nx, nk, nm = amf.nx, amf.nk, amf.nm
    # Matrices for the multiplicative decomposition
```

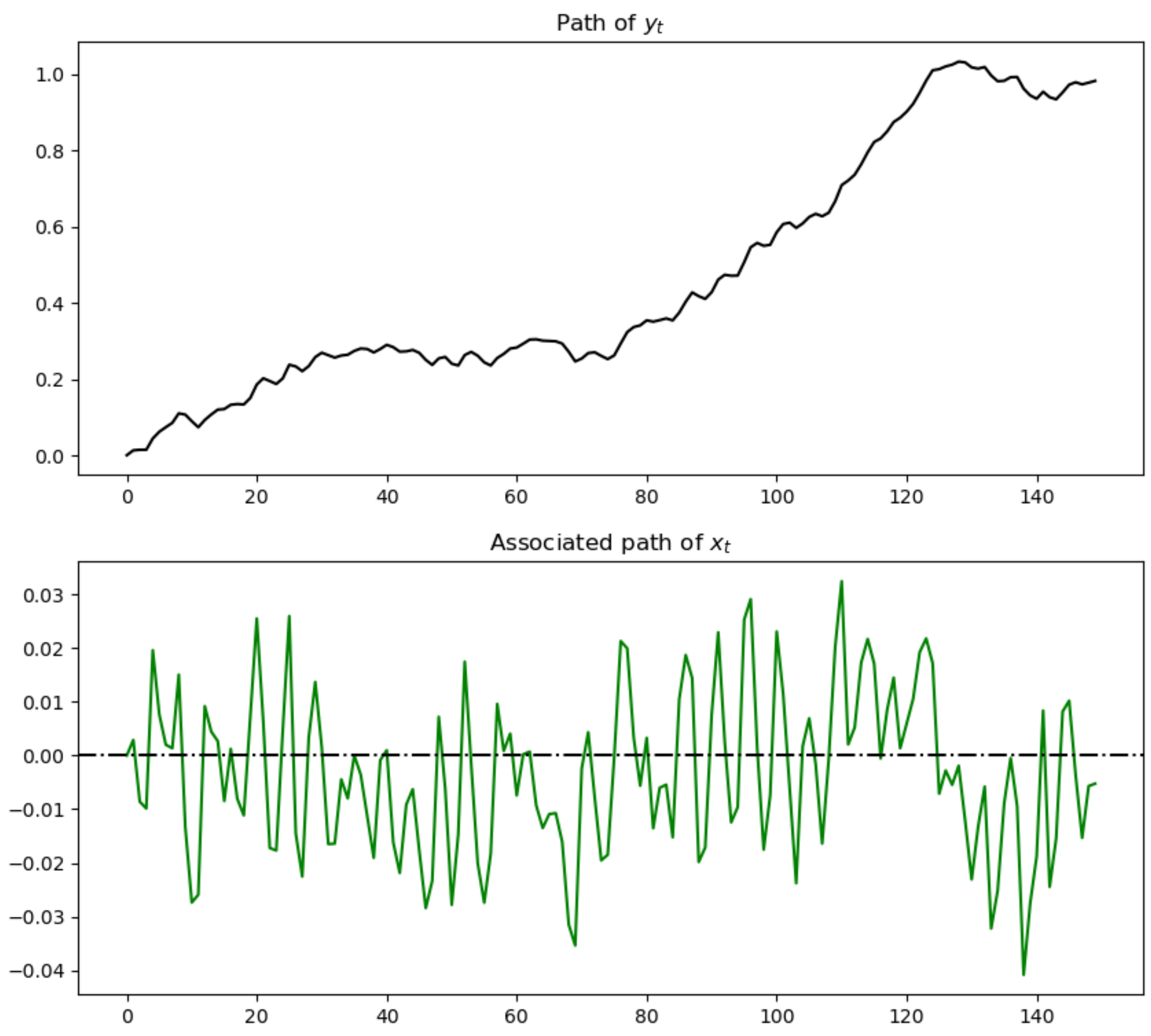
(continues on next page)

(continued from previous page)

```
    v_tilde, H, g = amf.multiplicative_decomp()

    # Allocate space (nm is the number of functionals -
    # we want npaths for each)
    mpath_mult = np.empty((nm*npaths, T))
    mbounds_mult = np.empty((nm*2, T))

    # Simulate for as long as we wanted
    moment_generator = amf.lss.moment_sequence()
    # Pull out population moments
    for t in range (T):
        tmoms = next(moment_generator)
        ymeans = tmoms[1]
        yvar = tmoms[3]

        # Lower and upper bounds - for each functional
        for ii in range(nm):
            li, ui = ii*2, (ii+1)*2
            Mdist = lognorm(np.sqrt(yvar[nx+nm+ii, nx+nm+ii]).item(),
                            scale= np.exp(ymeans[nx+nm+ii] \
                                                        - t * (.5)
                                                        * np.expand_dims(
                                                            np.diag(H @ H.T),
                                                            1)[ii]

                                                ).item()
                            )
            mbounds_mult[li:ui, t] = Mdist.ppf([.01, .99])

    # Pull out paths
    for n in range(npaths):
        x, y = amf.lss.simulate(T)
        for ii in range(nm):
            mpath_mult[npaths*ii+n, :] = np.exp(y[nx+nm + ii, :] \
                                                - np.arange(T) * (.5)
                                                * np.expand_dims(np.diag(H
                                                                    @ H.T),
                                                                1)[ii]
                                                )

    mart_figs = []

    for ii in range(nm):
        li, ui = npaths*(ii), npaths*(ii+1)
        LI, UI = 2*(ii), 2*(ii+1)
        mart_figs.append(plot_martingale_paths(amf, T, mpath_mult[li:ui, :],
                                                    mbounds_mult[LI:UI, :],
                                                    horline=1))
        mart_figs[ii].suptitle(f'Martingale components for many paths of \
                                $y_{ii+1}$', fontsize=14)

    return mart_figs
```

For now, we just plot y_t and x_t, postponing until later a description of exactly how we compute them.

```
ϕ_1, ϕ_2, ϕ_3, ϕ_4 = 0.5, -0.2, 0, 0.5
σ = 0.01
ν = 0.01   # Growth rate

# A matrix should be n x n
A = np.array([[ϕ_1, ϕ_2, ϕ_3, ϕ_4],
              [  1,   0,   0,   0],
              [  0,   1,   0,   0],
              [  0,   0,   1,   0]])

# B matrix should be n x k
B = np.array([[σ, 0, 0, 0]]).T

D = np.array([1, 0, 0, 0]) @ A
F = np.array([1, 0, 0, 0]) @ B

amf = AMF_LSS_VAR(A, B, D, F, ν=ν)

T = 150
x, y = amf.lss.simulate(T)

fig, ax = plt.subplots(2, 1, figsize=(10, 9))

ax[0].plot(np.arange(T), y[amf.nx, :], color='k')
ax[0].set_title('Path of $y_t$')
ax[1].plot(np.arange(T), y[0, :], color='g')
ax[1].axhline(0, color='k', linestyle='-.')
ax[1].set_title('Associated path of $x_t$')
plt.show()
```



Notice the irregular but persistent growth in y_t.

17.3.2 Decomposition

Hansen and Sargent [Hansen and Sargent, 2024] describe how to construct a decomposition of an additive functional into four parts:

- a constant inherited from initial values x_0 and y_0
- a linear trend
- a martingale
- an (asymptotically) stationary component

To attain this decomposition for the particular class of additive functionals defined by (17.1) and (17.2), we first construct the matrices

$$
\begin{aligned}
H &:= F + D(I - A)^{-1} B \\
g &:= D(I - A)^{-1}
\end{aligned}
$$

Then the Hansen [Hansen, 2012], [Hansen and Sargent, 2024] decomposition is

$$y_t = \underbrace{t\nu}_{\text{trend component}} + \overbrace{\sum_{j=1}^{t} Hz_j}^{\text{Martingale component}} - \underbrace{gx_t}_{\text{stationary component}} + \overbrace{gx_0 + y_0}^{\text{initial conditions}}$$

At this stage, you should pause and verify that $y_{t+1} - y_t$ satisfies (17.2).

It is convenient for us to introduce the following notation:

- $\tau_t = \nu t$, a linear, deterministic trend
- $m_t = \sum_{j=1}^{t} Hz_j$, a martingale with time $t+1$ increment Hz_{t+1}
- $s_t = gx_t$, an (asymptotically) stationary component

We want to characterize and simulate components τ_t, m_t, s_t of the decomposition.

A convenient way to do this is to construct an appropriate instance of a linear state space system by using LinearStateSpace from QuantEcon.py.

This will allow us to use the routines in LinearStateSpace to study dynamics.

To start, observe that, under the dynamics in (17.1) and (17.2) and with the definitions just given,

$$\begin{bmatrix} 1 \\ t+1 \\ x_{t+1} \\ y_{t+1} \\ m_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & A & 0 & 0 \\ \nu & 0 & D & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ x_t \\ y_t \\ m_t \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ B \\ F \\ H \end{bmatrix} z_{t+1}$$

and

$$\begin{bmatrix} x_t \\ y_t \\ \tau_t \\ m_t \\ s_t \end{bmatrix} = \begin{bmatrix} 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & \nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -g & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ x_t \\ y_t \\ m_t \end{bmatrix}$$

With

$$\tilde{x} := \begin{bmatrix} 1 \\ t \\ x_t \\ y_t \\ m_t \end{bmatrix} \quad \text{and} \quad \tilde{y} := \begin{bmatrix} x_t \\ y_t \\ \tau_t \\ m_t \\ s_t \end{bmatrix}$$

we can write this as the linear state space system

$$\begin{aligned} \tilde{x}_{t+1} &= \tilde{A}\tilde{x}_t + \tilde{B}z_{t+1} \\ \tilde{y}_t &= \tilde{D}\tilde{x}_t \end{aligned}$$

By picking out components of $\tilde{y}_t$, we can track all variables of interest.

17.4 Code

The class `AMF_LSS_VAR` mentioned *above* does all that we want to study our additive functional.

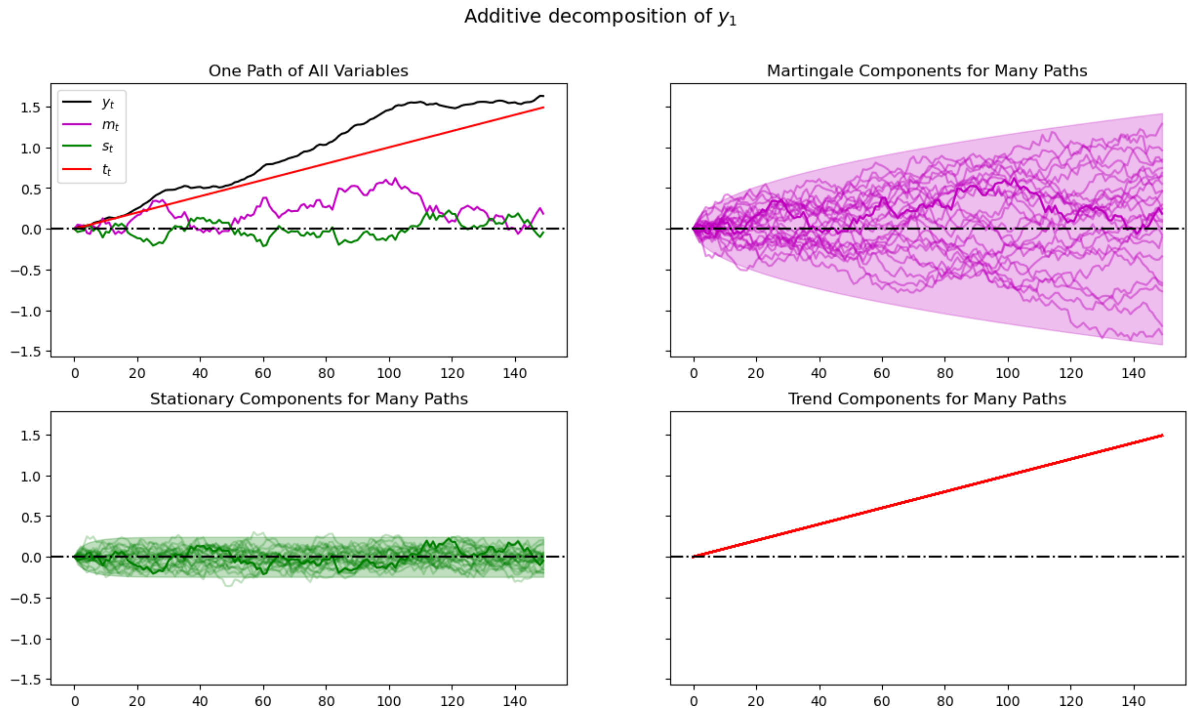
In fact, `AMF_LSS_VAR` does more because it allows us to study an associated multiplicative functional as well.

(A hint that it does more is the name of the class – here AMF stands for "additive and multiplicative functional" – the code computes and displays objects associated with multiplicative functionals too.)

Let's use this code (embedded above) to explore the *example process described above*.

If you run *the code that first simulated that example* again and then the method call you will generate (modulo randomness) the plot

```
plot_additive(amf, T)
plt.show()
```



When we plot multiple realizations of a component in the 2nd, 3rd, and 4th panels, we also plot the population 95% probability coverage sets computed using the LinearStateSpace class.

We have chosen to simulate many paths, all starting from the *same* non-random initial conditions x_0, y_0 (you can tell this from the shape of the 95% probability coverage shaded areas).

Notice tell-tale signs of these probability coverage shaded areas

- the purple one for the martingale component m_t grows with $\sqrt{t}$
- the green one for the stationary component s_t converges to a constant band

17.4.1 Associated Multiplicative Functional

Where $\{y_t\}$ is our additive functional, let $M_t = \exp(y_t)$.

As mentioned above, the process $\{M_t\}$ is called a **multiplicative functional**.

Corresponding to the additive decomposition described above we have a multiplicative decomposition of M_t

$$\frac{M_t}{M_0} = \exp(t\nu) \exp\Big(\sum_{j=1}^{t} H \cdot Z_j\Big) \exp\Big(D(I-A)^{-1}x_0 - D(I-A)^{-1}x_t\Big)$$

or

$$\frac{M_t}{M_0} = \exp(\tilde{\nu} t) \left(\frac{\widetilde{M}_t}{\widetilde{M}_0}\right) \left(\frac{\tilde{e}(X_0)}{\tilde{e}(x_t)}\right)$$

where

$$\tilde{\nu} = \nu + \frac{H \cdot H}{2}, \quad \widetilde{M}_t = \exp\left(\sum_{j=1}^{t}\left(H \cdot z_j - \frac{H \cdot H}{2}\right)\right), \quad \widetilde{M}_0 = 1$$

and

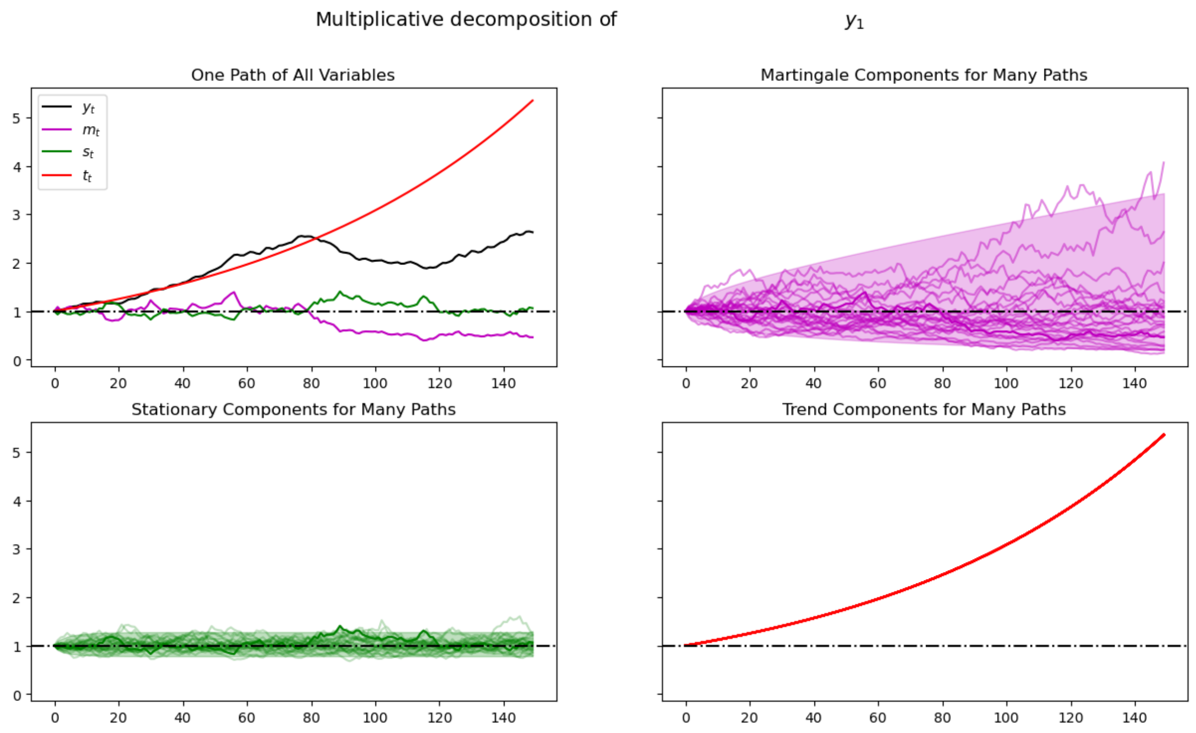
$$\tilde{e}(x) = \exp[g(x)] = \exp\big[D(I-A)^{-1}x\big]$$

An instance of class `AMF_LSS_VAR` (*above*) includes this associated multiplicative functional as an attribute.

Let's plot this multiplicative functional for our example.

If you run *the code that first simulated that example* again and then the method call in the cell below you'll obtain the graph in the next cell.

```
plot_multiplicative(amf, T)
plt.show()
```



As before, when we plotted multiple realizations of a component in the 2nd, 3rd, and 4th panels, we also plotted population 95% confidence bands computed using the LinearStateSpace class.

Comparing this figure and the last also helps show how geometric growth differs from arithmetic growth.

The top right panel of the above graph shows a panel of martingales associated with the panel of $M_t = \exp(y_t)$ that we have generated for a limited horizon T.

It is interesting to how the martingale behaves as $T \to +\infty$.

Let's see what happens when we set $T = 12000$ instead of 150.

17.4.2 Peculiar Large Sample Property

Hansen and Sargent [Hansen and Sargent, 2024] (ch. 8) describe the following two properties of the martingale component $\widetilde{M}_t$ of the multiplicative decomposition

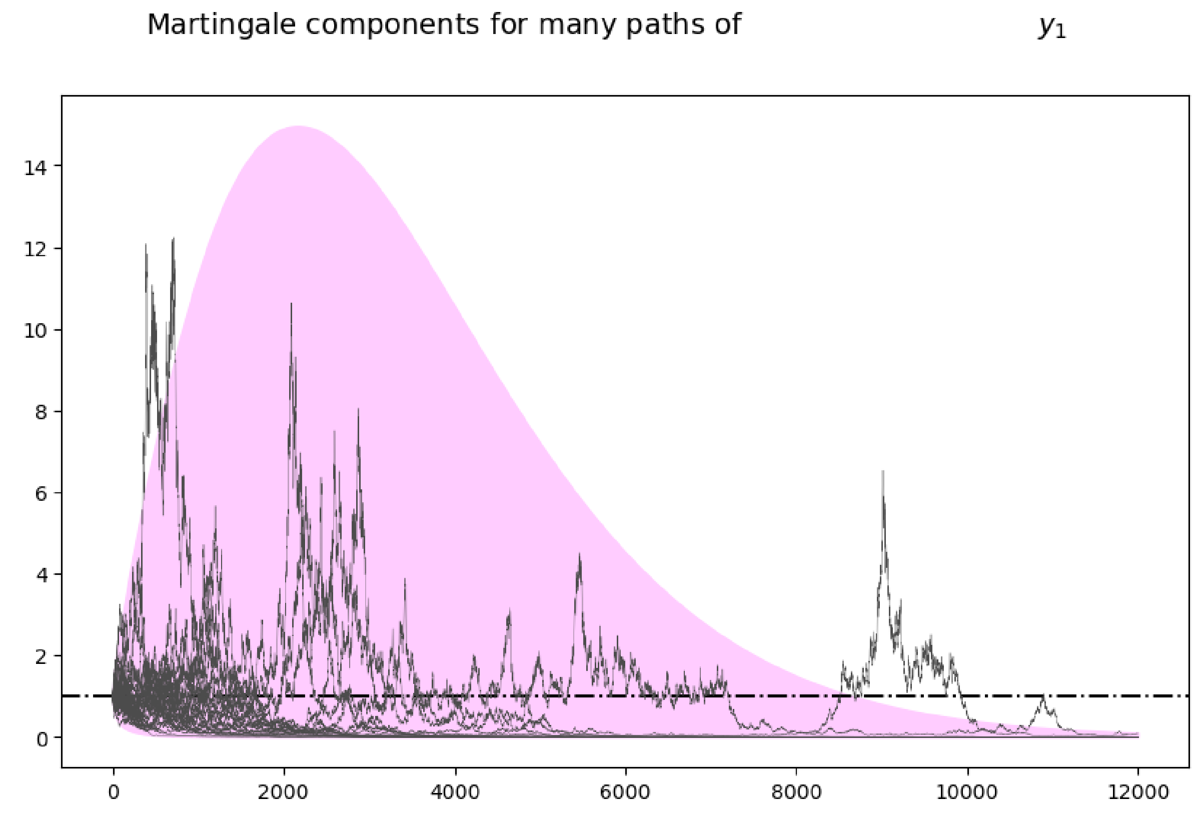
- while $E_0 \widetilde{M}_t = 1$ for all $t \geq 0$, nevertheless …
- as $t \to +\infty$, $\widetilde{M}_t$ converges to zero almost surely

The first property follows from the fact that $\widetilde{M}_t$ is a multiplicative martingale with initial condition $\widetilde{M}_0 = 1$.

The second is a **peculiar property** noted and proved by Hansen and Sargent [Hansen and Sargent, 2024].

The following simulation of many paths of $\widetilde{M}_t$ illustrates both properties

```
np.random.seed(10021987)
plot_martingales(amf, 12000)
plt.show()
```



The dotted line in the above graph is the mean $E\tilde{M}_t = 1$ of the martingale.

It remains constant at unity, illustrating the first property.

The purple 95 percent frequency coverage interval collapses around zero, illustrating the second property.

17.5 More About the Multiplicative Martingale

Let's drill down and study probability distribution of the multiplicative martingale $\{\widetilde{M}_t\}_{t=0}^\infty$ in more detail.

As we have seen, it has representation

$$\widetilde{M}_t = \exp\left(\sum_{j=1}^{t}\left(H \cdot z_j - \frac{H \cdot H}{2}\right)\right), \quad \widetilde{M}_0 = 1$$

where $H = [F + D(I - A)^{-1}B]$.

It follows that $\log \widetilde{M}_t \sim \mathcal{N}(-\frac{tH \cdot H}{2}, tH \cdot H)$ and that consequently $\widetilde{M}_t$ is log normal.

17.5.1 Simulating a Multiplicative Martingale Again

Next, we want a program to simulate the likelihood ratio process $\{\tilde{M}_t\}_{t=0}^\infty$.

In particular, we want to simulate 5000 sample paths of length T for the case in which x is a scalar and $[A, B, D, F] = [0.8, 0.001, 1.0, 0.01]$ and $\nu = 0.005$.

After accomplishing this, we want to display and study histograms of $\tilde{M}_T^i$ for various values of T.

Here is code that accomplishes these tasks.

17.5.2 Sample Paths

Let's write a program to simulate sample paths of $\{x_t, y_t\}_{t=0}^\infty$.

We'll do this by formulating the additive functional as a linear state space model and putting the LinearStateSpace class to work.

```python
class AMF_LSS_VAR:
    """
    This class is written to transform a scalar additive functional
    into a linear state space system.
    """
    def __init__(self, A, B, D, F=0.0, ν=0.0):
        # Unpack required elements
        self.A, self.B, self.D, self.F, self.ν = A, B, D, F, ν

        # Create space for additive decomposition
        self.add_decomp = None
        self.mult_decomp = None

        # Construct BIG state space representation
        self.lss = self.construct_ss()

    def construct_ss(self):
```

(continues on next page)

(continued from previous page)

```
        """
        This creates the state space representation that can be passed
        into the quantecon LSS class.
        """
        # Pull out useful info
        A, B, D, F, ν = self.A, self.B, self.D, self.F, self.ν
        nx, nk, nm = 1, 1, 1
        if self.add_decomp:
            ν, H, g = self.add_decomp
        else:
            ν, H, g = self.additive_decomp()

        # Build A matrix for LSS
        # Order of states is: [1, t, xt, yt, mt]
        A1 = np.hstack([1, 0, 0, 0, 0])       # Transition for 1
        A2 = np.hstack([1, 1, 0, 0, 0])       # Transition for t
        A3 = np.hstack([0, 0, A, 0, 0])       # Transition for x_{t+1}
        A4 = np.hstack([ν, 0, D, 1, 0])       # Transition for y_{t+1}
        A5 = np.hstack([0, 0, 0, 0, 1])       # Transition for m_{t+1}
        Abar = np.vstack([A1, A2, A3, A4, A5])

        # Build B matrix for LSS
        Bbar = np.vstack([0, 0, B, F, H])

        # Build G matrix for LSS
        # Order of observation is: [xt, yt, mt, st, tt]
        G1 = np.hstack([0, 0, 1, 0, 0])            # Selector for x_{t}
        G2 = np.hstack([0, 0, 0, 1, 0])            # Selector for y_{t}
        G3 = np.hstack([0, 0, 0, 0, 1])            # Selector for martingale
        G4 = np.hstack([0, 0, -g, 0, 0])           # Selector for stationary
        G5 = np.hstack([0, ν, 0, 0, 0])            # Selector for trend
        Gbar = np.vstack([G1, G2, G3, G4, G5])

        # Build H matrix for LSS
        Hbar = np.zeros((1, 1))

        # Build LSS type
        x0 = np.hstack([1, 0, 0, 0, 0])
        S0 = np.zeros((5, 5))
        lss = qe.LinearStateSpace(Abar, Bbar, Gbar, Hbar,
                                  mu_0=x0, Sigma_0=S0)

        return lss

    def additive_decomp(self):
        """
        Return values for the martingale decomposition (Proposition 4.3.3.)
            - ν     : unconditional mean difference in Y
            - H     : coefficient for the (linear) martingale component (kappa_a)
            - g     : coefficient for the stationary component g(x)
            - Y_0   : it should be the function of X_0 (for now set it to 0.0)
        """
        A_res = 1 / (1 - self.A)
        g = self.D * A_res
        H = self.F + self.D * A_res * self.B

```

(continues on next page)

(continued from previous page)

```
        return self.v, H, g

    def multiplicative_decomp(self):
        """
        Return values for the multiplicative decomposition (Example 5.4.4.)
            - v_tilde  : eigenvalue
            - H        : vector for the Jensen term
        """
        v, H, g = self.additive_decomp()
        v_tilde = v + (.5) * H**2

        return v_tilde, H, g

    def loglikelihood_path(self, x, y):
        A, B, D, F = self.A, self.B, self.D, self.F
        T = y.T.size
        FF = F**2
        FFinv = 1 / FF
        temp = y[1:] - y[:-1] - D * x[:-1]
        obs = temp * FFinv * temp
        obssum = np.cumsum(obs)
        scalar = (np.log(FF) + np.log(2 * np.pi)) * np.arange(1, T)

        return (-0.5) * (obssum + scalar)

    def loglikelihood(self, x, y):
        llh = self.loglikelihood_path(x, y)

        return llh[-1]
```

The heavy lifting is done inside the `AMF_LSS_VAR` class.

The following code adds some simple functions that make it straightforward to generate sample paths from an instance of `AMF_LSS_VAR`.

```
def simulate_xy(amf, T):
    "Simulate individual paths."
    foo, bar = amf.lss.simulate(T)
    x = bar[0, :]
    y = bar[1, :]

    return x, y

def simulate_paths(amf, T=150, I=5000):
    "Simulate multiple independent paths."

    # Allocate space
    storeX = np.empty((I, T))
    storeY = np.empty((I, T))

    for i in range(I):
        # Do specific simulation
        x, y = simulate_xy(amf, T)

        # Fill in our storage matrices
        storeX[i, :] = x
```

(continues on next page)

(continued from previous page)

```
        storeY[i, :] = y

    return storeX, storeY

def population_means(amf, T=150):
    # Allocate Space
    xmean = np.empty(T)
    ymean = np.empty(T)

    # Pull out moment generator
    moment_generator = amf.lss.moment_sequence()

    for tt in range (T):
        tmoms = next(moment_generator)
        ymeans = tmoms[1]
        xmean[tt] = ymeans[0]
        ymean[tt] = ymeans[1]

    return xmean, ymean
```

Now that we have these functions in our toolkit, let's apply them to run some simulations.

```
def simulate_martingale_components(amf, T=1000, I=5000):
    # Get the multiplicative decomposition
    v, H, g = amf.multiplicative_decomp()

    # Allocate space
    add_mart_comp = np.empty((I, T))

    # Simulate and pull out additive martingale component
    for i in range(I):
        foo, bar = amf.lss.simulate(T)

        # Martingale component is third component
        add_mart_comp[i, :] = bar[2, :]

    mul_mart_comp = np.exp(add_mart_comp - (np.arange(T) * H**2)/2)

    return add_mart_comp, mul_mart_comp


# Build model
amf_2 = AMF_LSS_VAR(0.8, 0.001, 1.0, 0.01,.005)

amc, mmc = simulate_martingale_components(amf_2, 1000, 5000)

amcT = amc[:, -1]
mmcT = mmc[:, -1]

print("The (min, mean, max) of additive Martingale component in period T is")
print(f"\t ({np.min(amcT)}, {np.mean(amcT)}, {np.max(amcT)})")

print("The (min, mean, max) of multiplicative Martingale component \
in period T is")
print(f"\t ({np.min(mmcT)}, {np.mean(mmcT)}, {np.max(mmcT)})")
```

```
The (min, mean, max) of additive Martingale component in period T is
	(-1.8379907335579106, 0.011040789361757435, 1.4697384727035145)
The (min, mean, max) of multiplicative Martingale component in period T is
	(0.14222026893384476, 1.006753060146832, 3.8858858377907133)
```

Let's plot the probability density functions for $\log \widetilde{M}_t$ for $t = 100, 500, 1000, 10000, 100000$.

Then let's use the plots to investigate how these densities evolve through time.

We will plot the densities of $\log \widetilde{M}_t$ for different values of t.

Note: `scipy.stats.lognorm` expects you to pass the standard deviation first $(tH \cdot H)$ and then the exponent of the mean as a keyword argument `scale` (`scale=np.exp(-t * H2 / 2)`).

- See the documentation here.

This is peculiar, so make sure you are careful in working with the log normal distribution.

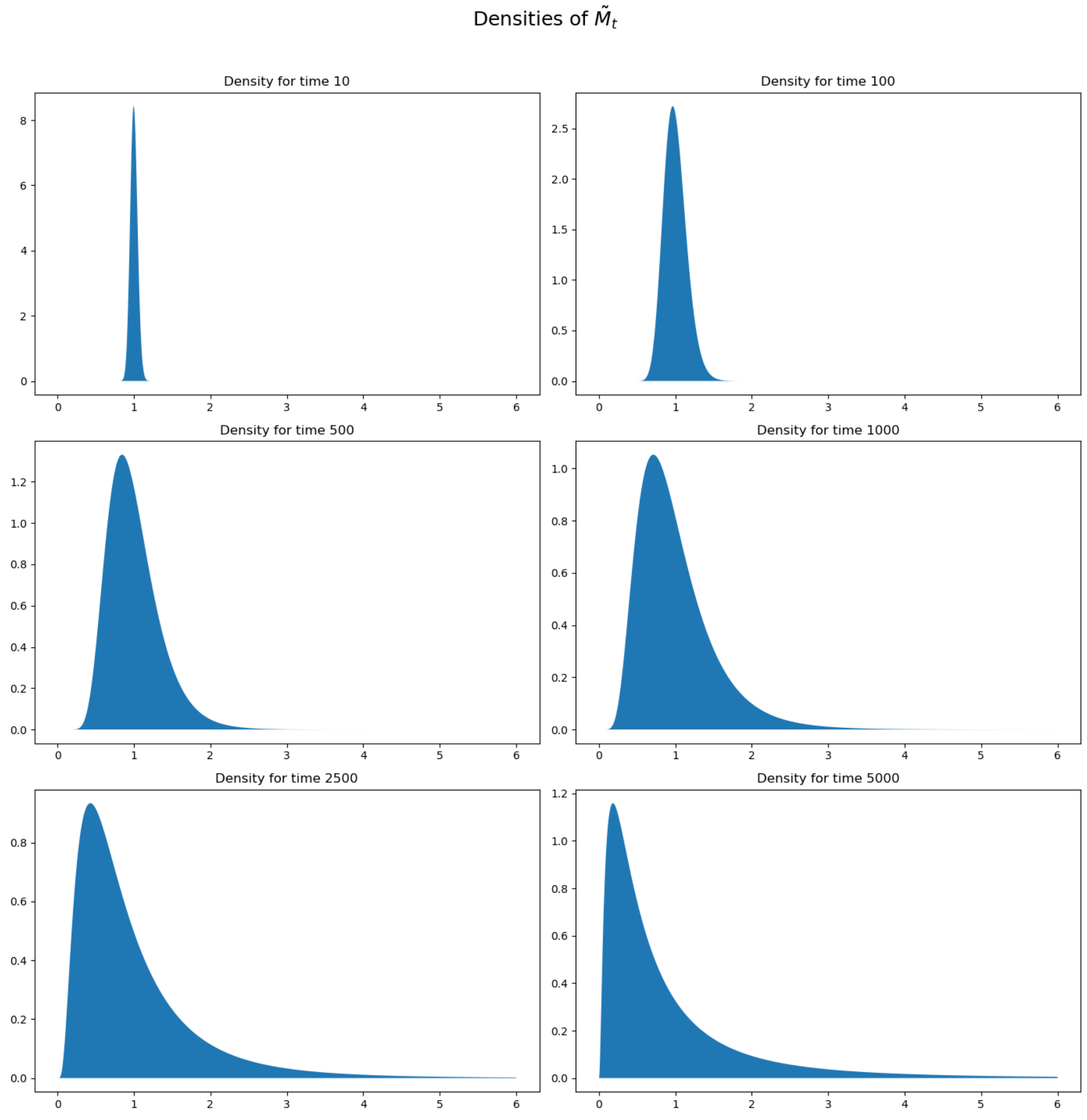
Here is some code that tackles these tasks

```python
def Mtilde_t_density(amf, t, xmin=1e-8, xmax=5.0, npts=5000):

    # Pull out the multiplicative decomposition
    vtilde, H, g = amf.multiplicative_decomp()
    H2 = H*H

    # The distribution
    mdist = lognorm(np.sqrt(t*H2), scale=np.exp(-t*H2/2))
    x = np.linspace(xmin, xmax, npts)
    pdf = mdist.pdf(x)

    return x, pdf


def logMtilde_t_density(amf, t, xmin=-15.0, xmax=15.0, npts=5000):

    # Pull out the multiplicative decomposition
    vtilde, H, g = amf.multiplicative_decomp()
    H2 = H*H

    # The distribution
    lmdist = norm(-t*H2/2, np.sqrt(t*H2))
    x = np.linspace(xmin, xmax, npts)
    pdf = lmdist.pdf(x)

    return x, pdf


times_to_plot = [10, 100, 500, 1000, 2500, 5000]
dens_to_plot = map(lambda t: Mtilde_t_density(amf_2, t, xmin=1e-8, xmax=6.0),
                   times_to_plot)
ldens_to_plot = map(lambda t: logMtilde_t_density(amf_2, t, xmin=-10.0,
                    xmax=10.0), times_to_plot)

fig, ax = plt.subplots(3, 2, figsize=(14, 14))
ax = ax.flatten()
```

(continues on next page)

(continued from previous page)

```
fig.suptitle(r"Densities of $\tilde{M}_t$", fontsize=18, y=1.02)
for (it, dens_t) in enumerate(dens_to_plot):
    x, pdf = dens_t
    ax[it].set_title(f"Density for time {times_to_plot[it]}")
    ax[it].fill_between(x, np.zeros_like(pdf), pdf)

plt.tight_layout()
plt.show()
```



These probability density functions help us understand mechanics underlying the **peculiar property** of our multiplicative martingale

- As T grows, most of the probability mass shifts leftward toward zero.

- For example, note that most mass is near 1 for $T = 10$ or $T = 100$ but most of it is near 0 for $T = 5000$.
- As T grows, the tail of the density of $\widetilde{M}_T$ lengthens toward the right.
- Enough mass moves toward the right tail to keep $E\widetilde{M}_T = 1$ even as most mass in the distribution of $\widetilde{M}_T$ collapses around 0.

17.5.3 Multiplicative Martingale as Likelihood Ratio Process

This lecture studies **likelihood processes** and **likelihood ratio processes**.

A **likelihood ratio process** is a multiplicative martingale with mean unity.

Likelihood ratio processes exhibit the peculiar property that naturally also appears here.

Part V

Risk, Model Uncertainty and Robustness

CHAPTER

EIGHTEEN

RISK AND MODEL UNCERTAINTY

18.1 Overview

As an introduction to one possible approach to modeling **Knightian uncertainty**, this lecture describes static representations of five classes of preferences over risky prospects.

These preference specifications allow us to distinguish **risk** from **uncertainty** along lines proposed by [Knight, 1921].

All five preference specifications incorporate **risk aversion**, meaning displeasure from risks governed by well known probability distributions.

Two of them also incorporate **uncertainty aversion**, meaning dislike of not knowing a probability distribution.

The preference orderings are

- Expected utility preferences
- Constraint preferences
- Multiplier preferences
- Risk-sensitive preferences
- Ex post Bayesian expected utility preferences

This labeling scheme is taken from [Hansen and Sargent, 2001].

Constraint and multiplier preferences express aversion to not knowing a unique probabiity distribution that desribes random outcomes.

Expected utility, risk-sensitive, and ex post Bayesian expected utility preferences all attribute a unique known probability distribution to a decision maker.

We present things in a simple before-and-after one-period setting.

In addition to learning about these preference orderings, this lecture also describes some interesting code for computing and graphing some representations of indifference curves, utility functions, and related objects.

Staring at these indifference curves provides insights into the different preferences.

Watch for the presence of a kink at the 45 degree line for the constraint preference indifference curves.

We begin with some that we'll use to create some graphs.

```
# Package imports
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)
```

(continues on next page)

(continued from previous page)

```
from matplotlib import rc, cm
from mpl_toolkits.mplot3d import Axes3D
from scipy import optimize, stats
from scipy.io import loadmat
from matplotlib.collections import LineCollection
from matplotlib.colors import ListedColormap, BoundaryNorm
from numba import njit
```

18.2 Basic objects

Basic ingredients are

- a set of states of the world
- plans describing outcomes as functions of the state of the world,
- a utility function mapping outcomes into utilities
- either a probability distribution or a **set** of probability distributions over states of the world; and
- a way of measuring a discrepancy between two probability distributions.

In more detail, we'll work with the following setting.

- A finite set of possible **states** $I = \{i = 1, \ldots, I\}$.
- A (consumption) **plan** is a function $c : I \to \mathbb{R}$.
- $u : \mathbb{R} \to \mathbb{R}$ is a **utility function**.
- π is an $I \times 1$ vector of nonnegative **probabilities** over states, with $\pi_i \geq 0, \sum_{i=1}^{I} \pi_i = 1$.
- **Relative entropy** $\textrm{ent}(\pi, \hat{\pi})$ of a probability vector $\hat{\pi}$ with respect to a probability vector π is the expected value of the logarithm of the likelihood ratio $m_i \doteq \left(\frac{\hat{\pi}_i}{\pi_i}\right)$ under distribution $\hat{\pi}$ defined as:

$$\textrm{ent}(\pi, \hat{\pi}) = \sum_{i=1}^{I} \hat{\pi}_i \log\left(\frac{\hat{\pi}_i}{\pi_i}\right) = \sum_{i=1}^{I} \pi_i \left(\frac{\hat{\pi}_i}{\pi_i}\right) \log\left(\frac{\hat{\pi}_i}{\pi_i}\right)$$

or

$$\textrm{ent}(\pi, \hat{\pi}) = \sum_{i=1}^{I} \pi_i m_i \log m_i.$$

Remark: A likelihood ratio m_i is a discrete random variable. For any discrete random variable $\{x_i\}_{i=1}^{I}$, the expected value of x under the $\hat{\pi}_i$ distribution can be represented as the expected value under the π distribution of the product of x_i times the `shock' m_i:

$$\hat{E}x = \sum_{i=1}^{I} x_i \hat{\pi}_i = \sum_{i=1}^{I} m_i x_i \pi_i = Emx,$$

where $\hat{E}$ is the mathematical expectation under the $\hat{\pi}$ distribution and E is the expectation under the π distribution.

Evidently,

$$\hat{E}1 = Em = 1$$

and relative entropy is

$$E m \log m = \hat{E} \log m.$$

In the three figures below, we plot relative entropy from several perspectives.

Our first figure depicts entropy as a function of $\hat{\pi}_1$ when $I = 2$ and $\pi_1 = .5$.

When $\pi_1 \in (0, 1)$, entropy is finite for both $\hat{\pi}_1 = 0$ and $\hat{\pi}_1 = 1$ because $\lim_{x \to 0} x \log x = 0$

However, when $\pi_1 = 0$ or $\pi_1 = 1$, entropy is infinite.

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
```

(continues on next page)

(continued from previous page)

```
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
```

(continues on next page)

(continued from previous page)

```
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
```

(continues on next page)

(continued from previous page)

```
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath, dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop, ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop, ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop, ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex, fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
```

(continues on next page)

(continued from previous page)

```
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 500x300 with 1 Axes>
```

Fig. 18.1: Figure 1

The heat maps in the next two figures vary both $\hat{\pi}_1$ and π_1.

The following figure plots entropy.

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
```

(continues on next page)

(continued from previous page)

```
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
 ↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
 ↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
 ↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
 ↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
```

(continues on next page)

(continued from previous page)

```
     154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3034, in _AxesBase.draw(self, renderer)
   3031     for spine in self.spines.values():
   3032         artists.remove(spine)
-> 3034 self._update_title_position(renderer)
   3036 if not self.axison:
   3037     for _axis in self._axis_map.values():

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:2978, in _AxesBase._update_title_position(self, renderer)
   2976 top = max(top, bb.ymax)
   2977 if title.get_text():
-> 2978     ax.yaxis.get_tightbbox(renderer)  # update offsetText
   2979     if ax.yaxis.offsetText.get_text():
   2980         bb = ax.yaxis.offsetText.get_tightbbox(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1336, in Axis.get_tightbbox(self, renderer, for_layout_only)
   1333     renderer = self.figure._get_renderer()
   1334 ticks_to_draw = self._update_ticks()
-> 1336 self._update_label_position(renderer)
   1338 # go back to just this axis's tick labels
   1339 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:2609, in YAxis._update_label_position(self, renderer)
   2605     return
   2607 # get bounding boxes for this axis and any siblings
   2608 # that have been set by `fig.align_ylabels()`
-> 2609 bboxes, bboxes2 = self._get_tick_boxes_siblings(renderer=renderer)
   2610 x, y = self.label.get_position()
   2611 if self.label_position == 'left':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:2161, in Axis._get_tick_boxes_siblings(self, renderer)
   2159 axis = ax._axis_map[name]
   2160 ticks_to_draw = axis._update_ticks()
-> 2161 tlb, tlb2 = axis._get_ticklabel_bboxes(ticks_to_draw, renderer)
   2162 bboxes.extend(tlb)
   2163 bboxes2.extend(tlb2)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
```

(continues on next page)

(continued from previous page)

```
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
```

(continues on next page)

(continued from previous page)

```
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1000x800 with 2 Axes>
```

The next figure plots the logarithm of entropy.

```
3.8205752275831846
```

```
/tmp/ipykernel_2218/3759713737.py:2: RuntimeWarning: divide by zero encountered in␣
↪log
```

(continues on next page)

(continued from previous page)

```
plt.pcolormesh(x, y, np.log(ent_vals_mat.T), shading='gouraud', cmap='seismic')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:
```

(continues on next page)

(continued from previous page)

```
RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
```

(continues on next page)

(continued from previous page)

```
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3034, in _AxesBase.draw(self, renderer)
   3031     for spine in self.spines.values():
   3032         artists.remove(spine)
-> 3034 self._update_title_position(renderer)
   3036 if not self.axison:
   3037     for _axis in self._axis_map.values():

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:2978, in _AxesBase._update_title_position(self, renderer)
   2976 top = max(top, bb.ymax)
   2977 if title.get_text():
-> 2978     ax.yaxis.get_tightbbox(renderer)  # update offsetText
   2979     if ax.yaxis.offsetText.get_text():
   2980         bb = ax.yaxis.offsetText.get_tightbbox(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1336, in Axis.get_tightbbox(self, renderer, for_layout_only)
   1333     renderer = self.figure._get_renderer()
   1334 ticks_to_draw = self._update_ticks()
-> 1336 self._update_label_position(renderer)
   1338 # go back to just this axis's tick labels
   1339 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:2609, in YAxis._update_label_position(self, renderer)
```

(continues on next page)

(continued from previous page)

```
   2605     return
   2607 # get bounding boxes for this axis and any siblings
   2608 # that have been set by `fig.align_ylabels()`
-> 2609 bboxes, bboxes2 = self._get_tick_boxes_siblings(renderer=renderer)
   2610 x, y = self.label.get_position()
   2611 if self.label_position == 'left':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:2161, in Axis._get_tick_boxes_siblings(self, renderer)
   2159 axis = ax._axis_map[name]
   2160 ticks_to_draw = axis._update_ticks()
-> 2161 tlb, tlb2 = axis._get_ticklabel_bboxes(ticks_to_draw, renderer)
   2162 bboxes.extend(tlb)
   2163 bboxes2.extend(tlb2)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
```

(continues on next page)

(continued from previous page)

```
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1000x800 with 2 Axes>
```

18.3 Five preference specifications

We describe five types of preferences over plans.

- Expected utility preferences
- Constraint preferences
- Multiplier preferences
- Risk-sensitive preferences
- Ex post Bayesian expected utility preferences

Expected utility, risk-sensitive, and ex post Bayesian prefernces are each cast in terms of a unique probability distribution, so they can express risk-aversion, but not model ambiguity aversion.

Multiplier and constraint prefernces both express aversion to concerns about model misppecification, i.e., model uncertainty; both are cast in terms of a set or sets of probability distributions.

- The set of distributions expresses the decision maker's ambiguity about the probability model.
- Minimization over probability distributions expresses his aversion to ambiguity.

18.4 Expected utility

A decision maker is said to have **expected utility preferences** when he ranks plans c by their expected utilities

$$\sum_{i=1}^{I} u(c_i)\pi_i, \tag{18.1}$$

where u is a unique utility function and π is a unique probability measure over states.

- A known π expresses risk.
- Curvature of u expresses risk aversion.

18.5 Constraint preferences

A decision maker is said to have **constraint preferences** when he ranks plans c according to

$$\min_{\{m_i \geq 0\}_{i=1}^{I}} \sum_{i=1}^{I} m_i \pi_i u(c_i) \tag{18.2}$$

subject to

$$\sum_{i=1}^{I} \pi_i m_i \log m_i \leq \eta \tag{18.3}$$

and

$$\sum_{i=1}^{I} \pi_i m_i = 1. \tag{18.4}$$

In (18.3), $\eta \geq 0$ defines an entropy ball of probability distributions $\hat{\pi} = m\pi$ that surround a baseline distribution π.

As noted earlier, $\sum_{i=1}^{I} m_i \pi_i u(c_i)$ is the expected value of $u(c)$ under a twisted probability distribution $\{\hat{\pi}_i\}_{i=1}^{I} = \{m_i \pi_i\}_{i=1}^{I}$.

Larger values of the entropy constraint η indicate more apprehension about the baseline probability distribution $\{\pi_i\}_{i=1}^{I}$.

Following [Hansen and Sargent, 2001] and [Hansen and Sargent, 2008], we call minimization problem (18.2) subject to (18.3) and(18.4) a **constraint problem**.

To find minimizing probabilities, we form a Lagrangian

$$L = \sum_{i=1}^{I} m_i \pi_i u(c_i) + \tilde{\theta}\Big[\sum_{i=1}^{I} \pi_i m_i \log m_i - \eta\Big] \tag{18.5}$$

where $\tilde{\theta} \geq 0$ is a Lagrange multiplier associated with the entropy constraint.

Subject to the additional constraint that $\sum_{i=1}^{I} m_i \pi_i = 1$, we want to minimize (18.5) with respect to $\{m_i\}_{i=1}^{I}$ and to maximize it with respect to $\tilde{\theta}$.

The minimizing probability distortions (likelihood ratios) are

$$\tilde{m}_i(c; \tilde{\theta}) = \frac{\exp(-u(c_i)/\tilde{\theta})}{\sum_j \pi_j \exp(-u(c_j)/\tilde{\theta})}. \tag{18.6}$$

To compute the Lagrange multiplier $\tilde{\theta}(c, \eta)$, we must solve

$$\sum_i \pi_i \tilde{m}_i(c; \tilde{\theta}) \log(\tilde{m}_i(c; \tilde{\theta})) = \eta$$

or

$$\sum_i \pi_i \frac{\exp(-u(c_i)/\tilde{\theta})}{\sum_j \pi_j \exp(-u(c_j)/\tilde{\theta})} \log\left[\frac{\exp(-u(c_i)/\tilde{\theta})}{\sum_j \pi_j \exp(-u(c_j)/\tilde{\theta})}\right] = \eta \tag{18.7}$$

for $\tilde{\theta} = \tilde{\theta}(c; \eta)$.

For a fixed η, the $\tilde{\theta}$ that solves equation (18.7) is evidently a function of the consumption plan c.

With $\tilde{\theta}(c; \eta)$ in hand we can obtain worst-case probabilities as functions $\pi_i \tilde{m}_i(c; \eta)$ of η.

The **indirect (expected) utility function** under constraint preferences is

$$\sum_{i=1}^I \pi_i \tilde{m}_i(c_i; \eta) u(c_i) = \sum_{i=1}^I \pi_i \left[\frac{\exp(-\tilde{\theta}^{-1} u(c_i))}{\sum_{j=1}^I \exp(-\tilde{\theta}^{-1} u(c_j)) \pi_j}\right] u(c_i). \tag{18.8}$$

Entropy evaluated at the minimizing probability distortion (18.6) equals $E\tilde{m} \log \tilde{m}$ or

$$\begin{aligned} & \sum_{i=1}^I \left[\frac{\exp(-\tilde{\theta}^{-1} u(c_i))}{\sum_{j=1}^I \exp(-\tilde{\theta}^{-1} u(c_j)) \pi_j}\right] \times \\ & \left\{-\tilde{\theta}^{-1} u(c_i) + \log\left(\sum_{j=1}^I \exp(-\tilde{\theta}^{-1} u(c_j)) \pi_j\right)\right\} \pi_i \\ = \quad & -\tilde{\theta}^{-1} \sum_{i=1}^I \pi_i \left[\frac{\exp(-\tilde{\theta}^{-1} u(c_i))}{\sum_{j=1}^I \exp(-\tilde{\theta}^{-1} u(c_j)) \pi_j}\right] u(c_i) \\ & + \log\left(\sum_{j=1}^I \exp(-\tilde{\theta}^{-1} u(c_j)) \pi_j\right). \end{aligned} \tag{18.9}$$

Expression (18.9) implies that

$$\begin{aligned} -\tilde{\theta} \log\left(\sum_{j=1}^I \exp(-\tilde{\theta}^{-1} u(c_j)) \pi_j\right) = & \sum_{i=1}^I \pi_i \left[\frac{\exp(-\tilde{\theta}^{-1} u(c_i))}{\sum_{j=1}^I \exp(-\tilde{\theta}^{-1} u(c_j)) \pi_j}\right] u(c_i) \\ & + \tilde{\theta}(c; \eta) \sum_{i=1}^I \log \tilde{m}_i(c; \eta) \tilde{m}_i(c; \eta) \pi_i, \end{aligned} \tag{18.10}$$

where the last term is $\tilde{\theta}$ times the entropy of the worst-case probability distribution.

18.6 Multiplier preferences

A decision maker is said to have **multiplier preferences** when he ranks consumption plans c according to

$$\mathsf{T}u(c) \doteq \min_{\{m_i \geq 0\}_{i=1}^I} \sum_{i=1}^I \pi_i m_i [u(c_i) + \theta \log m_i] \tag{18.11}$$

where minimization is subject to

$$\sum_{i=1}^I \pi_i m_i = 1.$$

Here $\theta \in (\underline{\theta}, +\infty)$ is a 'penalty parameter' that governs a 'cost' to an 'evil alter ego' who distorts probabilities by choosing $\{m_i\}_{i=1}^I$.

Lower values of the penalty parameter θ express more apprehension about the baseline probability distribution π.

Following [Hansen and Sargent, 2001] and [Hansen and Sargent, 2008], we call the minimization problem on the right side of (18.11) a **multiplier problem**.

The minimizing probability distortion that solves the multiplier problem is

$$\hat{m}_i(c;\theta) = \frac{\exp(-u(c_i)/\theta)}{\sum_j \pi_j \exp(-u(c_j)/\theta)}. \tag{18.12}$$

We can solve

$$\sum_i \pi_i \frac{\exp(-u(c_i)/\theta)}{\sum_j \pi_j \exp(-u(c_j)/\theta)} \log\left[\frac{\exp(-u(c_i)/\theta)}{\sum_j \pi_j \exp(-u(c_j)/\theta)}\right] = \tilde{\eta} \tag{18.13}$$

to find an entropy level $\tilde{\eta}(c;\theta)$ associated with multiplier preferences with penalty parameter θ and allocation c.

For a fixed θ, the $\tilde{\eta}$ that solves equation (18.13) is a function of the consumption plan c

The forms of expressions (18.6) and (18.12) are identical, but the Lagrange multiplier $\tilde{\theta}$ appears in (18.6), while the penalty parameter θ appears in (18.12).

Formulas (18.6) and (18.12) show that worst-case probabilities are **context specific** in the sense that they depend on both the utility function u and the consumption plan c.

If we add θ times entropy under the worst-case model to expected utility under the worst-case model, we find that the **indirect expected utility function** under multiplier preferences is

$$-\theta \log\left(\sum_{j=1}^I \exp(-\theta^{-1}u(c_j))\pi_j\right). \tag{18.14}$$

18.7 Risk-sensitive preferences

Substituting $\hat{m}_i$ into $\sum_{i=1}^I \pi_i \hat{m}_i[u(c_i) + \theta \log \hat{m}_i]$ gives the indirect utility function

$$\mathsf{T}u(c) \doteq -\theta \log \sum_{i=1}^I \pi_i \exp(-u(c_i)/\theta). \tag{18.15}$$

Here $\mathsf{T}u$ in (18.15) is the **risk-sensitivity** operator of [Jacobson, 1973], [Whittle, 1981], and [Whittle, 1990].

It defines a **risk-sensitive** preference ordering over plans c.

Because it is not linear in utilities $u(c_i)$ and probabilities π_i, it is said not to be separable across states.

Because risk-sensitive preferences use a unique probability distribution, they apparently express no model distrust or ambiguity.

Instead, they make an additional adjustment for risk-aversion beyond that embedded in the curvature of u.

For $I = 2, c_1 = 2, c_2 = 1, u(c) = \ln c$, the following figure plots the risk-sensitive criterion $\mathsf{T}u(c)$ defined in (18.15) as a function of π_1 for values of θ of 100 and .6.

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
```

(continues on next page)

(continued from previous page)

```
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
```

(continues on next page)

(continued from previous page)

```
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
```

(continues on next page)

(continued from previous page)

```
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
```

(continues on next page)

(continued from previous page)

```
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪formatters.py:340, in BaseFormatter.__call__(self, obj)
```

(continues on next page)

(continued from previous page)

```
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
```

(continues on next page)

(continued from previous page)

```
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
```

(continues on next page)

(continued from previous page)

```
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
```

(continues on next page)

(continued from previous page)

```
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1000x800 with 1 Axes>
```

For large values of θ, $\mathsf{T}u(c)$ is approximately linear in the probability π_1, but for lower values of θ, $\mathsf{T}u(c)$ has considerable curvature as a function of π_1.

Under expected utility, i.e., $\theta = +\infty$, $\mathsf{T}u(c)$ is linear in π_1, but it is convex as a function of π_1 when $\theta < +\infty$.

The two panels in the next figure below can help us to visualize the extra adjustment for risk that the risk-sensitive operator entails.

This will help us understand how the **T** transformation works by envisioning what function is being averaged.

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
```

(continues on next page)

(continued from previous page)

```
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
```

(continues on next page)

(continued from previous page)

```
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3034, in _AxesBase.draw(self, renderer)
   3031     for spine in self.spines.values():
   3032         artists.remove(spine)
-> 3034 self._update_title_position(renderer)
   3036 if not self.axison:
   3037     for _axis in self._axis_map.values():

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:2978, in _AxesBase._update_title_position(self, renderer)
   2976 top = max(top, bb.ymax)
   2977 if title.get_text():
-> 2978     ax.yaxis.get_tightbbox(renderer)  # update offsetText
   2979     if ax.yaxis.offsetText.get_text():
   2980         bb = ax.yaxis.offsetText.get_tightbbox(renderer)
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1336, in Axis.get_tightbbox(self, renderer, for_layout_only)
   1333     renderer = self.figure._get_renderer()
   1334 ticks_to_draw = self._update_ticks()
-> 1336 self._update_label_position(renderer)
   1338 # go back to just this axis's tick labels
   1339 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:2609, in YAxis._update_label_position(self, renderer)
   2605     return
   2607 # get bounding boxes for this axis and any siblings
   2608 # that have been set by `fig.align_ylabels()`
-> 2609 bboxes, bboxes2 = self._get_tick_boxes_siblings(renderer=renderer)
   2610 x, y = self.label.get_position()
   2611 if self.label_position == 'left':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:2161, in Axis._get_tick_boxes_siblings(self, renderer)
   2159 axis = ax._axis_map[name]
   2160 ticks_to_draw = axis._update_ticks()
-> 2161 tlb, tlb2 = axis._get_ticklabel_bboxes(ticks_to_draw, renderer)
   2162 bboxes.extend(tlb)
   2163 bboxes2.extend(tlb2)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
```

(continues on next page)

(continued from previous page)

```
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
```

(continues on next page)

(continued from previous page)

```
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
 ↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
 ↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
 ↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
 ↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
```

(continues on next page)

(continued from previous page)

```
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3034, in _AxesBase.draw(self, renderer)
   3031     for spine in self.spines.values():
   3032         artists.remove(spine)
-> 3034 self._update_title_position(renderer)
   3036 if not self.axison:
   3037     for _axis in self._axis_map.values():

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:2978, in _AxesBase._update_title_position(self, renderer)
   2976 top = max(top, bb.ymax)
   2977 if title.get_text():
-> 2978     ax.yaxis.get_tightbbox(renderer)  # update offsetText
   2979     if ax.yaxis.offsetText.get_text():
   2980         bb = ax.yaxis.offsetText.get_tightbbox(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1336, in Axis.get_tightbbox(self, renderer, for_layout_only)
   1333     renderer = self.figure._get_renderer()
   1334 ticks_to_draw = self._update_ticks()
-> 1336 self._update_label_position(renderer)
   1338 # go back to just this axis's tick labels
   1339 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:2609, in YAxis._update_label_position(self, renderer)
   2605     return
   2607 # get bounding boxes for this axis and any siblings
   2608 # that have been set by `fig.align_ylabels()`
-> 2609 bboxes, bboxes2 = self._get_tick_boxes_siblings(renderer=renderer)
   2610 x, y = self.label.get_position()
   2611 if self.label_position == 'left':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:2161, in Axis._get_tick_boxes_siblings(self, renderer)
   2159 axis = ax._axis_map[name]
   2160 ticks_to_draw = axis._update_ticks()
-> 2161 tlb, tlb2 = axis._get_ticklabel_bboxes(ticks_to_draw, renderer)
   2162 bboxes.extend(tlb)
   2163 bboxes2.extend(tlb2)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
```

(continues on next page)

(continued from previous page)

```
   1316         for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
```

(continues on next page)

(continued from previous page)

```
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1600x600 with 2 Axes>
```

The panel on the right portrays how the transformation $\exp\left(\frac{-u(c)}{\theta}\right)$ sends $u(c)$ to a new function by (i) flipping the sign, and (ii) increasing curvature in proportion to θ.

In the left panel, the red line is our tool for computing the mathematical expectation for different values of π.

The green lot indicates the mathematical expectation of $\exp\left(\frac{-u(c)}{\theta}\right)$ when $\pi = .5$.

Notice that the distance between the green dot and the curve is greater in the transformed space than the original space as a result of additional curvature.

The inverse transformation $\theta \log E\left[\exp\left(\frac{-u(c)}{\theta}\right)\right]$ generates the green dot on the left panel that constitutes the risk-sensitive utility index.

The gap between the green dot and the red line on the left panel measures the additional adjustment for risk that risk-sensitive preferences make relative to plain vanilla expected utility preferences.

18.7.1 Digression on moment generating functions

The risk-sensitivity operator T is intimately connected to a moment generating function.

In particular, a principal constinuent of the T operator, namely,

$$E \exp(-u(c_i)/\theta) = \sum_{i=1}^{I} \pi_i \exp(-u(c_i)/\theta)$$

is evidently a **moment generating function** for the random variable $u(c_i)$, while

$$g(\theta^{-1}) \doteq \log \sum_{i=1}^{I} \pi_i \exp(-u(c_i)/\theta)$$

is a **cumulant generating function**,

$$g(\theta^{-1}) = \sum_{j=1}^{\infty} \kappa_j \frac{(-\theta^{-1})^j}{j!}.$$

where κ_j is the jth cumulant of the random variable $u(c)$.

Then

$$\mathsf{T}u(c) = -\theta g(\theta^{-1}) = -\theta \sum_{j=1}^{\infty} \kappa_j \frac{(-\theta^{-1})^j}{j!}.$$

In general, when $\theta < +\infty$, $\mathsf{T}u(c)$ depends on cumulants of all orders.

These statements extend to cases with continuous probability distributions for c and therefore for $u(c)$.

For the special case $u(c) \sim \mathcal{N}(\mu_u, \sigma_u^2)$, $\kappa_1 = \mu_u, \kappa_2 = \sigma_u^2$, and $\kappa_j = 0 \ \forall j \geq 3$, so

$$\mathsf{T}u(c) = \mu_u - \frac{1}{2\theta}\sigma_u^2, \tag{18.16}$$

which becomes expected utility μ_u when $\theta^{-1} = 0$.

The right side of equation (18.16) is a special case of **stochastic differential utility** preferences in which consumption plans are ranked not just by their expected utilities μ_u but also the variances σ_u^2 of their expected utilities.

18.8 Ex post Bayesian preferences

A decision maker is said to have **ex post Bayesian preferences** when he ranks consumption plans according to the expected utility function

$$\sum_i \hat{\pi}_i(c^*)u(c_i) \tag{18.17}$$

where $\hat{\pi}(c^*)$ is the worst-case probability distribution associated with multiplier or constraint preferences evaluated at a particular consumption plan $c^* = \{c_i^*\}_{i=1}^I$.

At c^*, an ex post Bayesian's indifference curves are tangent to those for multiplier and constraint preferences with appropriately chosen θ and η, respectively.

18.9 Comparing preferences

For the special case in which $I = 2$, $c_1 = 2, c_2 = 1$, $u(c) = \ln c$, and $\pi_1 = .5$, the following two figures depict how worst-case probabilities are determined under constraint and multiplier preferences, respectively.

The first figure graphs entropy as a function of $\hat{\pi}_1$.

It also plots expected utility under the twisted probability distribution, namely, $\hat{E}u(c) = u(c_2) + \hat{\pi}_1(u(c_1) - u(c_2))$, which is evidently a linear function of $\hat{\pi}_1$.

The entropy constraint $\sum_{i=1}^{I} \pi_i m_i \log m_i \leq \eta$ implies a convex set $\hat{\Pi}_1$ of $\hat{\pi}_1$'s that constrains the adversary who chooses $\hat{\pi}_1$, namely, the set of $\hat{\pi}_1$'s for which the entropy curve lies below the horizontal dotted line at an entropy level of $\eta = .25$.

Unless $u(c_1) = u(c_2)$, the $\hat{\pi}_1$ that minimizes $\hat{E}u(c)$ is at the boundary of the set $\hat{\Pi}_1$.

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
```

(continues on next page)

(continued from previous page)

```
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
```

(continues on next page)

(continued from previous page)

```
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
```

(continues on next page)

(continued from previous page)

```
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
```

(continues on next page)

(continued from previous page)

```
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
```

(continues on next page)

(continued from previous page)

```
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
```

(continues on next page)

(continued from previous page)

```
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
```

(continues on next page)

(continued from previous page)

```
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
```

(continues on next page)

(continued from previous page)

```
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1200x800 with 1 Axes>
```

The next figure shows the function $\sum_{i=1}^{I} \pi_i m_i [u(c_i) + \theta \log m_i]$ that is to be minimized in the multiplier problem. The argument of the function is $\hat{\pi}_1 = m_1 \pi_1$.

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
```

(continues on next page)

(continued from previous page)

```
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
```

(continues on next page)

(continued from previous page)

```
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
```

(continues on next page)

(continued from previous page)

```
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
```

(continues on next page)

(continued from previous page)

```
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
```

(continues on next page)

(continued from previous page)

```
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
```

(continues on next page)

(continued from previous page)

```
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
```

(continues on next page)

(continued from previous page)

```
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
```

(continues on next page)

(continued from previous page)

```
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
```

(continues on next page)

(continued from previous page)

```
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1200x800 with 1 Axes>
```

Evidently, from this figure and also from formula (18.12), lower values of θ lead to lower, and thus more distorted, minimizing values of $\hat{\pi}_1$.

The figure indicates how one can construct a Lagrange multiplier $\tilde{\theta}$ associated with a given entropy constraint η and a given consumption plan.

Thus, to draw the figure, we set the penalty parameter for multiplier preferences θ so that the minimizing $\hat{\pi}_1$ equals the minimizing $\hat{\pi}_1$ for the constraint problem from the previous figure.

The penalty parameter $\theta = .42$ also equals the Lagrange multiplier $\tilde{\theta}$ on the entropy constraint for the constraint preferences depicted in the previous figure because the $\hat{\pi}_1$ that minimizes the asymmetric curve associated with penalty parameter $\theta = .42$ is the same $\hat{\pi}_1$ associated with the intersection of the entropy curve and the entropy constraint dashed vertical line.

18.10 Risk aversion and misspecification aversion

All five types of preferences use curvature of u to express risk aversion.

Constraint preferences express **concern about misspecification** or **ambiguity** for short with a positive η that circumscribes an entropy ball around an approximating probability distribution π, and *aversion aversion to model misspecification* through minimization with respect to a likelihood ratio m.

Multiplier preferences express misspecification concerns with a parameter $\theta < +\infty$ that penalizes deviations from the approximating model as measured by relative entropy, and they express aversion to misspecification concerns with minimization over a probability distortion m.

By penalizing minimization over the likelihood ratio m, a decrease in θ represents an **increase** in ambiguity (or what [Knight, 1921] called uncertainty) about the specification of the baseline approximating model $\{\pi_i\}_{i=1}^I$.

Formulas (18.6) assert that the decision maker acts as if he is pessimistic relative to an approximating model π.

It expresses what [Bucklew, 2004] [p. 27] calls a statistical version of *Murphy's law*:

> **The probability of anything happening is in inverse ratio to its desirability.**

The minimizing likelihood ratio $\hat{m}$ slants worst-case probabilities $\hat{\pi}$ exponentially to increase probabilities of events that give lower utilities.

As expressed by the value function bound (18.19) to be displayed below, the decision maker uses **pessimism** instrumentally to protect himself against model misspecification.

The penalty parameter θ for multipler preferences or the entropy level η that determines the Lagrange multiplier $\tilde{\theta}$ for constraint preferences controls how adversely the decision maker exponentially slants probabilities.

A decision rule is said to be **undominated** in the sense of Bayesian decision theory if there exists a probability distribution π for which it is optimal.

A decision rule is said to be **admissible** if it is undominated.

[Hansen and Sargent, 2008] use ex post Bayesian preferences to show that robust decision rules are undominated and therefore admissible.

18.11 Indifference curves

Indifference curves illuminate how concerns about robustness affect asset pricing and utility costs of fluctuations. For $I = 2$, the slopes of the indifference curves for our five preference specifications are

- Expected utility:

$$\frac{dc_2}{dc_1} = -\frac{\pi_1}{\pi_2}\frac{u'(c_1)}{u'(c_2)}$$

- Constraint and ex post Bayesian preferences:

$$\frac{dc_2}{dc_1} = -\frac{\hat{\pi}_1}{\hat{\pi}_2}\frac{u'(c_1)}{u'(c_2)}$$

where $\hat{\pi}_1, \hat{\pi}_2$ are the minimizing probabilities computed from the worst-case distortions (18.6) from the constraint problem at (c_1, c_2).

- Multiplier and risk-sensitive preferences:

$$\frac{dc_2}{dc_1} = -\frac{\pi_1}{\pi_2} \frac{\exp(-u(c_1)/\theta)}{\exp(-u(c_2)/\theta)} \frac{u'(c_1)}{u'(c_2)}$$

When $c_1 > c_2$, the exponential twisting formula (18.12) implies that $\hat{\pi}_1 < \pi_1$, which in turn implies that the indifference curves through (c_1, c_2) for both constraint and multiplier preferences are flatter than the indifference curve associated with expected utility preferences.

As we shall see soon when we discuss state price deflators, this gives rise to higher estimates of prices of risk.

For an example with $u(c) = \ln c$, $I = 2$, and $\pi_1 = .5$, the next two figures show indifference curves for expected utility, multiplier, and constraint preferences.

The following figure shows indifference curves going through a point along the 45 degree line.

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
```

(continues on next page)

(continued from previous page)

```
   1033                             gid, gids, uid, umask,
   1034                             start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
```

(continues on next page)

(continued from previous page)

```
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
```

(continues on next page)

(continued from previous page)

```
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
 ↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
 ↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
 ↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
 ↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
 ↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
 ↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
 ↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
```

(continues on next page)

(continued from previous page)

```
   1033                             gid, gids, uid, umask,
   1034                             start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
 ↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
 ↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
 ↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
 ↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
 ↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
 ↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
 ↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
```

(continues on next page)

(continued from previous page)

```
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
```

(continues on next page)

(continued from previous page)

```
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1600x800 with 2 Axes>
```

Kink at 45 degree line

Notice the kink in the indifference curve for constraint preferences at the 45 degree line.

To understand the source of the kink, consider how the Lagrange multiplier and worst-case probabilities vary with the consumption plan under constraint preferences.

For fixed η, a given plan c, and a utility function increasing in c, worst case probabilities are **fixed numbers** $\hat{\pi}_1 < .5$ when $c_1 > c_2$ and $\hat{\pi}_1 > .5$ when $c_2 > c_1$.

This pattern makes the Lagrange multiplier $\tilde{\theta}$ vary discontinuously at $\hat{\pi}_1 = .5$.

The discontinuity in the worst case $\hat{\pi}_1$ at the 45 degree line accounts for the kink at the 45 degree line in an indifference curve for constraint preferences associated with a given positive entropy constraint η.

The code for generating the preceding figure is somewhat intricate we formulate a root finding problem for finding indifference curves.

Here is a brief literary description of the method we use.

Parameters

- Consumption bundle $c = (1, 1)$
- Penalty parameter $\theta = 2$
- Utility function $u = \log$

- Probability vector $\pi = (0.5, 0.5)$

Algorithm:

- Compute $\bar{u} = \pi_1 u(c_1) + \pi_2 u(c_2)$
- Given values for c_1, solve for values of c_2 such that $\bar{u} = u(c_1, c_2)$:
  - Expected utility: $c_{2,EU} = u^{-1}\left(\frac{\bar{u} - \pi_1 u(c_1)}{\pi_2}\right)$
  - Multiplier preferences: solve $\bar{u} - \sum_i \pi_i \frac{\exp\left(\frac{-u(c_i)}{\theta}\right)}{\sum_j \exp\left(\frac{-u(c_j)}{\theta}\right)}\left(u(c_i) + \theta \log\left(\frac{\exp\left(\frac{-u(c_i)}{\theta}\right)}{\sum_j \exp\left(\frac{-u(c_j)}{\theta}\right)}\right)\right) = 0$ numerically
  - Constraint preference: solve $\bar{u} - \sum_i \pi_i \frac{\exp\left(\frac{-u(c_i)}{\theta^*}\right)}{\sum_j \exp\left(\frac{-u(c_j)}{\theta^*}\right)} u(c_i) = 0$ numerically where θ^* solves $\sum_i \pi_i \frac{\exp\left(\frac{-u(c_i)}{\theta^*}\right)}{\sum_j \exp\left(\frac{-u(c_j)}{\theta^*}\right)} \log\left(\frac{\exp\left(\frac{-u(c_i)}{\theta^*}\right)}{\sum_j \exp\left(\frac{-u(c_j)}{\theta^*}\right)}\right) - \eta = 0$ numerically.

Remark: It seems that the constraint problem is hard to solve in its original form, i.e. by finding the distorting measure that minimizes the expected utility.

It seems that viewing equation (18.7) as a root finding problem works much better.

But notice that equation (18.7) does not always have a solution.

Under $u = \log$, $c_1 = c_2 = 1$, we have:

$$\sum_i \pi_i \frac{\exp\left(\frac{-u(c_i)}{\tilde{\theta}}\right)}{\sum_j \pi_j \exp\left(\frac{-u(c_j)}{\tilde{\theta}}\right)} \log\left(\frac{\exp\left(\frac{-u(c_i)}{\tilde{\theta}}\right)}{\sum_j \pi_j \exp\left(\frac{-u(c_j)}{\tilde{\theta}}\right)}\right) = 0$$

Conjecture: when our numerical method fails it because the derivative of the objective doesn't exist for our choice of parameters.

Remark: It is tricky to get the algorithm to work properly for all values of c_1. In particular, parameters were chosen with graduate student descent.

Tangent indifference curves off 45 degree line

For a given η and a given allocatin (c_1, c_2) off the 45 degree line, by solving equations (18.7) and (18.13), we can find $\tilde{\theta}(\eta, c)$ and $\tilde{\eta}(\theta, c)$ that make indifference curves for multiplier and constraint preferences be tangent to one another.

The following figure shows indifference curves for multiplier and constraint preferences through a point off the 45 degree line, namely, $(c(1), c(2)) = (3, 1)$, at which η and θ are adjusted to render the indifference curves for constraint and multiplier preferences tangent.

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
```

(continues on next page)

(continued from previous page)

```
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
```

(continues on next page)

(continued from previous page)

```
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
```

(continues on next page)

(continued from previous page)

```
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
```

(continues on next page)

(continued from previous page)

```
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
```

(continues on next page)

(continued from previous page)

```
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
```

(continues on next page)

(continued from previous page)

```
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1600x800 with 2 Axes>
```

Note that all three lines of the left graph intersect at (1, 3). While the intersection at (3, 1) is hard-coded, the intersection at (1,3) arises from the computation, which confirms that the code seems to be working properly.

As we move along the (kinked) indifference curve for the constraint preferences for a given η, the worst-case probabilities remain constant, but the Lagrange multiplier $\tilde{\theta}$ on the entropy constraint $\sum_{i=1}^{I} m_i \log m_i \leq \eta$ varies with (c_1, c_2).

As we move along the (smooth) indifference curve for the multiplier preferences for a given penalty parameter θ, the implied entropy $\tilde{\eta}$ from equation (18.13) and the worst-case probabilities both change with (c_1, c_2).

For constraint preferences, there is a kink in the indifference curve.

For ex post Bayesian preferences, there are effectively two sets of indifference curves depending on which side of the 45 degree line the (c_1, c_2) endowment point sits.

There are two sets of indifference curves because, while the worst-case probabilities differ above and below the 45 degree line, the idea of ex post Bayesian preferences is to use a *single* probability distribution to compute expected utilities for all consumption bundles.

Indifference curves through point $(c_1, c_2) = (3, 1)$ for expected logarithmic utility (less curved smooth line), multiplier (more curved line), constraint (solid line kinked at 45 degree line), and *ex post* Bayesian (dotted lines) preferences. The worst-case probability $\hat{\pi}_1 < .5$ when $c_1 = 3 > c_2 = 1$ and $\hat{\pi}_1 > .5$ when $c_1 = 1 < c_2 = 3$.

18.12 State price deflators

Concerns about model uncertainty boost prices of risk that are embedded in state-price deflators. With complete markets, let q_i be the price of consumption in state i.

The budget set of a representative consumer having endowment $\bar{c} = \{\bar{c}_i\}_{i=1}^{I}$ is expressed by $\sum_i^I q_i(c_i - \bar{c}_i) \leq 0$.

When a representative consumer has multiplier preferences, the state prices are

$$q_i = \pi_i \hat{m}_i u'(\bar{c}_i) = \pi_i \left(\frac{\exp(-u(\bar{c}_i)/\theta)}{\sum_j \pi_j \exp(-u(\bar{c}_j)/\theta)} \right) u'(\bar{c}_i). \tag{18.18}$$

The worst-case likelihood ratio $\hat{m}_i$ operates to increase prices q_i in relatively low utility states i.

State prices agree under multiplier and constraint preferences when η and θ are adjusted according to (18.7) or (18.13) to make the indifference curves tangent at the endowment point.

The next figure can help us think about state-price deflators under our different preference orderings.

In this figure, budget line and indifference curves through point $(c_1, c_2) = (3, 1)$ for expected logarithmic utility, multiplier, constraint (kinked at 45 degree line), and *ex post* Bayesian (dotted lines) preferences.

Figure 2.7:

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
```

(continues on next page)

(continued from previous page)

```
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
```

(continues on next page)

(continued from previous page)

```
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
```

(continues on next page)

(continued from previous page)

```
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
```

(continues on next page)

(continued from previous page)

```
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
 ↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
 ↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
 ↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
 ↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
 ↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
 ↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
 ↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
 ↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
 ↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
 ↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
```

(continues on next page)

(continued from previous page)

```
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
```

(continues on next page)

(continued from previous page)

```
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
```

(continues on next page)

(continued from previous page)

```
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
```

(continues on next page)

(continued from previous page)

```
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
 ↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
 ↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
 ↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
 ↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
```

(continues on next page)

(continued from previous page)

```
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1000x800 with 1 Axes>
```

Because budget constraints are linear, asset prices are identical under multiplier and constraint preferences for which θ and η are adjusted to verify (18.7) or (18.13) at a given consumption endowment $\{c_i\}_{i=1}^I$.

However, as we note next, though they are tangent at the endowment point, the fact that indifference curves differ for multiplier and constraint preferences means that certainty equivalent consumption compensations of the kind that [Lucas, 1987], [Hansen *et al.*, 1999], [Tallarini, 2000], and [Barillas *et al.*, 2009] used to measure the costs of business cycles must differ.

18.12.1 Consumption-equivalent measures of uncertainty aversion

For each of our five types of preferences, the following figure allows us to construct a certainty equivalent point (c^*, c^*) on the 45 degree line that renders the consumer indifferent between it and the risky point $(c(1), c(2)) = (3, 1)$.

Figure 2.8:

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
```

(continues on next page)

(continued from previous page)

```
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
 ↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
 ↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
 ↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
 ↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
 ↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
 ↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
 ↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
 ↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
 ↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
 ↪composite)
```

(continues on next page)

(continued from previous page)

```
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
```

(continues on next page)

(continued from previous page)

```
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
 ↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
 ↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
 ↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
 ↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
 ↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
 ↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
 ↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
 ↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
```

(continues on next page)

(continued from previous page)

```
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
```

(continues on next page)

(continued from previous page)

```
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
 ↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
 ↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
 ↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
 ↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
```

(continues on next page)

(continued from previous page)

```
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
```

(continues on next page)

(continued from previous page)

```
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
```

(continues on next page)

(continued from previous page)

```
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 800x800 with 1 Axes>
```

The figure indicates that the certainty equivalent level c^* is higher for the consumer with expected utility preferences than for the consumer with multiplier preferences, and that it is higher for the consumer with multiplier preferences than for the consumer with constraint preferences.

The gap between these certainty equivalents measures the uncertainty aversion of the multiplier preferences or constraint preferences consumer.

The gap between the expected value $.5c(1) + .5c(2)$ at point A and the certainty equivalent for the expected utility decision maker at point B is a measure of his risk aversion.

The gap between points B and C measures the multiplier preference consumer's aversion to model uncertainty.

The gap between points B and D measures the constraint preference consumer's aversion to model uncertainty.

18.13 Iso-utility and iso-entropy curves and expansion paths

The following figures show iso-entropy and iso-utility lines for the special case in which $I = 3$, $\pi_1 = .3, \pi_2 = .4$, and the utility function is $u(c) = \frac{c^{1-\alpha}}{1-\alpha}$ with $\alpha = 0$ and $\alpha = 3$, respectively, for the fixed plan $c(1) = 1, c(2) = 2, c(3) = 3$.

The iso-utility lines are the level curves of

$$\hat{\pi}_1 c_1 + \hat{\pi}_2 c_2 + (1 - \hat{\pi}_1 - \hat{\pi}_2) c_3$$

and are linear in $(\hat{\pi}_1, \hat{\pi}_2)$.

This is what it means to say 'expected utility is linear in probabilities.'

Both figures plot the locus of points of tangency between the iso-entropy and the iso-utility curves that is traced out as one varies θ^{-1} in the interval $[0, 2]$.

While the iso-entropy lines are identical in the two figures, these 'expansion paths' differ because the utility functions differ, meaning that for a given θ and (c_1, c_2, c_3) triple, the worst-case probabilities $\hat{\pi}_i(\theta) = \pi_i \frac{\exp(-u(c_i)/\theta)}{E \exp(-u(c)/\theta)}$ differ as we vary θ, causing the associated entropies to differ.

Color bars:

- First color bar: variation in θ
- Second color bar: variation in utility levels
- Third color bar: variation in entropy levels

```
/tmp/ipykernel_2218/3904427642.py:36: RuntimeWarning: invalid value encountered in␣
↪divide
  m =  m_unnormalized / (π * m_unnormalized).sum()
```

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
```

(continues on next page)

(continued from previous page)

```
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
```

(continues on next page)

(continued from previous page)

```
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
```

(continues on next page)

(continued from previous page)

```
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
```

(continues on next page)

(continued from previous page)

```
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
```

(continues on next page)

(continued from previous page)

```
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
```

(continues on next page)

(continued from previous page)

```
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
```

(continues on next page)

(continued from previous page)

```
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1400x600 with 4 Axes>
```

```
/tmp/ipykernel_2218/3904427642.py:35: RuntimeWarning: overflow encountered in exp
  m_unnormalized = np.exp(-u(c) / θ)
/tmp/ipykernel_2218/3904427642.py:36: RuntimeWarning: invalid value encountered in␣
↪divide
  m =  m_unnormalized / (π * m_unnormalized).sum()
```

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

(continues on next page)

(continued from previous page)

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:
```

(continues on next page)

(continued from previous page)

```
RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
```

(continues on next page)

(continued from previous page)

```
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
```

(continues on next page)

(continued from previous page)

```
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath, dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop, ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop, ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
```

(continues on next page)

(continued from previous page)

```
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
 ↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
 ↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪formatters.py:340, in BaseFormatter.__call__(self, obj)
```

(continues on next page)

(continued from previous page)

```
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
```

(continues on next page)

(continued from previous page)

```
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
```

(continues on next page)

(continued from previous page)

```
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
 ↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
 ↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
 ↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
 ↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
 ↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
```

(continues on next page)

(continued from previous page)

```
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1400x600 with 4 Axes>
```

18.14 Bounds on expected utility

Suppose that a decision maker wants a lower bound on expected utility $\sum_{i=1}^I \hat{\pi}_i u(c_i)$ that is satisfied for **any** distribution $\hat{\pi}$ with relative entropy less than or equal to η.

An attractive feature of multiplier and constraint preferences is that they carry with them such a bound.

To show this, it is useful to collect some findings in the following string of inequalities associated with multiplier preferences:

$$\begin{aligned}
\mathsf{T}_\theta u(c) = & \quad -\theta \log \sum_{i=1}^I \exp\left(\frac{-u(c_i)}{\theta}\right) \pi_i \\
= & \quad \sum_{i=1}^I m_i^* \pi_i (u(c_i) + \theta \log m_i^*) \\
\leq & \quad \sum_{i=1}^I m_i \pi_i u(c_i) + \theta \sum_{i=1}^i m_i \log m_i \pi_i
\end{aligned}$$

where $m_i^* \propto \exp\left(\frac{-u(c_i)}{\theta}\right)$ are the worst-case distortions to probabilities.

The inequality in the last line just asserts that minimizers minimize.

Therefore, we have the following useful bound:

$$\sum_{i=1}^I m_i \pi_i u(c_i) \geq \mathsf{T}_\theta u(c) - \theta \sum_{i=1}^I \pi_i m_i \log m_i. \tag{18.19}$$

The left side is expected utility under the probability distribution $\{m_i \pi_i\}$.

The right side is a *lower bound* on expected utility under *all* distributions expressed as an affine function of relative entropy $\sum_{i=1}^I \pi_i m_i \log m_i$.

The bound is attained for $m_i = m_i^* \propto \exp\left(\frac{-u(c_i)}{\theta}\right)$.

The *intercept* in the bound is the risk-sensitive criterion $\mathsf{T}_\theta u(c)$, while the *slope* is the penalty parameter θ.

Lowering θ does two things:

- it lowers the intercept $\mathsf{T}_\theta u(c)$, which makes the bound less informative for small values of entropy; and

- it lowers the absolute value of the slope, which makes the bound more informative for larger values of relative entropy $\sum_{i=1}^{I} \pi_i m_i \log m_i$.

The following figure reports best-case and worst-case expected utilities.

We calculate the lines in this figure numerically by solving optimization problems with respect to the change of measure.

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
```

(continues on next page)

(continued from previous page)

```
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
```

(continues on next page)

(continued from previous page)

```
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
```

(continues on next page)

(continued from previous page)

```
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
```

(continues on next page)

(continued from previous page)

```
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
```

(continues on next page)

(continued from previous page)

```
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
```

(continues on next page)

(continued from previous page)

```
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
```

(continues on next page)

(continued from previous page)

```
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
```

(continues on next page)

(continued from previous page)

```
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 800x600 with 1 Axes>
```

In this figure, expected utility is on the co-ordinate axis while entropy is on the ordinate axis.

The *lower* curved line depicts expected utility under the worst-case model associated with each value of entropy η recorded on the ordinate axis, i.e., it is $\sum_{i=1}^{I} \pi_i \tilde{m}_i(\tilde{\theta}(c,\eta))u(c_i)$, where $\tilde{m}_i(\tilde{\theta}(\eta)) \propto \exp\left(\frac{-u(c_i)}{\tilde{\theta}}\right)$ and $\tilde{\theta}$ is the Lagrange multiplier associated with the constraint that entropy cannot exceed the value on the ordinate axis.

The *higher* curved line depicts expected utility under the *best*-case model indexed by the value of the Lagrange multiplier $\check{\theta} > 0$ associated with each value of entropy less than or equal to η recorded on the ordinate axis, i.e., it is $\sum_{i=1}^{I} \pi_i \check{m}_i(\check{\theta}(\eta))u(c_i)$ where $\check{m}_i(\check{\theta}(c,\eta)) \propto \exp\left(\frac{u(c_i)}{\check{\theta}}\right)$.

(Here $\check{\theta}$ is the Lagrange multiplier associated with max-max expected utility.)

Points between these two curves are possible values of expected utility for some distribution with entropy less than or equal to the value η on the ordinate axis.

The straight line depicts the right side of inequality (18.19) for a particular value of the penalty parameter θ.

As noted, when one lowers θ, the intercept $\mathsf{T}_\theta u(c)$ and the absolute value of the slope both decrease.

Thus, as θ is lowered, $\mathsf{T}_\theta u(c)$ becomes a more conservative estimate of expected utility under the approximating model π.

However, as θ is lowered, the robustness bound (18.19) becomes more informative for sufficiently large values of entropy.

The slope of straight line depicting a bound is $-\theta$ and the projection of the point of tangency with the curved depicting the lower bound of expected utility is the entropy associated with that θ when it is interpreted as a Lagrange multiplier on the entropy constraint in the constraint problem .

This is an application of the envelope theorem.

18.15 Why entropy?

Beyond the helpful mathematical fact that it leads directly to convenient exponential twisting formulas (18.6) and (18.12) for worst-case probability distortions, there are two related justifications for using entropy to measure discrepancies between probability distribution.

One arises from the role of entropy in statistical tests for discriminating between models.

The other comes from axioms.

18.15.1 Entropy and statistical detection

Robust control theory starts with a decision maker who has constructed a good baseline approximating model whose free parameters he has estimated to fit historical data well.

The decision maker recognizes that actual outcomes might be generated by one of a vast number of other models that fit the historical data nearly as well as his.

Therefore, he wants to evaluate outcomes under a set of alternative models that are plausible in the sense of being statistically close to his model.

He uses relative entropy to quantify what close means.

[Anderson *et al.*, 2003] and [Barillas *et al.*, 2009]describe links between entropy and large deviations bounds on test statistics for discriminating between models, in particular, statistics that describe the probability of making an error in applying a likelihood ratio test to decide whether model A or model B generated a data record of length T.

For a given sample size, an informative bound on the detection error probability is a function of the entropy parameter η in constraint preferences. [Anderson *et al.*, 2003] and [Barillas *et al.*, 2009] use detection error probabilities to calibrate reasonable values of η.

[Anderson *et al.*, 2003] and [Hansen and Sargent, 2008] also use detection error probabilities to calibrate reasonable values of the penalty parameter θ in multiplier preferences.

For a fixed sample size and a fixed θ, they would calculate the worst-case $\hat{m}_i(\theta)$, an associated entropy $\eta(\theta)$, and an associated detection error probability. In this way they build up a detection error probability as a function of θ.

They then invert this function to calibrate θ to deliver a reasonable detection error probability.

To indicate outcomes from this approach, the following figure plots the histogram for U.S. quarterly consumption growth along with a representative agent's approximating density and a worst-case density that [Barillas *et al.*, 2009] show imply high measured market prices of risk even when a representative consumer has the unit coefficient of relative risk aversion associated with a logarithmic one-period utility function.

```
Error in callback <function _draw_all_if_interactive at 0x7fb2b3488b80> (for post_
↪execute), with arguments args (),kwargs {}:
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/pyplot.
↪py:197, in _draw_all_if_interactive()
    195 def _draw_all_if_interactive() -> None:
    196     if matplotlib.is_interactive():
--> 197         draw_all()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/_pylab_
↪helpers.py:132, in Gcf.draw_all(cls, force)
    130 for manager in cls.get_all_fig_managers():
    131     if force or manager.canvas.figure.stale:
--> 132         manager.canvas.draw_idle()
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:1893, in FigureCanvasBase.draw_idle(self, *args, **kwargs)
   1891 if not self._is_idle_drawing:
   1892     with self._idle_draw_cntx():
-> 1893         self.draw(*args, **kwargs)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:388, in FigureCanvasAgg.draw(self)
    385 # Acquire a lock on the shared font cache.
    386 with (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    387       else nullcontext()):
--> 388     self.figure.draw(self.renderer)
    389     # A GUI class may be need to update a window using this draw, so
    390     # don't forget to call the superclass.
    391     super().draw()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
```

(continues on next page)

(continued from previous page)

```
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
```

(continues on next page)

(continued from previous page)

```
   1318         for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
 ↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
 ↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
 ↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
 ↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
 ↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
 ↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
 ↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
 ↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
 ↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
 ↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
```

(continues on next page)

(continued from previous page)

```
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
 ↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
 ↪texmanager.py:250, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    249 try:
--> 250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:466, in check_
 ↪output(timeout, *popenargs, **kwargs)
    464     kwargs['input'] = empty
--> 466 return run(*popenargs, stdout=PIPE, timeout=timeout, check=True,
    467            **kwargs).stdout

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:548, in run(input,␣
 ↪capture_output, timeout, check, *popenargs, **kwargs)
    546     kwargs['stderr'] = PIPE
--> 548 with Popen(*popenargs, **kwargs) as process:
    549     try:

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1026, in Popen.__
 ↪init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_
 ↪fds, shell, cwd, env, universal_newlines, startupinfo, creationflags, restore_
 ↪signals, start_new_session, pass_fds, user, group, extra_groups, encoding,␣
 ↪errors, text, umask, pipesize, process_group)
   1023             self.stderr = io.TextIOWrapper(self.stderr,
   1024                     encoding=encoding, errors=errors)
-> 1026     self._execute_child(args, executable, preexec_fn, close_fds,
   1027                         pass_fds, cwd, env,
   1028                         startupinfo, creationflags, shell,
   1029                         p2cread, p2cwrite,
   1030                         c2pread, c2pwrite,
   1031                         errread, errwrite,
   1032                         restore_signals,
   1033                         gid, gids, uid, umask,
   1034                         start_new_session, process_group)
   1035 except:
   1036     # Cleanup if the child failed starting.

File ~/miniconda3/envs/quantecon/lib/python3.11/subprocess.py:1950, in Popen._
 ↪execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd, env,␣
 ↪startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite, errread,
 ↪ errwrite, restore_signals, gid, gids, uid, umask, start_new_session, process_
 ↪group)
   1949         err_msg = os.strerror(errno_num)
-> 1950     raise child_exception_type(errno_num, err_msg, err_filename)
   1951 raise child_exception_type(err_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'latex'

The above exception was the direct cause of the following exception:

RuntimeError                              Traceback (most recent call last)
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪formatters.py:340, in BaseFormatter.__call__(self, obj)
    338     pass
    339 else:
--> 340     return printer(obj)
    341 # Finally look for special method names
    342 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:169, in retina_figure(fig, base64, **kwargs)
    160 def retina_figure(fig, base64=False, **kwargs):
```

(continues on next page)

(continued from previous page)

```
    161     """format a figure as a pixel-doubled (retina) PNG
    162 
    163     If `base64` is True, return base64-encoded str instead of raw bytes
   (...)
    167         base64 argument
    168     """
--> 169     pngdata = print_figure(fig, fmt="retina", base64=False, **kwargs)
    170     # Make sure that retina_figure acts just like print_figure and returns
    171     # None when the figure is empty.
    172     if pngdata is None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/IPython/core/
 ↪pylabtools.py:152, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    149     from matplotlib.backend_bases import FigureCanvasBase
    150     FigureCanvasBase(fig)
--> 152 fig.canvas.print_figure(bytes_io, **kw)
    153 data = bytes_io.getvalue()
    154 if fmt == 'svg':

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
 ↪bases.py:2158, in FigureCanvasBase.print_figure(self, filename, dpi, facecolor,␣
 ↪edgecolor, orientation, format, bbox_inches, pad_inches, bbox_extra_artists,␣
 ↪backend, **kwargs)
   2155     # we do this instead of `self.figure.draw_without_rendering`
   2156     # so that we can inject the orientation
   2157     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2158         self.figure.draw(renderer)
   2159 if bbox_inches:
   2160     if bbox_inches == "tight":

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
 ↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer, *args,␣
 ↪**kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
 ↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/figure.
 ↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axes/_
↪base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/image.
↪py:132, in _draw_list_compositing_images(renderer, parent, artists, suppress_
↪composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/artist.
↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1388, in Axis.draw(self, renderer, *args, **kwargs)
   1385 renderer.open_group(__name__, gid=self.get_gid())
   1387 ticks_to_draw = self._update_ticks()
-> 1388 tlb1, tlb2 = self._get_ticklabel_bboxes(ticks_to_draw, renderer)
   1390 for tick in ticks_to_draw:
   1391     tick.draw(renderer)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in Axis._get_ticklabel_bboxes(self, ticks, renderer)
   1313 if renderer is None:
```

(continues on next page)

(continued from previous page)

```
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/axis.
↪py:1315, in <listcomp>(.0)
   1313 if renderer is None:
   1314     renderer = self.figure._get_renderer()
-> 1315 return ([tick.label1.get_window_extent(renderer)
   1316          for tick in ticks if tick.label1.get_visible()],
   1317         [tick.label2.get_window_extent(renderer)
   1318          for tick in ticks if tick.label2.get_visible()])

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:956, in Text.get_window_extent(self, renderer, dpi)
    951     raise RuntimeError(
    952         "Cannot get window extent of text w/o renderer. You likely "
    953         "want to call 'figure.draw_without_rendering()' first.")
    955 with cbook._setattr_cm(self.figure, dpi=dpi):
--> 956     bbox, info, descent = self._get_layout(self._renderer)
    957     x, y = self.get_unitless_position()
    958     x, y = self.get_transform().transform((x, y))

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:373, in Text._get_layout(self, renderer)
    370 ys = []
    372 # Full vertical extent of font, including ascenders and descenders:
--> 373 _, lp_h, lp_d = _get_text_metrics_with_cache(
    374     renderer, "lp", self._fontproperties,
    375     ismath="TeX" if self.get_usetex() else False, dpi=self.figure.dpi)
    376 min_dy = (lp_h - lp_d) * self._linespacing
    378 for i, line in enumerate(lines):

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:69, in _get_text_metrics_with_cache(renderer, text, fontprop, ismath, dpi)
     66 """Call ``renderer.get_text_width_height_descent``, caching the results."""
     67 # Cached based on a copy of fontprop so that later in-place mutations of
     68 # the passed-in argument do not mess up the cache.
---> 69 return _get_text_metrics_with_cache_impl(
     70     weakref.ref(renderer), text, fontprop.copy(), ismath, dpi)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/text.
↪py:77, in _get_text_metrics_with_cache_impl(renderer_ref, text, fontprop, ismath,
↪ dpi)
     73 @functools.lru_cache(4096)
     74 def _get_text_metrics_with_cache_impl(
     75         renderer_ref, text, fontprop, ismath, dpi):
     76     # dpi is unused, but participates in cache invalidation (via the␣
↪renderer).
---> 77     return renderer_ref().get_text_width_height_descent(text, fontprop,␣
↪ismath)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backends/
↪backend_agg.py:213, in RendererAgg.get_text_width_height_descent(self, s, prop,␣
↪ismath)
```

(continues on next page)

(continued from previous page)

```
    211 _api.check_in_list(["TeX", True, False], ismath=ismath)
    212 if ismath == "TeX":
--> 213     return super().get_text_width_height_descent(s, prop, ismath)
    215 if ismath:
    216     ox, oy, width, height, descent, font_image = \
    217         self.mathtext_parser.parse(s, self.dpi, prop)

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/backend_
↪bases.py:652, in RendererBase.get_text_width_height_descent(self, s, prop,␣
↪ismath)
    648 fontsize = prop.get_size_in_points()
    650 if ismath == 'TeX':
    651     # todo: handle properties
--> 652     return self.get_texmanager().get_text_width_height_descent(
    653         s, fontsize, renderer=self)
    655 dpi = self.points_to_pixels(72)
    656 if ismath:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:363, in TexManager.get_text_width_height_descent(cls, tex,␣
↪fontsize, renderer)
    361 if tex.strip() == '':
    362     return 0, 0, 0
--> 363 dvifile = cls.make_dvi(tex, fontsize)
    364 dpi_fraction = renderer.points_to_pixels(1.) if renderer else 1
    365 with dviread.Dvi(dvifile, 72 * dpi_fraction) as dvi:

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:295, in TexManager.make_dvi(cls, tex, fontsize)
    293     with TemporaryDirectory(dir=cwd) as tmpdir:
    294         tmppath = Path(tmpdir)
--> 295         cls._run_checked_subprocess(
    296             ["latex", "-interaction=nonstopmode", "--halt-on-error",
    297              f"--output-directory={tmppath.name}",
    298              f"{texfile.name}"], tex, cwd=cwd)
    299         (tmppath / Path(dvifile).name).replace(dvifile)
    300 return dvifile

File ~/miniconda3/envs/quantecon/lib/python3.11/site-packages/matplotlib/
↪texmanager.py:254, in TexManager._run_checked_subprocess(cls, command, tex, cwd)
    250     report = subprocess.check_output(
    251         command, cwd=cwd if cwd is not None else cls._texcache,
    252         stderr=subprocess.STDOUT)
    253 except FileNotFoundError as exc:
--> 254     raise RuntimeError(
    255         f'Failed to process string with tex because {command[0]} '
    256         'could not be found') from exc
    257 except subprocess.CalledProcessError as exc:
    258     raise RuntimeError(
    259         '{prog} was not able to process the following string:\n'
    260         '{tex!r}\n\n'
   (...)
    267             exc=exc.output.decode('utf-8', 'backslashreplace'))
    268         ) from None

RuntimeError: Failed to process string with tex because latex could not be found
```

```
<Figure size 1000x800 with 2 Axes>
```

The density for the approximating model is $\log c_{t+1} - \log c_t = \mu + \sigma_c \epsilon_{t+1}$ where $\epsilon_{t+1} \sim N(0,1)$ and μ and σ_c are estimated by maximum likelihood from the U.S. quarterly data in the histogram over the period 1948.I-2006.IV.

The consumer's value function under logarithmic utility implies that the worst-case model is $\log c_{t+1} - \log c_t = (\mu + \sigma_c w) + \sigma_c \tilde{\epsilon}_{t+1}$ where $\{\tilde{\epsilon}_{t+1}\}$ is also a normalized Gaussian random sequence and where w is calculated by setting a detection error probability to .05.

The worst-case model appears to fit the histogram nearly as well as the approximating model.

18.15.2 Axiomatic justifications

Multiplier and constraint preferences are both special cases of what [Maccheroni *et al.*, 2006] call variational preferences.

They provide an axiomatic foundation for variational preferences and describe how they express ambiguity aversion.

Constraint preferences are particular instances of the multiple priors model of [Gilboa and Schmeidler, 1989].

CHAPTER

NINETEEN

ETYMOLOGY OF ENTROPY

This lecture describes and compares several notions of entropy.

Among the senses of entropy, we'll encounter these

- A measure of **uncertainty** of a random variable advanced by Claude Shannon [Shannon and Weaver, 1949]
- A key object governing thermodynamics
- Kullback and Leibler's measure of the statistical divergence between two probability distributions
- A measure of the volatility of stochastic discount factors that appear in asset pricing theory
- Measures of unpredictability that occur in classical Wiener-Kolmogorov linear prediction theory
- A frequency domain criterion for constructing robust decision rules

The concept of entropy plays an important role in robust control formulations described in this lecture Risk and Model Uncertainty and in this lecture Robustness.

19.1 Information Theory

In information theory [Shannon and Weaver, 1949], entropy is a measure of the unpredictability of a random variable.

To illustrate things, let X be a discrete random variable taking values $x_1, \dots, x_n$ with probabilities $p_i = \text{Prob}(X = x_i) \geq 0, \sum_i p_i = 1$.

Claude Shannon's [Shannon and Weaver, 1949] definition of entropy is

$$H(p) = \sum_i p_i \log_b(p_i^{-1}) = -\sum_i p_i \log_b(p_i). \tag{19.1}$$

where $\log_b$ denotes the log function with base b.

Inspired by the limit

$$\lim_{p \downarrow 0} p \log p = \lim_{p \downarrow 0} \frac{\log p}{p^{-1}} = \lim_{p \downarrow 0} p = 0,$$

we set $p \log p = 0$ in equation (19.1).

Typical bases for the logarithm are 2, e, and 10.

In the information theory literature, logarithms of base 2, e, and 10 are associated with units of information called bits, nats, and dits, respectively.

Shannon typically used base 2.

19.2 A Measure of Unpredictability

For a discrete random variable X with probability density $p = \{p_i\}_{i=1}^n$, the **surprisal** for state i is $s_i = \log\left(\frac{1}{p_i}\right)$.

The quantity $\log\left(\frac{1}{p_i}\right)$ is called the **surprisal** because it is inversely related to the likelihood that state i will occur.

Note that entropy $H(p)$ equals the **expected surprisal**

$$H(p) = \sum_i p_i s_i = \sum_i p_i \log\left(\frac{1}{p_i}\right) = -\sum_i p_i \log\left(p_i\right).$$

19.2.1 Example

Take a possibly unfair coin, so $X = \{0, 1\}$ with $p = \text{Prob}(X = 1) = p \in [0, 1]$.

Then

$$H(p) = -(1-p)\log(1-p) - p\log p.$$

Evidently,

$$H'(p) = \log(1-p) - \log p = 0$$

at $p = .5$ and $H''(p) = -\frac{1}{1-p} - \frac{1}{p} < 0$ for $p \in (0, 1)$.

So $p = .5$ maximizes entropy, while entropy is minimized at $p = 0$ and $p = 1$.

Thus, among all coins, a fair coin is the most unpredictable.

See Fig. 19.1

19.2.2 Example

Take an n-sided possibly unfair die with a probability distribution $\{p_i\}_{i=1}^n$. The die is fair if $p_i = \frac{1}{n} \forall i$.

Among all dies, a fair die maximizes entropy.

For a fair die, entropy equals $H(p) = -n^{-1}\sum_i \log\left(\frac{1}{n}\right) = \log(n)$.

To specify the expected number of bits needed to isolate the outcome of one roll of a fair n-sided die requires $\log_2(n)$ bits of information.

For example, if $n = 2$, $\log_2(2) = 1$.

For $n = 3$, $\log_2(3) = 1.585$.

19.3 Mathematical Properties of Entropy

For a discrete random variable with probability vector p, entropy $H(p)$ is a function that satisfies

- H is *continuous*.
- H is *symmetric*: $H(p_1, p_2, \ldots, p_n) = H(p_{r_1}, \ldots, p_{r_n})$ for any permutation $r_1, \ldots, r_n$ of $1, \ldots, n$.
- A uniform distribution maximizes $H(p)$: $H(p_1, \ldots, p_n) \leq H(\frac{1}{n}, \ldots, \frac{1}{n})$.
- Maximum entropy increases with the number of states: $H(\frac{1}{n}, \ldots, \frac{1}{n}) \leq H(\frac{1}{n+1}, \ldots, \frac{1}{n+1})$.
- Entropy is not affected by events zero probability.

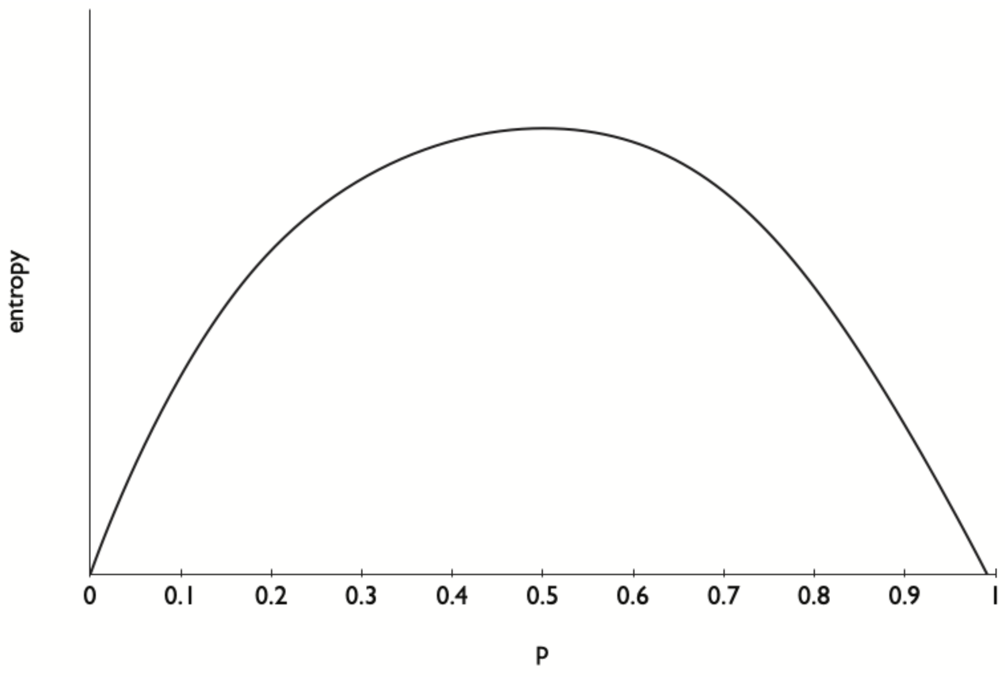


Fig. 19.1: Entropy as a function of $\hat{\pi}_1$ when $\pi_1 = .5$.

19.4 Conditional Entropy

Let (X, Y) be a bivariate discrete random vector with outcomes $x_1, \dots, x_n$ and $y_1, \dots, y_m$, respectively, occurring with probability density $p(x_i, y_i)$.

Conditional entropy $H(X|Y)$ is defined as

$$H(X|Y) = \sum_{i,j} p(x_i, y_j) \log \frac{p(y_j)}{p(x_i, y_j)}. \tag{19.2}$$

Here $\frac{p(y_j)}{p(x_i,y_j)}$, the reciprocal of the conditional probability of x_i given y_j, can be defined as the **conditional surprisal**.

19.5 Independence as Maximum Conditional Entropy

Let $m = n$ and $[x_1, \dots, x_n] = [y_1, \dots, y_n]$.

Let $\sum_j p(x_i, y_j) = \sum_j p(x_j, y_i)$ for all i, so that the marginal distributions of x and y are identical.

Thus, x and y are identically distributed, but they are not necessarily independent.

Consider the following problem: choose a joint distribution $p(x_i, y_j)$ to maximize conditional entropy (19.2) subject to the restriction that x and y are identically distributed.

The conditional-entropy-maximizing $p(x_i, y_j)$ sets

$$\frac{p(x_i, y_j)}{p(y_j)} = \sum_j p(x_i, y_j) = p(x_i) \forall i.$$

Thus, among all joint distributions with identical marginal distributions, the conditional entropy maximizing joint distribution makes x and y be independent.

19.6 Thermodynamics

Josiah Willard Gibbs (see https://en.wikipedia.org/wiki/Josiah_Willard_Gibbs) defined entropy as

$$S = -k_B \sum_i p_i \log p_i \tag{19.3}$$

where p_i is the probability of a micro state and k_B is Boltzmann's constant.

- The Boltzmann constant k_b relates energy at the micro particle level with the temperature observed at the macro level. It equals what is called a gas constant divided by an Avogadro constant.

The second law of thermodynamics states that the entropy of a closed physical system increases until S defined in (19.3) attains a maximum.

19.7 Statistical Divergence

Let X be a discrete state space $x_1, \ldots, x_n$ and let p and q be two discrete probability distributions on X.

Assume that $\frac{p_i}{q_t} \in (0, \infty)$ for all i for which $p_i > 0$.

Then the Kullback-Leibler statistical divergence, also called **relative entropy**, is defined as

$$D(p|q) = \sum_i p_i \log \left(\frac{p_i}{q_i}\right) = \sum_i q_i \left(\frac{p_i}{q_i}\right) \log \left(\frac{p_i}{q_i}\right). \tag{19.4}$$

Evidently,

$$\begin{aligned} D(p|q) &= -\sum_i p_i \log q_i + \sum_i p_i \log p_i \\ &= H(p,q) - H(p), \end{aligned}$$

where $H(p,q) = \sum_i p_i \log q_i$ is the cross-entropy.

It is easy to verify, as we have done above, that $D(p|q) \geq 0$ and that $D(p|q) = 0$ implies that $p_i = q_i$ when $q_i > 0$.

19.8 Continuous distributions

For a continuous random variable, Kullback-Leibler divergence between two densities p and q is defined as

$$D(p|q) = \int p(x) \log \left(\frac{p(x)}{q(x)}\right) d\,x.$$

19.9 Relative entropy and Gaussian distributions

We want to compute relative entropy for two continuous densities ϕ and $\hat{\phi}$ when ϕ is $N(0, I)$ and $\hat{\phi}$ is $N(w, \Sigma)$, where the covariance matrix Σ is nonsingular.

We seek a formula for

$$\text{ent} = \int (\log \hat{\phi}(\varepsilon) - \log \phi(\varepsilon))\hat{\phi}(\varepsilon) d\varepsilon.$$

Claim

$$\text{ent} = -\frac{1}{2} \log \det \Sigma + \frac{1}{2} w' w + \frac{1}{2} \text{trace}(\Sigma - I). \tag{19.5}$$

Proof

The log likelihood ratio is

$$\log \hat{\phi}(\varepsilon) - \log \phi(\varepsilon) = \frac{1}{2} \left[-(\varepsilon - w)' \Sigma^{-1} (\varepsilon - w) + \varepsilon' \varepsilon - \log \det \Sigma \right]. \tag{19.6}$$

Observe that

$$-\int \frac{1}{2} (\varepsilon - w)' \Sigma^{-1} (\varepsilon - w) \hat{\phi}(\varepsilon) d\varepsilon = -\frac{1}{2} \text{trace}(I).$$

Applying the identity $\varepsilon = w + (\varepsilon - w)$ gives

$$\frac{1}{2} \varepsilon' \varepsilon = \frac{1}{2} w' w + \frac{1}{2} (\varepsilon - w)' (\varepsilon - w) + w' (\varepsilon - w).$$

Taking mathematical expectations

$$\frac{1}{2} \int \varepsilon' \varepsilon \hat{\phi}(\varepsilon) d\varepsilon = \frac{1}{2} w' w + \frac{1}{2} \text{trace}(\Sigma).$$

Combining terms gives

$$\text{ent} = \int (\log \hat{\phi} - \log \phi) \hat{\phi} d\varepsilon = -\frac{1}{2} \log \det \Sigma + \frac{1}{2} w' w + \frac{1}{2} \text{trace}(\Sigma - I). \tag{19.7}$$

which agrees with equation (19.5). Notice the separate appearances of the mean distortion w and the covariance distortion $\Sigma - I$ in equation (19.7).

Extension

Let $N_0 = \mathcal{N}(\mu_0, \Sigma_0)$ and $N_1 = \mathcal{N}(\mu_1, \Sigma_1)$ be two multivariate Gaussian distributions.

Then

$$D(N_0 | N_1) = \frac{1}{2} \left(\text{trace}(\Sigma_1^{-1} \Sigma_0) + (\mu_1 - \mu_0)' \Sigma_1^{-1} (\mu_1 - \mu_0) - \log \left(\frac{\det \Sigma_0}{\det \Sigma_1} \right) - k \right). \tag{19.8}$$

19.10 Von Neumann Entropy

Let P and Q be two positive-definite symmetric matrices.

A measure of the divergence between two P and Q is

$$D(P|Q) = \text{trace}(P \ln P - P \ln Q - P + Q)$$

where the log of a matrix is defined here (https://en.wikipedia.org/wiki/Logarithm_of_a_matrix).

A density matrix P from quantum mechanics is a positive definite matrix with trace 1.

The von Neumann entropy of a density matrix P is

$$S = -\text{trace}(P \ln P)$$

19.11 Backus-Chernov-Zin Entropy

After flipping signs, [Backus *et al.*, 2014] use Kullback-Leibler relative entropy as a measure of volatility of stochastic discount factors that they assert is useful for characterizing features of both the data and various theoretical models of stochastic discount factors.

Where p_{t+1} is the physical or true measure, p^*_{t+1} is the risk-neutral measure, and E_t denotes conditional expectation under the p_{t+1} measure, [Backus *et al.*, 2014] define entropy as

$$L_t(p^*_{t+1}/p_{t+1}) = -E_t \log(p^*_{t+1}/p_{t+1}). \tag{19.9}$$

Evidently, by virtue of the minus sign in equation (19.9),

$$L_t(p^*_{t+1}/p_{t+1}) = D_{KL,t}(p^*_{t+1}|p_{t+1}), \tag{19.10}$$

where $D_{KL,t}$ denotes conditional relative entropy.

Let m_{t+1} be a stochastic discount factor, r_{t+1} a gross one-period return on a risky security, and $(r^1_{t+1})^{-1} \equiv q^1_t = E_t m_{t+1}$ be the reciprocal of a risk-free one-period gross rate of return. Then

$$E_t(m_{t+1} r_{t+1}) = 1$$

[Backus *et al.*, 2014] note that a stochastic discount factor satisfies

$$m_{t+1} = q^1_t p^*_{t+1}/p_{t+1}.$$

They derive the following **entropy bound**

$$EL_t(m_{t+1}) \geq E(\log r_{t+1} - \log r^1_{t+1})$$

which they propose as a complement to a Hansen-Jagannathan [Hansen and Jagannathan, 1991] bound.

19.12 Wiener-Kolmogorov Prediction Error Formula as Entropy

Let $\{x_t\}_{t=-\infty}^{\infty}$ be a covariance stationary stochastic process with mean zero and spectral density $S_x(\omega)$.

The variance of x is

$$\sigma^2_x = \left(\frac{1}{2\pi}\right) \int_{-\pi}^{\pi} S_x(\omega) d\omega.$$

As described in chapter XIV of [Sargent, 1987], the Wiener-Kolmogorov formula for the one-period ahead prediction error is

$$\sigma^2_\epsilon = \exp\left[\left(\frac{1}{2\pi}\right) \int_{-\pi}^{\pi} \log S_x(\omega) d\omega\right]. \tag{19.11}$$

Occasionally the logarithm of the one-step-ahead prediction error σ_ϵ^2 is called entropy because it measures unpredictability.

Consider the following problem reminiscent of one described earlier.

Problem:

Among all covariance stationary univariate processes with unconditional variance σ_x^2, find a process with maximal one-step-ahead prediction error.

The maximizer is a process with spectral density

$$S_x(\omega) = 2\pi\sigma_x^2.$$

Thus, among all univariate covariance stationary processes with variance σ_x^2, a process with a flat spectral density is the most uncertain, in the sense of one-step-ahead prediction error variance.

This no-patterns-across-time outcome for a temporally dependent process resembles the no-pattern-across-states outcome for the static entropy maximizing coin or die in the classic information theoretic analysis described above.

19.13 Multivariate Processes

Let y_t be an $n \times 1$ covariance stationary stochastic process with mean 0 with matrix covariogram $C_y(j) = Ey_t y_{t-j}'$ and spectral density matrix

$$S_y(\omega) = \sum_{j=-\infty}^{\infty} e^{-i\omega j} C_y(j), \quad \omega \in [-\pi, \pi].$$

Let

$$y_t = D(L)\epsilon_t \equiv \sum_{j=0}^{\infty} D_j \epsilon_t$$

be a Wold representation for y, where $D(0)\epsilon_t$ is a vector of one-step-ahead errors in predicting y_t conditional on the infinite history $y^{t-1} = [y_{t-1}, y_{t-2}, ...]$ and ϵ_t is an $n \times 1$ vector of serially uncorrelated random disturbances with mean zero and identity contemporaneous covariance matrix $E\epsilon_t\epsilon_t' = I$.

Linear-least-squares predictors have one-step-ahead prediction error $D(0)D(0)'$ that satisfies

$$\log\det[D(0)D(0)'] = \left(\frac{1}{2\pi}\right)\int_{-\pi}^{\pi} \log\det[S_y(\omega)]d\omega. \tag{19.12}$$

Being a measure of the unpredictability of an $n \times 1$ vector covariance stationary stochastic process, the left side of (19.12) is sometimes called entropy.

19.14 Frequency Domain Robust Control

Chapter 8 of [Hansen and Sargent, 2008] adapts work in the control theory literature to define a **frequency domain entropy** criterion for robust control as

$$\int_\Gamma \log\det[\theta I - G_F(\zeta)' G_F(\zeta)]d\lambda(\zeta), \tag{19.13}$$

where $\theta \in (\underline{\theta}, +\infty)$ is a positive robustness parameter and $G_F(\zeta)$ is a ζ-transform of the objective function.

Hansen and Sargent [Hansen and Sargent, 2008] show that criterion (19.13) can be represented as

$$\log \det[D(0)'D(0)] = \int_\Gamma \log \det[\theta I - G_F(\zeta)' G_F(\zeta)] d\lambda(\zeta), \tag{19.14}$$

for an appropriate covariance stationary stochastic process derived from $\theta, G_F(\zeta)$.

This explains the moniker **maximum entropy** robust control for decision rules F designed to maximize criterion (19.13).

19.15 Relative Entropy for a Continuous Random Variable

Let x be a continuous random variable with density $\phi(x)$, and let $g(x)$ be a nonnegative random variable satisfying $\int g(x)\phi(x)dx = 1$.

The relative entropy of the distorted density $\hat{\phi}(x) = g(x)\phi(x)$ is defined as

$$\text{ent}(g) = \int g(x) \log g(x) \phi(x) dx.$$

Fig. 19.2 plots the functions $g \log g$ and $g - 1$ over the interval $g \geq 0$.

That relative entropy $\text{ent}(g) \geq 0$ can be established by noting (a) that $g \log g \geq g - 1$ (see Fig. 19.2) and (b) that under ϕ, $Eg = 1$.

Fig. 19.3 and Fig. 19.4 display aspects of relative entropy visually for a continuous random variable x for two densities with likelihood ratio $g \geq 0$.

Where the numerator density is $\mathcal{N}(0, 1)$, for two denominator Gaussian densities $\mathcal{N}(0, 1.5)$ and $\mathcal{N}(0, .95)$, respectively, Fig. 19.3 and Fig. 19.4 display the functions $g \log g$ and $g - 1$ as functions of x.

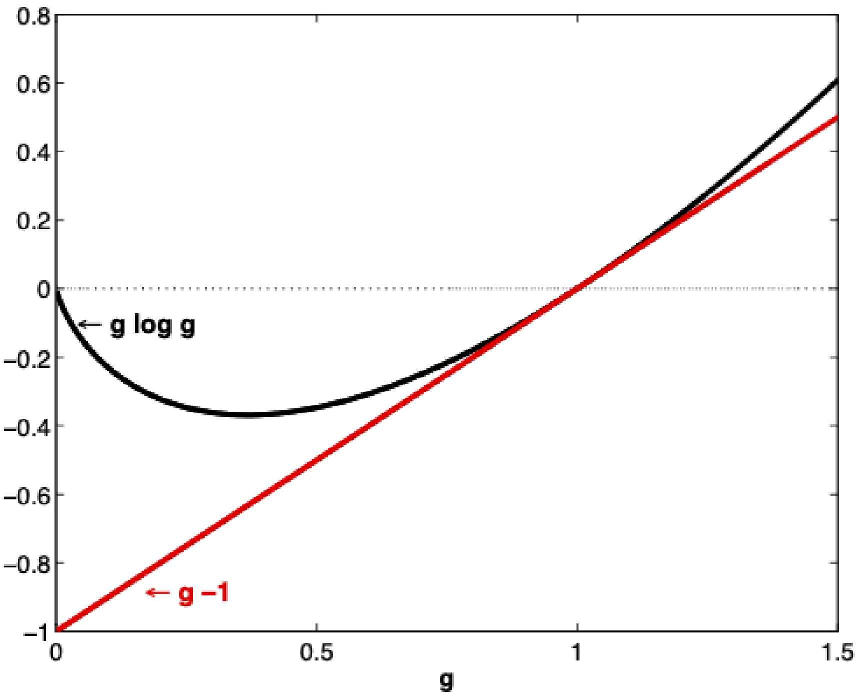


Fig. 19.2: The function $g \log g$ for $g \geq 0$. For a random variable g with $Eg = 1$, $Eg \log g \geq 0$.

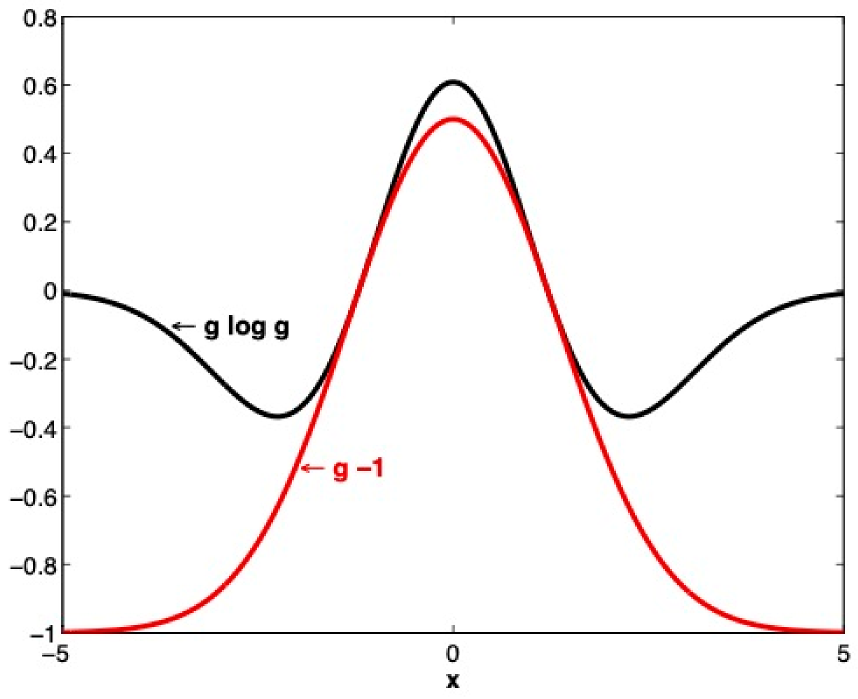


Fig. 19.3: Graphs of $g \log g$ and $g - 1$ where g is the ratio of the density of a $\mathcal{N}(0, 1)$ random variable to the density of a $\mathcal{N}(0, 1.5)$ random variable. Under the $\mathcal{N}(0, 1.5)$ density, $Eg = 1$.

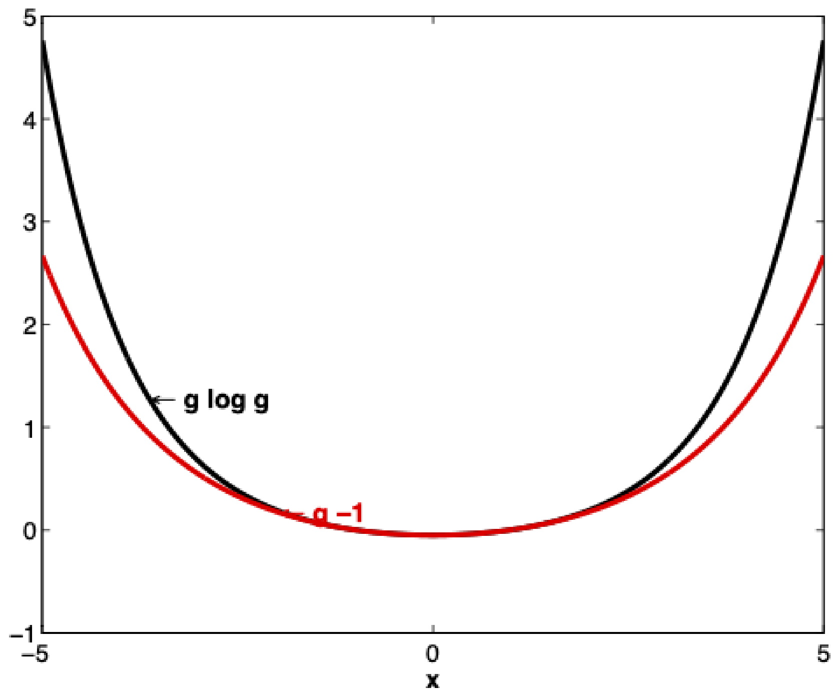


Fig. 19.4: $g \log g$ and $g - 1$ where g is the ratio of the density of a $\mathcal{N}(0, 1)$ random variable to the density of a $\mathcal{N}(0, 1.5)$ random variable. Under the $\mathcal{N}(0, 1.5)$ density, $Eg = 1$.

CHAPTER

TWENTY

ROBUSTNESS

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
```

20.1 Overview

This lecture modifies a Bellman equation to express a decision-maker's doubts about transition dynamics.

His specification doubts make the decision-maker want a *robust* decision rule.

Robust means insensitive to misspecification of transition dynamics.

The decision-maker has a single *approximating model*.

He calls it *approximating* to acknowledge that he doesn't completely trust it.

He fears that outcomes will actually be determined by another model that he cannot describe explicitly.

All that he knows is that the actual data-generating model is in some (uncountable) set of models that surrounds his approximating model.

He quantifies the discrepancy between his approximating model and the genuine data-generating model by using a quantity called *entropy*.

(We'll explain what entropy means below)

He wants a decision rule that will work well enough no matter which of those other models actually governs outcomes.

This is what it means for his decision rule to be "robust to misspecification of an approximating model".

This may sound like too much to ask for, but ….

… a *secret weapon* is available to design robust decision rules.

The secret weapon is max-min control theory.

A value-maximizing decision-maker enlists the aid of an (imaginary) value-minimizing model chooser to construct *bounds* on the value attained by a given decision rule under different models of the transition dynamics.

The original decision-maker uses those bounds to construct a decision rule with an assured performance level, no matter which model actually governs outcomes.

Note: In reading this lecture, please don't think that our decision-maker is paranoid when he conducts a worst-case analysis. By designing a rule that works well against a worst-case, his intention is to construct a rule that will work well across a *set* of models.

Let's start with some imports:

```
import pandas as pd
import numpy as np
from scipy.linalg import eig
import matplotlib.pyplot as plt
import quantecon as qe
```

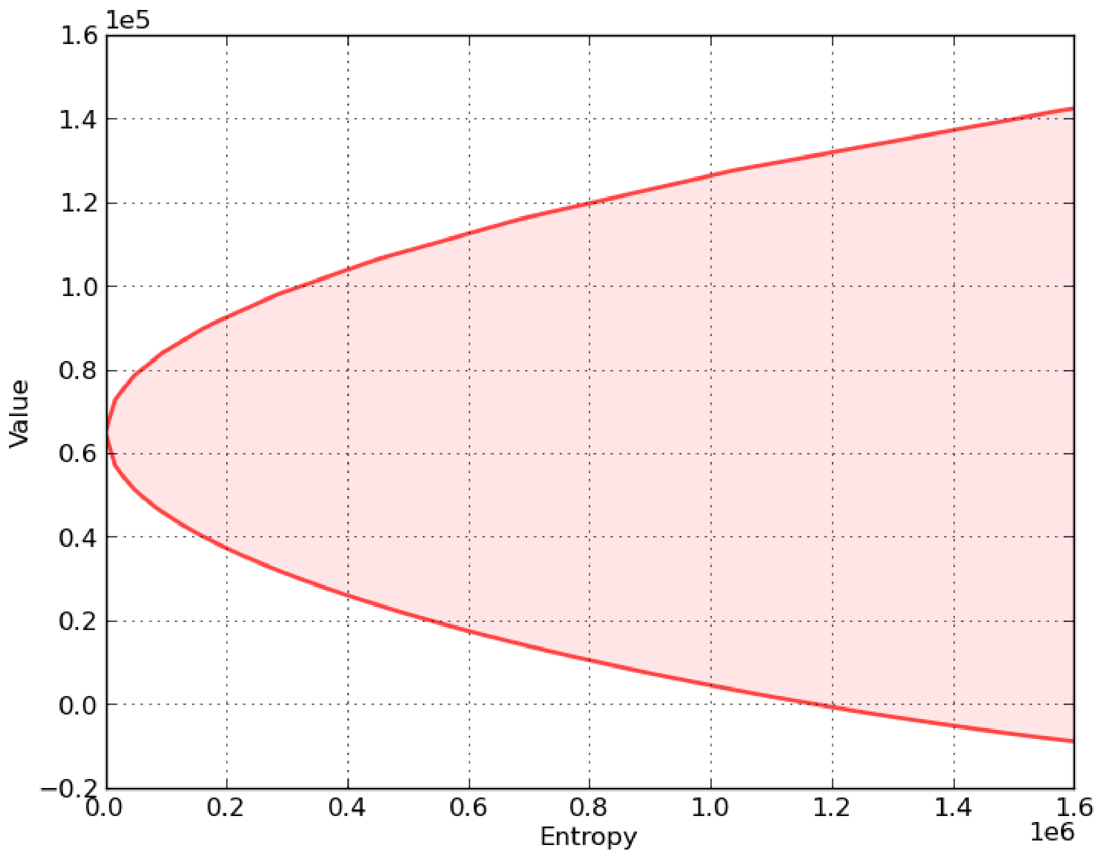
20.1.1 Sets of Models Imply Sets Of Values

Our "robust" decision-maker wants to know how well a given rule will work when he does not *know* a single transition law ….

… he wants to know *sets* of values that will be attained by a given decision rule F under a *set* of transition laws.

Ultimately, he wants to design a decision rule F that shapes these *sets* of values in ways that he prefers.

With this in mind, consider the following graph, which relates to a particular decision problem to be explained below



The figure shows a *value-entropy correspondence* for a particular decision rule F.

The shaded set is the graph of the correspondence, which maps entropy to a set of values associated with a set of models that surround the decision-maker's approximating model.

Here

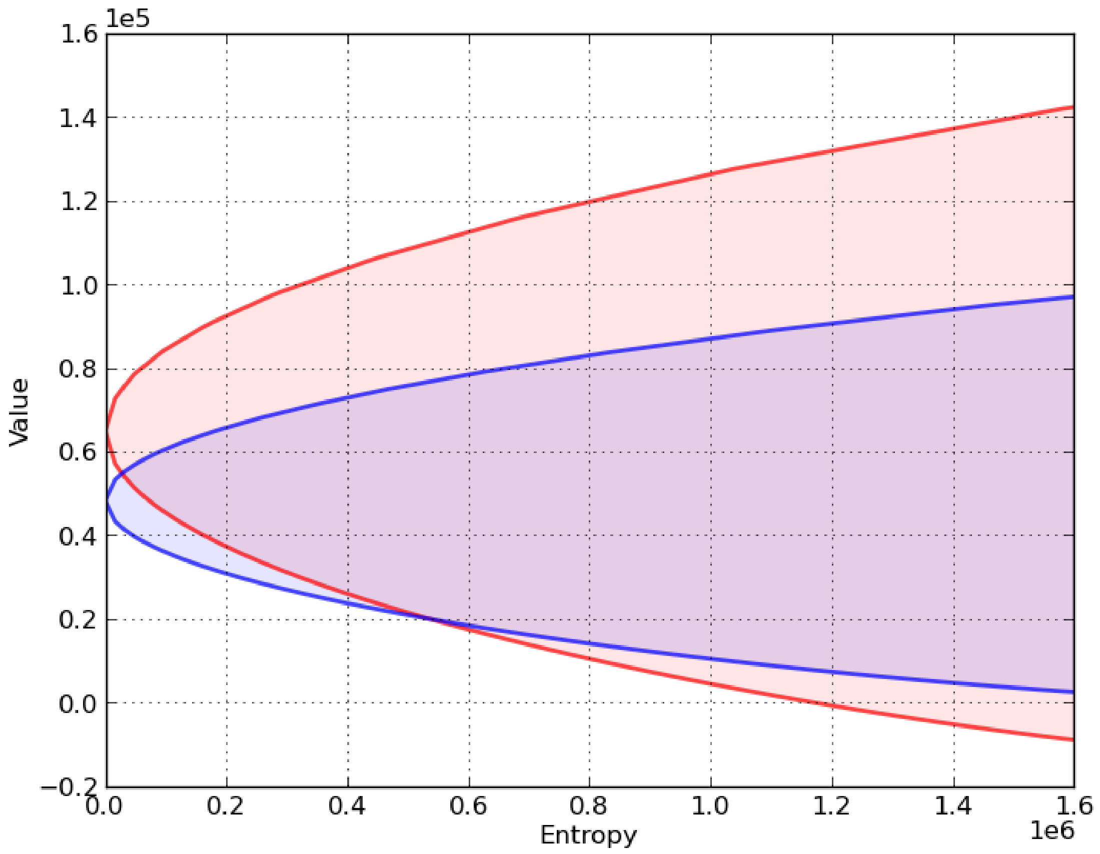
- *Value* refers to a sum of discounted rewards obtained by applying the decision rule F when the state starts at some fixed initial state x_0.
- *Entropy* is a non-negative number that measures the size of a set of models surrounding the decision-maker's approximating model.
  - Entropy is zero when the set includes only the approximating model, indicating that the decision-maker completely trusts the approximating model.
  - Entropy is bigger, and the set of surrounding models is bigger, the less the decision-maker trusts the approximating model.

The shaded region indicates that for **all** models having entropy less than or equal to the number on the horizontal axis, the value obtained will be somewhere within the indicated set of values.

Now let's compare sets of values associated with two different decision rules, F_r and F_b.

In the next figure,

- The red set shows the value-entropy correspondence for decision rule F_r.
- The blue set shows the value-entropy correspondence for decision rule F_b.



The blue correspondence is skinnier than the red correspondence.

This conveys the sense in which the decision rule F_b is *more robust* than the decision rule F_r

- *more robust* means that the set of values is less sensitive to *increasing misspecification* as measured by entropy

Notice that the less robust rule F_r promises higher values for small misspecifications (small entropy).

(But it is more fragile in the sense that it is more sensitive to perturbations of the approximating model)

Below we'll explain in detail how to construct these sets of values for a given F, but for now ….

Here is a hint about the *secret weapons* we'll use to construct these sets

- We'll use some min problems to construct the lower bounds
- We'll use some max problems to construct the upper bounds

We will also describe how to choose F to shape the sets of values.

This will involve crafting a *skinnier* set at the cost of a lower *level* (at least for low values of entropy).

20.1.2 Inspiring Video

If you want to understand more about why one serious quantitative researcher is interested in this approach, we recommend Lars Peter Hansen's Nobel lecture.

20.1.3 Other References

Our discussion in this lecture is based on

- [Hansen and Sargent, 2000]
- [Hansen and Sargent, 2008]

20.2 The Model

For simplicity, we present ideas in the context of a class of problems with linear transition laws and quadratic objective functions.

To fit in with our earlier lecture on LQ control, we will treat loss minimization rather than value maximization.

To begin, recall the infinite horizon LQ problem, where an agent chooses a sequence of controls $\{u_t\}$ to minimize

$$\sum_{t=0}^{\infty} \beta^t \left\{ x_t' R x_t + u_t' Q u_t \right\} \tag{20.1}$$

subject to the linear law of motion

$$x_{t+1} = A x_t + B u_t + C w_{t+1}, \qquad t = 0, 1, 2, \ldots \tag{20.2}$$

As before,

- x_t is $n \times 1$, A is $n \times n$
- u_t is $k \times 1$, B is $n \times k$
- w_t is $j \times 1$, C is $n \times j$
- R is $n \times n$ and Q is $k \times k$

Here x_t is the state, u_t is the control, and w_t is a shock vector.

For now, we take $\{w_t\} := \{w_t\}_{t=1}^{\infty}$ to be deterministic — a single fixed sequence.

We also allow for *model uncertainty* on the part of the agent solving this optimization problem.

In particular, the agent takes $w_t = 0$ for all $t \geq 0$ as a benchmark model but admits the possibility that this model might be wrong.

As a consequence, she also considers a set of alternative models expressed in terms of sequences $\{w_t\}$ that are "close" to the zero sequence.

She seeks a policy that will do well enough for a set of alternative models whose members are pinned down by sequences $\{w_t\}$.

Soon we'll quantify the quality of a model specification in terms of the maximal size of the expression $\sum_{t=0}^{\infty} \beta^{t+1} w_{t+1}' w_{t+1}$.

20.3 Constructing More Robust Policies

If our agent takes $\{w_t\}$ as a given deterministic sequence, then, drawing on intuition from earlier lectures on dynamic programming, we can anticipate Bellman equations such as

$$J_{t-1}(x) = \min_u \{x' R x + u' Q u + \beta \, J_t(Ax + Bu + C w_t)\}$$

(Here J depends on t because the sequence $\{w_t\}$ is not recursive)

Our tool for studying robustness is to construct a rule that works well even if an adverse sequence $\{w_t\}$ occurs.

In our framework, "adverse" means "loss increasing".

As we'll see, this will eventually lead us to construct the Bellman equation

$$J(x) = \min_u \max_w \{x' R x + u' Q u + \beta \, [J(Ax + Bu + Cw) - \theta w' w]\} \tag{20.3}$$

Notice that we've added the penalty term $-\theta w' w$.

Since $w' w = \|w\|^2$, this term becomes influential when w moves away from the origin.

The penalty parameter θ controls how much we penalize the maximizing agent for "harming" the minimizing agent.

By raising θ more and more, we more and more limit the ability of maximizing agent to distort outcomes relative to the approximating model.

So bigger θ is implicitly associated with smaller distortion sequences $\{w_t\}$.

20.3.1 Analyzing the Bellman Equation

So what does J in (20.3) look like?

As with the ordinary LQ control model, J takes the form $J(x) = x' P x$ for some symmetric positive definite matrix P.

One of our main tasks will be to analyze and compute the matrix P.

Related tasks will be to study associated feedback rules for u_t and w_{t+1}.

First, using matrix calculus, you will be able to verify that

$$\begin{aligned} \max_w & \{(Ax + Bu + Cw)' P (Ax + Bu + Cw) - \theta w' w\} \\ & = (Ax + Bu)' \mathcal{D}(P) (Ax + Bu) \end{aligned} \tag{20.4}$$

where

$$\mathcal{D}(P) := P + PC(\theta I - C'PC)^{-1}C'P \tag{20.5}$$

and I is a $j \times j$ identity matrix. Substituting this expression for the maximum into (20.3) yields

$$x'Px = \min_u \{x'Rx + u'Qu + \beta\,(Ax+Bu)'\mathcal{D}(P)(Ax+Bu)\} \tag{20.6}$$

Using similar mathematics, the solution to this minimization problem is $u = -Fx$ where $F := (Q + \beta B'\mathcal{D}(P)B)^{-1}\beta B'\mathcal{D}(P)A$.

Substituting this minimizer back into (20.6) and working through the algebra gives $x'Px = x'\mathcal{B}(\mathcal{D}(P))x$ for all x, or, equivalently,

$$P = \mathcal{B}(\mathcal{D}(P))$$

where $\mathcal{D}$ is the operator defined in (20.5) and

$$\mathcal{B}(P) := R - \beta^2 A'PB(Q + \beta B'PB)^{-1}B'PA + \beta A'PA$$

The operator $\mathcal{B}$ is the standard (i.e., non-robust) LQ Bellman operator, and $P = \mathcal{B}(P)$ is the standard matrix Riccati equation coming from the Bellman equation — see this discussion.

Under some regularity conditions (see [Hansen and Sargent, 2008]), the operator $\mathcal{B} \circ \mathcal{D}$ has a unique positive definite fixed point, which we denote below by $\hat{P}$.

A robust policy, indexed by θ, is $u = -\hat{F}x$ where

$$\hat{F} := (Q + \beta B'\mathcal{D}(\hat{P})B)^{-1}\beta B'\mathcal{D}(\hat{P})A \tag{20.7}$$

We also define

$$\hat{K} := (\theta I - C'\hat{P}C)^{-1}C'\hat{P}(A - B\hat{F}) \tag{20.8}$$

The interpretation of $\hat{K}$ is that $w_{t+1} = \hat{K}x_t$ on the worst-case path of $\{x_t\}$, in the sense that this vector is the maximizer of (20.4) evaluated at the fixed rule $u = -\hat{F}x$.

Note that $\hat{P}, \hat{F}, \hat{K}$ are all determined by the primitives and θ.

Note also that if θ is very large, then $\mathcal{D}$ is approximately equal to the identity mapping.

Hence, when θ is large, $\hat{P}$ and $\hat{F}$ are approximately equal to their standard LQ values.

Furthermore, when θ is large, $\hat{K}$ is approximately equal to zero.

Conversely, smaller θ is associated with greater fear of model misspecification and greater concern for robustness.

20.4 Robustness as Outcome of a Two-Person Zero-Sum Game

What we have done above can be interpreted in terms of a two-person zero-sum game in which $\hat{F}, \hat{K}$ are Nash equilibrium objects.

Agent 1 is our original agent, who seeks to minimize loss in the LQ program while admitting the possibility of misspecification.

Agent 2 is an imaginary malevolent player.

Agent 2's malevolence helps the original agent to compute bounds on his value function across a set of models.

We begin with agent 2's problem.

20.4.1 Agent 2's Problem

Agent 2

1. knows a fixed policy F specifying the behavior of agent 1, in the sense that $u_t = -Fx_t$ for all t
2. responds by choosing a shock sequence $\{w_t\}$ from a set of paths sufficiently close to the benchmark sequence $\{0, 0, 0, \ldots\}$

A natural way to say "sufficiently close to the zero sequence" is to restrict the summed inner product $\sum_{t=1}^{\infty} w_t' w_t$ to be small.

However, to obtain a time-invariant recursive formulation, it turns out to be convenient to restrict a discounted inner product

$$\sum_{t=1}^{\infty} \beta^t w_t' w_t \leq \eta \tag{20.9}$$

Now let F be a fixed policy, and let $J_F(x_0, \mathbf{w})$ be the present-value cost of that policy given sequence $\mathbf{w} := \{w_t\}$ and initial condition $x_0 \in \mathbb{R}^n$.

Substituting $-Fx_t$ for u_t in (20.1), this value can be written as

$$J_F(x_0, \mathbf{w}) := \sum_{t=0}^{\infty} \beta^t x_t'(R + F'QF)x_t \tag{20.10}$$

where

$$x_{t+1} = (A - BF)x_t + Cw_{t+1} \tag{20.11}$$

and the initial condition x_0 is as specified in the left side of (20.10).

Agent 2 chooses $\mathbf{w}$ to maximize agent 1's loss $J_F(x_0, \mathbf{w})$ subject to (20.9).

Using a Lagrangian formulation, we can express this problem as

$$\max_{\mathbf{w}} \sum_{t=0}^{\infty} \beta^t \left\{ x_t'(R + F'QF)x_t - \beta\theta(w_{t+1}' w_{t+1} - \eta) \right\}$$

where $\{x_t\}$ satisfied (20.11) and θ is a Lagrange multiplier on constraint (20.9).

For the moment, let's take θ as fixed, allowing us to drop the constant $\beta\theta\eta$ term in the objective function, and hence write the problem as

$$\max_{\mathbf{w}} \sum_{t=0}^{\infty} \beta^t \left\{ x_t'(R + F'QF)x_t - \beta\theta w_{t+1}' w_{t+1} \right\}$$

or, equivalently,

$$\min_{\mathbf{w}} \sum_{t=0}^{\infty} \beta^t \left\{ -x_t'(R + F'QF)x_t + \beta\theta w_{t+1}' w_{t+1} \right\} \tag{20.12}$$

subject to (20.11).

What's striking about this optimization problem is that it is once again an LQ discounted dynamic programming problem, with $\mathbf{w} = \{w_t\}$ as the sequence of controls.

The expression for the optimal policy can be found by applying the usual LQ formula (see here).

We denote it by $K(F, \theta)$, with the interpretation $w_{t+1} = K(F, \theta)x_t$.

The remaining step for agent 2's problem is to set θ to enforce the constraint (20.9), which can be done by choosing $\theta = \theta_\eta$ such that

$$\beta \sum_{t=0}^{\infty} \beta^t x_t' K(F, \theta_\eta)' K(F, \theta_\eta) x_t = \eta \tag{20.13}$$

Here x_t is given by (20.11) — which in this case becomes $x_{t+1} = (A - BF + CK(F, \theta))x_t$.

20.4.2 Using Agent 2's Problem to Construct Bounds on the Value Sets

The Lower Bound

Define the minimized object on the right side of problem (20.12) as $R_\theta(x_0, F)$.

Because "minimizers minimize" we have

$$R_\theta(x_0, F) \leq \sum_{t=0}^{\infty} \beta^t \left\{ -x_t'(R + F'QF)x_t \right\} + \beta\theta \sum_{t=0}^{\infty} \beta^t w_{t+1}' w_{t+1},$$

where $x_{t+1} = (A - BF + CK(F, \theta))x_t$ and x_0 is a given initial condition.

This inequality in turn implies the inequality

$$R_\theta(x_0, F) - \theta \, \text{ent} \leq \sum_{t=0}^{\infty} \beta^t \left\{ -x_t'(R + F'QF)x_t \right\} \tag{20.14}$$

where

$$\text{ent} := \beta \sum_{t=0}^{\infty} \beta^t w_{t+1}' w_{t+1}$$

The left side of inequality (20.14) is a straight line with slope $-\theta$.

Technically, it is a "separating hyperplane".

At a particular value of entropy, the line is tangent to the lower bound of values as a function of entropy.

In particular, the lower bound on the left side of (20.14) is attained when

$$\text{ent} = \beta \sum_{t=0}^{\infty} \beta^t x_t' K(F, \theta)' K(F, \theta) x_t \tag{20.15}$$

To construct the *lower bound* on the set of values associated with all perturbations **w** satisfying the entropy constraint (20.9) at a given entropy level, we proceed as follows:

- For a given θ, solve the minimization problem (20.12).
- Compute the minimizer $R_\theta(x_0, F)$ and the associated entropy using (20.15).
- Compute the lower bound on the value function $R_\theta(x_0, F) - \theta$ ent and plot it against ent.
- Repeat the preceding three steps for a range of values of θ to trace out the lower bound.

Note: This procedure sweeps out a set of separating hyperplanes indexed by different values for the Lagrange multiplier θ.

The Upper Bound

To construct an *upper bound* we use a very similar procedure.

We simply replace the *minimization* problem (20.12) with the *maximization* problem

$$V_{\tilde\theta}(x_0, F) = \max_{\mathbf{w}} \sum_{t=0}^{\infty} \beta^t \left\{ -x_t'(R + F'QF)x_t - \beta\tilde\theta w_{t+1}' w_{t+1} \right\} \tag{20.16}$$

where now $\tilde\theta > 0$ penalizes the choice of **w** with larger entropy.

(Notice that $\tilde\theta = -\theta$ in problem (20.12))

Because "maximizers maximize" we have

$$V_{\tilde\theta}(x_0, F) \geq \sum_{t=0}^{\infty} \beta^t \left\{ -x_t'(R + F'QF)x_t \right\} - \beta\tilde\theta \sum_{t=0}^{\infty} \beta^t w_{t+1}' w_{t+1}$$

which in turn implies the inequality

$$V_{\tilde\theta}(x_0, F) + \tilde\theta \, \mathrm{ent} \geq \sum_{t=0}^{\infty} \beta^t \left\{ -x_t'(R + F'QF)x_t \right\} \tag{20.17}$$

where

$$\mathrm{ent} \equiv \beta \sum_{t=0}^{\infty} \beta^t w_{t+1}' w_{t+1}$$

The left side of inequality (20.17) is a straight line with slope $\tilde\theta$.

The upper bound on the left side of (20.17) is attained when

$$\mathrm{ent} = \beta \sum_{t=0}^{\infty} \beta^t x_t' K(F, \tilde\theta)' K(F, \tilde\theta) x_t \tag{20.18}$$

To construct the *upper bound* on the set of values associated all perturbations **w** with a given entropy we proceed much as we did for the lower bound

- For a given $\tilde\theta$, solve the maximization problem (20.16).
- Compute the maximizer $V_{\tilde\theta}(x_0, F)$ and the associated entropy using (20.18).
- Compute the upper bound on the value function $V_{\tilde\theta}(x_0, F) + \tilde\theta \, \mathrm{ent}$ and plot it against ent.
- Repeat the preceding three steps for a range of values of $\tilde\theta$ to trace out the upper bound.

Reshaping the Set of Values

Now in the interest of *reshaping* these sets of values by choosing F, we turn to agent 1's problem.

20.4.3 Agent 1's Problem

Now we turn to agent 1, who solves

$$\min_{\{u_t\}} \sum_{t=0}^{\infty} \beta^t \left\{ x_t' R x_t + u_t' Q u_t - \beta\theta w_{t+1}' w_{t+1} \right\} \tag{20.19}$$

where $\{w_{t+1}\}$ satisfies $w_{t+1} = Kx_t$.

In other words, agent 1 minimizes

$$\sum_{t=0}^{\infty} \beta^t \left\{ x_t'(R - \beta\theta K'K)x_t + u_t'Qu_t \right\} \tag{20.20}$$

subject to

$$x_{t+1} = (A + CK)x_t + Bu_t \tag{20.21}$$

Once again, the expression for the optimal policy can be found here — we denote it by $\tilde{F}$.

20.4.4 Nash Equilibrium

Clearly, the $\tilde{F}$ we have obtained depends on K, which, in agent 2's problem, depended on an initial policy F.

Holding all other parameters fixed, we can represent this relationship as a mapping Φ, where

$$\tilde{F} = \Phi(K(F, \theta))$$

The map $F \mapsto \Phi(K(F, \theta))$ corresponds to a situation in which

1. agent 1 uses an arbitrary initial policy F
2. agent 2 best responds to agent 1 by choosing $K(F, \theta)$
3. agent 1 best responds to agent 2 by choosing $\tilde{F} = \Phi(K(F, \theta))$

As you may have already guessed, the robust policy $\hat{F}$ defined in (20.7) is a fixed point of the mapping Φ.

In particular, for any given θ,

1. $K(\hat{F}, \theta) = \hat{K}$, where $\hat{K}$ is as given in (20.8)
2. $\Phi(\hat{K}) = \hat{F}$

A sketch of the proof is given in *the appendix*.

20.5 The Stochastic Case

Now we turn to the stochastic case, where the sequence $\{w_t\}$ is treated as an IID sequence of random vectors.

In this setting, we suppose that our agent is uncertain about the *conditional probability distribution* of w_{t+1}.

The agent takes the standard normal distribution $N(0, I)$ as the baseline conditional distribution, while admitting the possibility that other "nearby" distributions prevail.

These alternative conditional distributions of w_{t+1} might depend nonlinearly on the history $x_s, s \leq t$.

To implement this idea, we need a notion of what it means for one distribution to be near another one.

Here we adopt a very useful measure of closeness for distributions known as the *relative entropy*, or Kullback-Leibler divergence.

For densities p, q, the Kullback-Leibler divergence of q from p is defined as

$$D_{KL}(p, q) := \int \ln \left[\frac{p(x)}{q(x)} \right] p(x) \, dx$$

Using this notation, we replace (20.3) with the stochastic analog

$$J(x) = \min_u \max_{\psi \in \mathcal{P}} \left\{ x'Rx + u'Qu + \beta \left[\int J(Ax + Bu + Cw) \, \psi(dw) - \theta D_{KL}(\psi, \phi) \right] \right\} \tag{20.22}$$

Here $\mathcal{P}$ represents the set of all densities on $\mathbb{R}^n$ and ϕ is the benchmark distribution $N(0, I)$.

The distribution ϕ is chosen as the least desirable conditional distribution in terms of next period outcomes, while taking into account the penalty term $\theta D_{KL}(\psi, \phi)$.

This penalty term plays a role analogous to the one played by the deterministic penalty $\theta w'w$ in (20.3), since it discourages large deviations from the benchmark.

20.5.1 Solving the Model

The maximization problem in (20.22) appears highly nontrivial — after all, we are maximizing over an infinite dimensional space consisting of the entire set of densities.

However, it turns out that the solution is tractable, and in fact also falls within the class of normal distributions.

First, we note that J has the form $J(x) = x'Px + d$ for some positive definite matrix P and constant real number d.

Moreover, it turns out that if $(I - \theta^{-1} C'PC)^{-1}$ is nonsingular, then

$$\begin{aligned} \max_{\psi \in \mathcal{P}} & \left\{ \int (Ax + Bu + Cw)' P (Ax + Bu + Cw) \, \psi(dw) - \theta D_{KL}(\psi, \phi) \right\} \\ & = (Ax + Bu)' \mathcal{D}(P) (Ax + Bu) + \kappa(\theta, P) \end{aligned} \tag{20.23}$$

where

$$\kappa(\theta, P) := \theta \ln[\det(I - \theta^{-1} C'PC)^{-1}]$$

and the maximizer is the Gaussian distribution

$$\psi = N\left((\theta I - C'PC)^{-1} C'P(Ax + Bu), (I - \theta^{-1} C'PC)^{-1} \right) \tag{20.24}$$

Substituting the expression for the maximum into Bellman equation (20.22) and using $J(x) = x'Px + d$ gives

$$x'Px + d = \min_u \left\{ x'Rx + u'Qu + \beta \, (Ax + Bu)' \mathcal{D}(P)(Ax + Bu) + \beta \, [d + \kappa(\theta, P)] \right\} \tag{20.25}$$

Since constant terms do not affect minimizers, the solution is the same as (20.6), leading to

$$x'Px + d = x' \mathcal{B}(\mathcal{D}(P)) x + \beta \, [d + \kappa(\theta, P)]$$

To solve this Bellman equation, we take $\hat{P}$ to be the positive definite fixed point of $\mathcal{B} \circ \mathcal{D}$.

In addition, we take $\hat{d}$ as the real number solving $d = \beta \, [d + \kappa(\theta, P)]$, which is

$$\hat{d} := \frac{\beta}{1 - \beta} \kappa(\theta, P) \tag{20.26}$$

The robust policy in this stochastic case is the minimizer in (20.25), which is once again $u = -\hat{F}x$ for $\hat{F}$ given by (20.7).

Substituting the robust policy into (20.24) we obtain the worst-case shock distribution:

$$w_{t+1} \sim N(\hat{K} x_t, (I - \theta^{-1} C' \hat{P} C)^{-1})$$

where $\hat{K}$ is given by (20.8).

Note that the mean of the worst-case shock distribution is equal to the same worst-case w_{t+1} as in the earlier deterministic setting.

20.5.2 Computing Other Quantities

Before turning to implementation, we briefly outline how to compute several other quantities of interest.

Worst-Case Value of a Policy

One thing we will be interested in doing is holding a policy fixed and computing the discounted loss associated with that policy.

So let F be a given policy and let $J_F(x)$ be the associated loss, which, by analogy with (20.22), satisfies

$$J_F(x) = \max_{\psi \in \mathcal{P}} \left\{ x'(R + F'QF)x + \beta \left[\int J_F((A - BF)x + Cw) \, \psi(dw) - \theta D_{KL}(\psi, \phi) \right] \right\}$$

Writing $J_F(x) = x' P_F x + d_F$ and applying the same argument used to derive (20.23) we get

$$x' P_F x + d_F = x'(R + F'QF)x + \beta \left[x'(A - BF)' \mathcal{D}(P_F)(A - BF)x + d_F + \kappa(\theta, P_F) \right]$$

To solve this we take P_F to be the fixed point

$$P_F = R + F'QF + \beta(A - BF)' \mathcal{D}(P_F)(A - BF)$$

and

$$d_F := \frac{\beta}{1 - \beta} \kappa(\theta, P_F) = \frac{\beta}{1 - \beta} \theta \ln[\det(I - \theta^{-1} C' P_F C)^{-1}] \tag{20.27}$$

If you skip ahead to *the appendix*, you will be able to verify that $-P_F$ is the solution to the Bellman equation in agent 2's problem *discussed above* — we use this in our computations.

20.6 Implementation

The QuantEcon.py package provides a class called `RBLQ` for implementation of robust LQ optimal control.

The code can be found on GitHub.

Here is a brief description of the methods of the class

- `d_operator()` and `b_operator()` implement $\mathcal{D}$ and $\mathcal{B}$ respectively
- `robust_rule()` and `robust_rule_simple()` both solve for the triple $\hat{F}, \hat{K}, \hat{P}$, as described in equations (20.7) – (20.8) and the surrounding discussion
  - `robust_rule()` is more efficient
  - `robust_rule_simple()` is more transparent and easier to follow
- `K_to_F()` and `F_to_K()` solve the decision problems of *agent 1* and *agent 2* respectively
- `compute_deterministic_entropy()` computes the left-hand side of (20.13)
- `evaluate_F()` computes the loss and entropy associated with a given policy — see *this discussion*

20.7 Application

Let us consider a monopolist similar to this one, but now facing model uncertainty.

The inverse demand function is $p_t = a_0 - a_1 y_t + d_t$.

where

$$d_{t+1} = \rho d_t + \sigma_d w_{t+1}, \quad \{w_t\} \overset{\text{IID}}{\sim} N(0,1)$$

and all parameters are strictly positive.

The period return function for the monopolist is

$$r_t = p_t y_t - \gamma \frac{(y_{t+1} - y_t)^2}{2} - c y_t$$

Its objective is to maximize expected discounted profits, or, equivalently, to minimize $\mathbb{E} \sum_{t=0}^{\infty} \beta^t (-r_t)$.

To form a linear regulator problem, we take the state and control to be

$$x_t = \begin{bmatrix} 1 \\ y_t \\ d_t \end{bmatrix} \quad \text{and} \quad u_t = y_{t+1} - y_t$$

Setting $b := (a_0 - c)/2$ we define

$$R = -\begin{bmatrix} 0 & b & 0 \\ b & -a_1 & 1/2 \\ 0 & 1/2 & 0 \end{bmatrix} \quad \text{and} \quad Q = \gamma/2$$

For the transition matrices, we set

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \rho \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \qquad C = \begin{bmatrix} 0 \\ 0 \\ \sigma_d \end{bmatrix}$$

Our aim is to compute the value-entropy correspondences *shown above*.

The parameters are

$$a_0 = 100, a_1 = 0.5, \rho = 0.9, \sigma_d = 0.05, \beta = 0.95, c = 2, \gamma = 50.0$$

The standard normal distribution for w_t is understood as the agent's baseline, with uncertainty parameterized by θ.

We compute value-entropy correspondences for two policies

1. The no concern for robustness policy F_0, which is the ordinary LQ loss minimizer.
2. A "moderate" concern for robustness policy F_b, with $\theta = 0.02$.

The code for producing the graph shown above, with blue being for the robust policy, is as follows

```
# Model parameters

a_0 = 100
a_1 = 0.5
ρ = 0.9
σ_d = 0.05
β = 0.95
```

(continues on next page)

(continued from previous page)

```
c = 2
γ = 50.0

θ = 0.002
ac = (a_0 - c) / 2.0

# Define LQ matrices

R = np.array([[0.,   ac,   0.],
              [ac, -a_1,  0.5],
              [0.,  0.5,  0.]])

R = -R  # For minimization
Q = γ / 2

A = np.array([[1., 0., 0.],
              [0., 1., 0.],
              [0., 0., ρ]])
B = np.array([[0.],
              [1.],
              [0.]])
C = np.array([[0.],
              [0.],
              [σ_d]])

# ----------------------------------------------------------------------- #
#                                Functions
# ----------------------------------------------------------------------- #


def evaluate_policy(θ, F):

    """
    Given θ (scalar, dtype=float) and policy F (array_like), returns the
    value associated with that policy under the worst case path for {w_t},
    as well as the entropy level.
    """

    rlq = qe.RBLQ(Q, R, A, B, C, β, θ)
    K_F, P_F, d_F, O_F, o_F = rlq.evaluate_F(F)
    x0 = np.array([[1.], [0.], [0.]])
    value = - x0.T @ P_F @ x0 - d_F
    entropy = x0.T @ O_F @ x0 + o_F
    return list(map(float, (value, entropy)))


def value_and_entropy(emax, F, bw, grid_size=1000):

    """
    Compute the value function and entropy levels for a θ path
    increasing until it reaches the specified target entropy value.

    Parameters
    ==========
    emax: scalar
        The target entropy value
```

(continues on next page)

(continued from previous page)

```
    F: array_like
        The policy function to be evaluated

    bw: str
        A string specifying whether the implied shock path follows best
        or worst assumptions. The only acceptable values are 'best' and
        'worst'.

    Returns
    =======
    df: pd.DataFrame
        A pandas DataFrame containing the value function and entropy
        values up to the emax parameter. The columns are 'value' and
        'entropy'.
    """

    if bw == 'worst':
        θs = 1 / np.linspace(1e-8, 1000, grid_size)
    else:
        θs = -1 / np.linspace(1e-8, 1000, grid_size)

    df = pd.DataFrame(index=θs, columns=('value', 'entropy'))

    for θ in θs:
        df.loc[θ] = evaluate_policy(θ, F)
        if df.loc[θ, 'entropy'] >= emax:
            break

    df = df.dropna(how='any')
    return df


# ------------------------------------------------------------------------ #
#                                   Main
# ------------------------------------------------------------------------ #


# Compute the optimal rule
optimal_lq = qe.LQ(Q, R, A, B, C, beta=β)
Po, Fo, do = optimal_lq.stationary_values()

# Compute a robust rule given θ
baseline_robust = qe.RBLQ(Q, R, A, B, C, β, θ)
Fb, Kb, Pb = baseline_robust.robust_rule()

# Check the positive definiteness of worst-case covariance matrix to
# ensure that θ exceeds the breakdown point
test_matrix = np.identity(Pb.shape[0]) - (C.T @ Pb @ C) / θ
eigenvals, eigenvecs = eig(test_matrix)
assert (eigenvals >= 0).all(), 'θ below breakdown point.'


emax = 1.6e6

optimal_best_case = value_and_entropy(emax, Fo, 'best')
```

(continues on next page)

(continued from previous page)

```python
robust_best_case = value_and_entropy(emax, Fb, 'best')
optimal_worst_case = value_and_entropy(emax, Fo, 'worst')
robust_worst_case = value_and_entropy(emax, Fb, 'worst')

fig, ax = plt.subplots()

ax.set_xlim(0, emax)
ax.set_ylabel("Value")
ax.set_xlabel("Entropy")
ax.grid()

for axis in 'x', 'y':
    plt.ticklabel_format(style='sci', axis=axis, scilimits=(0, 0))

plot_args = {'lw': 2, 'alpha': 0.7}

colors = 'r', 'b'

df_pairs = ((optimal_best_case, optimal_worst_case),
            (robust_best_case, robust_worst_case))


class Curve:

    def __init__(self, x, y):
        self.x, self.y = x, y

    def __call__(self, z):
        return np.interp(z, self.x, self.y)


for c, df_pair in zip(colors, df_pairs):
    curves = []
    for df in df_pair:
        # Plot curves
        x, y = df['entropy'], df['value']
        x, y = (np.asarray(a, dtype='float') for a in (x, y))
        egrid = np.linspace(0, emax, 100)
        curve = Curve(x, y)
        print(ax.plot(egrid, curve(egrid), color=c, **plot_args))
        curves.append(curve)
    # Color fill between curves
    ax.fill_between(egrid,
                    curves[0](egrid),
                    curves[1](egrid),
                    color=c, alpha=0.1)

plt.show()
```

```
/tmp/ipykernel_2492/154157873.py:51: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  return list(map(float, (value, entropy)))
/tmp/ipykernel_2492/154157873.py:51: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
```

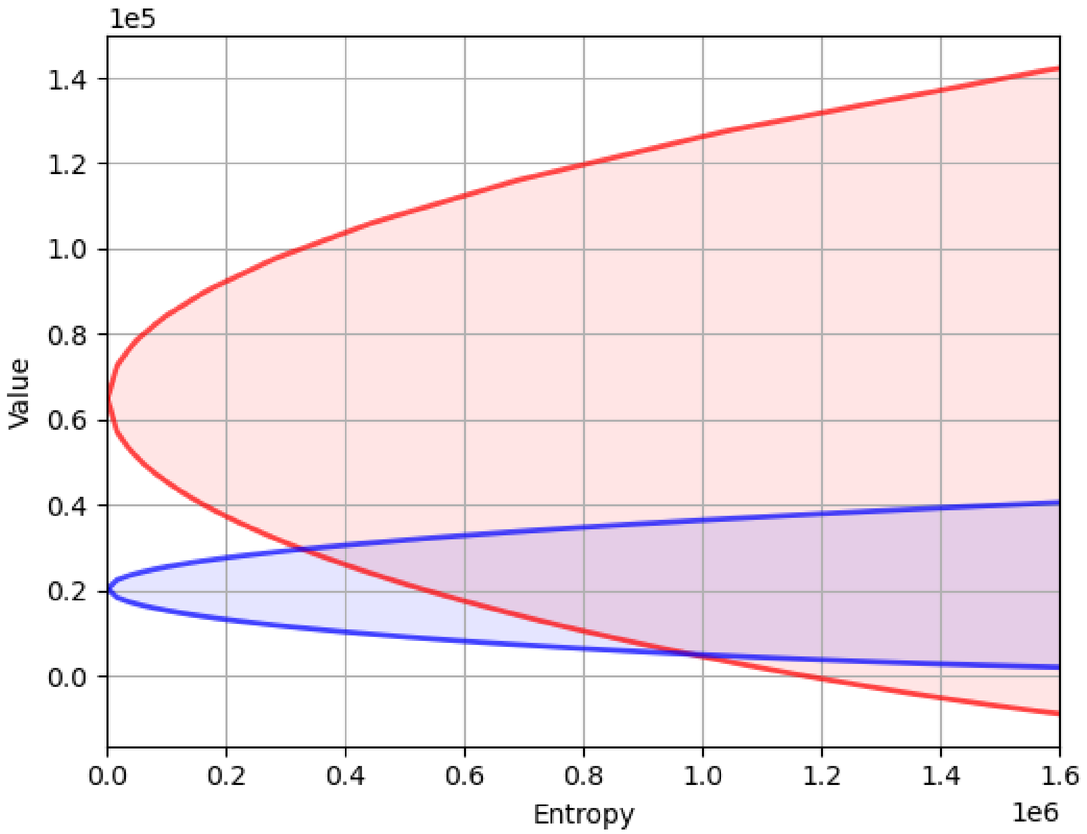
(continues on next page)

(continued from previous page)

```
  return list(map(float, (value, entropy)))
/tmp/ipykernel_2492/154157873.py:51: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  return list(map(float, (value, entropy)))
```

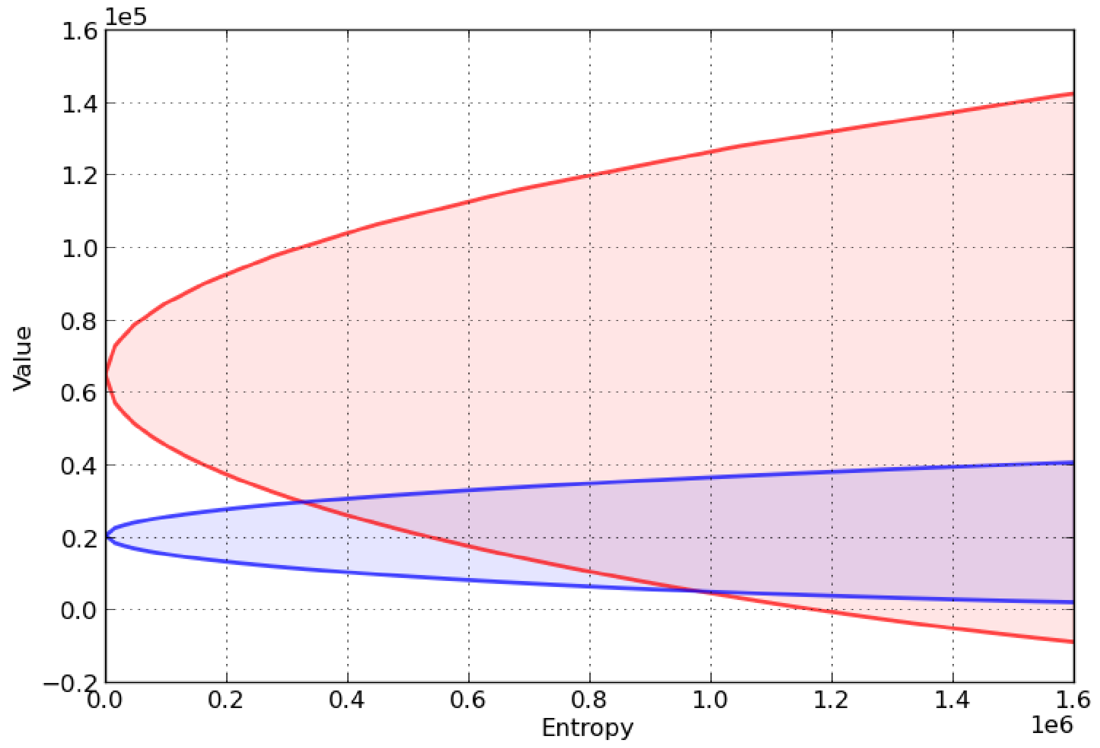
```
/tmp/ipykernel_2492/154157873.py:51: DeprecationWarning: Conversion of an array␣
↪with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you␣
↪extract a single element from your array before performing this operation.␣
↪(Deprecated NumPy 1.25.)
  return list(map(float, (value, entropy)))
```

```
[<matplotlib.lines.Line2D object at 0x7fce8234d5d0>]
[<matplotlib.lines.Line2D object at 0x7fce810208d0>]
[<matplotlib.lines.Line2D object at 0x7fce81021810>]
[<matplotlib.lines.Line2D object at 0x7fce81021bd0>]
```



Here's another such figure, with $\theta = 0.002$ instead of 0.02

Can you explain the different shape of the value-entropy correspondence for the robust policy?



20.8 Appendix

We sketch the proof only of the first claim in *this section*, which is that, for any given θ, $K(\hat{F}, \theta) = \hat{K}$, where $\hat{K}$ is as given in (20.8).

This is the content of the next lemma.

Lemma. If $\hat{P}$ is the fixed point of the map $\mathcal{B} \circ \mathcal{D}$ and $\hat{F}$ is the robust policy as given in (20.7), then

$$K(\hat{F}, \theta) = (\theta I - C'\hat{P}C)^{-1}C'\hat{P}(A - B\hat{F}) \tag{20.28}$$

Proof: As a first step, observe that when $F = \hat{F}$, the Bellman equation associated with the LQ problem (20.11) – (20.12) is

$$\tilde{P} = -R - \hat{F}'Q\hat{F} - \beta^2(A - B\hat{F})'\tilde{P}C(\beta\theta I + \beta C'\tilde{P}C)^{-1}C'\tilde{P}(A - B\hat{F}) + \beta(A - B\hat{F})'\tilde{P}(A - B\hat{F}) \tag{20.29}$$

(revisit this discussion if you don't know where (20.29) comes from) and the optimal policy is

$$w_{t+1} = -\beta(\beta\theta I + \beta C'\tilde{P}C)^{-1}C'\tilde{P}(A - B\hat{F})x_t$$

Suppose for a moment that $-\hat{P}$ solves the Bellman equation (20.29).

In this case, the policy becomes

$$w_{t+1} = (\theta I - C'\hat{P}C)^{-1}C'\hat{P}(A - B\hat{F})x_t$$

which is exactly the claim in (20.28).

Hence it remains only to show that $-\hat{P}$ solves (20.29), or, in other words,

$$\hat{P} = R + \hat{F}'Q\hat{F} + \beta(A - B\hat{F})'\hat{P}C(\theta I - C'\hat{P}C)^{-1}C'\hat{P}(A - B\hat{F}) + \beta(A - B\hat{F})'\hat{P}(A - B\hat{F})$$

Using the definition of $\mathcal{D}$, we can rewrite the right-hand side more simply as

$$R + \hat{F}'Q\hat{F} + \beta(A - B\hat{F})'\mathcal{D}(\hat{P})(A - B\hat{F})$$

Although it involves a substantial amount of algebra, it can be shown that the latter is just $\hat{P}$.

Hint: Use the fact that $\hat{P} = \mathcal{B}(\mathcal{D}(\hat{P}))$

CHAPTER

TWENTYONE

ROBUST MARKOV PERFECT EQUILIBRIUM

In addition to what's in Anaconda, this lecture will need the following libraries:

```
!pip install --upgrade quantecon
```

21.1 Overview

This lecture describes a Markov perfect equilibrium with robust agents.

We focus on special settings with

- two players
- quadratic payoff functions
- linear transition rules for the state vector

These specifications simplify calculations and allow us to give a simple example that illustrates basic forces.

This lecture is based on ideas described in chapter 15 of [Hansen and Sargent, 2008] and in Markov perfect equilibrium and *Robustness*.

Let's start with some standard imports:

```
import numpy as np
import quantecon as qe
from scipy.linalg import solve
import matplotlib.pyplot as plt
```

21.1.1 Basic Setup

Decisions of two agents affect the motion of a state vector that appears as an argument of payoff functions of both agents.

As described in Markov perfect equilibrium, when decision-makers have no concerns about the robustness of their decision rules to misspecifications of the state dynamics, a Markov perfect equilibrium can be computed via backward recursion on two sets of equations

- a pair of Bellman equations, one for each agent.
- a pair of equations that express linear decision rules for each agent as functions of that agent's continuation value function as well as parameters of preferences and state transition matrices.

This lecture shows how a similar equilibrium concept and similar computational procedures apply when we impute concerns about robustness to both decision-makers.

A Markov perfect equilibrium with robust agents will be characterized by

- a pair of Bellman equations, one for each agent.
- a pair of equations that express linear decision rules for each agent as functions of that agent's continuation value function as well as parameters of preferences and state transition matrices.
- a pair of equations that express linear decision rules for worst-case shocks for each agent as functions of that agent's continuation value function as well as parameters of preferences and state transition matrices.

Below, we'll construct a robust firms version of the classic duopoly model with adjustment costs analyzed in Markov perfect equilibrium.

21.2 Linear Markov Perfect Equilibria with Robust Agents

As we saw in Markov perfect equilibrium, the study of Markov perfect equilibria in dynamic games with two players leads us to an interrelated pair of Bellman equations.

In linear quadratic dynamic games, these "stacked Bellman equations" become "stacked Riccati equations" with a tractable mathematical structure.

21.2.1 Modified Coupled Linear Regulator Problems

We consider a general linear quadratic regulator game with two players, each of whom fears model misspecifications.

We often call the players agents.

The agents share a common baseline model for the transition dynamics of the state vector

- this is a counterpart of a 'rational expectations' assumption of shared beliefs

But now one or more agents doubt that the baseline model is correctly specified.

The agents express the possibility that their baseline specification is incorrect by adding a contribution Cv_{it} to the time t transition law for the state

- C is the usual *volatility matrix* that appears in stochastic versions of optimal linear regulator problems.
- v_{it} is a possibly history-dependent vector of distortions to the dynamics of the state that agent i uses to represent misspecification of the original model.

For convenience, we'll start with a finite horizon formulation, where t_0 is the initial date and t_1 is the common terminal date.

Player i takes a sequence $\{u_{-it}\}$ as given and chooses a sequence $\{u_{it}\}$ to minimize and $\{v_{it}\}$ to maximize

$$\sum_{t=t_0}^{t_1-1} \beta^{t-t_0} \{x_t' R_i x_t + u_{it}' Q_i u_{it} + u_{-it}' S_i u_{-it} + 2x_t' W_i u_{it} + 2u_{-it}' M_i u_{it} - \theta_i v_{it}' v_{it}\} \tag{21.1}$$

while thinking that the state evolves according to

$$x_{t+1} = Ax_t + B_1 u_{1t} + B_2 u_{2t} + Cv_{it} \tag{21.2}$$

Here

- x_t is an $n \times 1$ state vector, u_{it} is a $k_i \times 1$ vector of controls for player i, and

- v_{it} is an $h \times 1$ vector of distortions to the state dynamics that concern player i
- R_i is $n \times n$
- S_i is $k_{-i} \times k_{-i}$
- Q_i is $k_i \times k_i$
- W_i is $n \times k_i$
- M_i is $k_{-i} \times k_i$
- A is $n \times n$
- B_i is $n \times k_i$
- C is $n \times h$
- $\theta_i \in [\underline{\theta}_i, +\infty]$ is a scalar multiplier parameter of player i

If $\theta_i = +\infty$, player i completely trusts the baseline model.

If $\theta_i <_\infty$, player i suspects that some other unspecified model actually governs the transition dynamics.

The term $\theta_i v_{it}' v_{it}$ is a time t contribution to an entropy penalty that an (imaginary) loss-maximizing agent inside agent i's mind charges for distorting the law of motion in a way that harms agent i.

- the imaginary loss-maximizing agent helps the loss-minimizing agent by helping him construct bounds on the behavior of his decision rule over a large **set** of alternative models of state transition dynamics.

21.2.2 Computing Equilibrium

We formulate a linear robust Markov perfect equilibrium as follows.

Player i employs linear decision rules $u_{it} = -F_{it} x_t$, where F_{it} is a $k_i \times n$ matrix.

Player i's malevolent alter ego employs decision rules $v_{it} = K_{it} x_t$ where K_{it} is an $h \times n$ matrix.

A robust Markov perfect equilibrium is a pair of sequences $\{F_{1t}, F_{2t}\}$ and a pair of sequences $\{K_{1t}, K_{2t}\}$ over $t = t_0, \ldots, t_1 - 1$ that satisfy

- $\{F_{1t}, K_{1t}\}$ solves player 1's robust decision problem, taking $\{F_{2t}\}$ as given, and
- $\{F_{2t}, K_{2t}\}$ solves player 2's robust decision problem, taking $\{F_{1t}\}$ as given.

If we substitute $u_{2t} = -F_{2t} x_t$ into (21.1) and (21.2), then player 1's problem becomes minimization-maximization of

$$\sum_{t=t_0}^{t_1-1} \beta^{t-t_0} \left\{ x_t' \Pi_{1t} x_t + u_{1t}' Q_1 u_{1t} + 2 u_{1t}' \Gamma_{1t} x_t - \theta_1 v_{1t}' v_{1t} \right\} \tag{21.3}$$

subject to

$$x_{t+1} = \Lambda_{1t} x_t + B_1 u_{1t} + C v_{1t} \tag{21.4}$$

where

- $\Lambda_{it} := A - B_{-i} F_{-it}$
- $\Pi_{it} := R_i + F_{-it}' S_i F_{-it}$
- $\Gamma_{it} := W_i' - M_i' F_{-it}$

This is an LQ robust dynamic programming problem of the type studied in the *Robustness* lecture, which can be solved by working backward.

Maximization with respect to distortion v_{1t} leads to the following version of the $\mathcal{D}$ operator from the *Robustness* lecture, namely

$$\mathcal{D}_1(P) := P + PC(\theta_1 I - C'PC)^{-1}C'P \tag{21.5}$$

The matrix F_{1t} in the policy rule $u_{1t} = -F_{1t}x_t$ that solves agent 1's problem satisfies

$$F_{1t} = (Q_1 + \beta B_1' \mathcal{D}_1(P_{1t+1})B_1)^{-1}(\beta B_1' \mathcal{D}_1(P_{1t+1})\Lambda_{1t} + \Gamma_{1t}) \tag{21.6}$$

where P_{1t} solves the matrix Riccati difference equation

$$P_{1t} = \Pi_{1t} - (\beta B_1' \mathcal{D}_1(P_{1t+1})\Lambda_{1t} + \Gamma_{1t})'(Q_1 + \beta B_1' \mathcal{D}_1(P_{1t+1})B_1)^{-1}(\beta B_1' \mathcal{D}_1(P_{1t+1})\Lambda_{1t} + \Gamma_{1t}) + \beta \Lambda_{1t}' \mathcal{D}_1(P_{1t+1})\Lambda_{1t} \tag{21.7}$$

Similarly, the policy that solves player 2's problem is

$$F_{2t} = (Q_2 + \beta B_2' \mathcal{D}_2(P_{2t+1})B_2)^{-1}(\beta B_2' \mathcal{D}_2(P_{2t+1})\Lambda_{2t} + \Gamma_{2t}) \tag{21.8}$$

where P_{2t} solves

$$P_{2t} = \Pi_{2t} - (\beta B_2' \mathcal{D}_2(P_{2t+1})\Lambda_{2t} + \Gamma_{2t})'(Q_2 + \beta B_2' \mathcal{D}_2(P_{2t+1})B_2)^{-1}(\beta B_2' \mathcal{D}_2(P_{2t+1})\Lambda_{2t} + \Gamma_{2t}) + \beta \Lambda_{2t}' \mathcal{D}_2(P_{2t+1})\Lambda_{2t} \tag{21.9}$$

Here in all cases $t = t_0, \dots, t_1 - 1$ and the terminal conditions are $P_{it_1} = 0$.

The solution procedure is to use equations (21.6), (21.7), (21.8), and (21.9), and "work backwards" from time $t_1 - 1$.

Since we're working backwards, P_{1t+1} and P_{2t+1} are taken as given at each stage.

Moreover, since

- some terms on the right-hand side of (21.6) contain F_{2t}
- some terms on the right-hand side of (21.8) contain F_{1t}

we need to solve these $k_1 + k_2$ equations simultaneously.

21.2.3 Key Insight

As in Markov perfect equilibrium, a key insight here is that equations (21.6) and (21.8) are linear in F_{1t} and F_{2t}.

After these equations have been solved, we can take F_{it} and solve for P_{it} in (21.7) and (21.9).

Notice how j's control law F_{jt} is a function of $\{F_{is}, s \geq t, i \neq j\}$.

Thus, agent i's choice of $\{F_{it}; t = t_0, \dots, t_1 - 1\}$ influences agent j's choice of control laws.

However, in the Markov perfect equilibrium of this game, each agent is assumed to ignore the influence that his choice exerts on the other agent's choice.

After these equations have been solved, we can also deduce associated sequences of worst-case shocks.

21.2.4 Worst-case Shocks

For agent i the maximizing or worst-case shock v_{it} is

$$v_{it} = K_{it} x_t$$

where

$$K_{it} = \theta_i^{-1}(I - \theta_i^{-1} C' P_{i,t+1} C)^{-1} C' P_{i,t+1}(A - B_1 F_{it} - B_2 F_{2t})$$

21.2.5 Infinite Horizon

We often want to compute the solutions of such games for infinite horizons, in the hope that the decision rules F_{it} settle down to be time-invariant as $t_1 \to +\infty$.

In practice, we usually fix t_1 and compute the equilibrium of an infinite horizon game by driving $t_0 \to -\infty$.

This is the approach we adopt in the next section.

21.2.6 Implementation

We use the function nnash_robust to compute a Markov perfect equilibrium of the infinite horizon linear quadratic dynamic game with robust planers in the manner described above.

21.3 Application

21.3.1 A Duopoly Model

Without concerns for robustness, the model is identical to the duopoly model from the Markov perfect equilibrium lecture.

To begin, we briefly review the structure of that model.

Two firms are the only producers of a good the demand for which is governed by a linear inverse demand function

$$p = a_0 - a_1(q_1 + q_2) \tag{21.10}$$

Here $p = p_t$ is the price of the good, $q_i = q_{it}$ is the output of firm $i = 1, 2$ at time t and $a_0 > 0, a_1 > 0$.

In (21.10) and what follows,

- the time subscript is suppressed when possible to simplify notation
- $\hat{x}$ denotes a next period value of variable x

Each firm recognizes that its output affects total output and therefore the market price.

The one-period payoff function of firm i is price times quantity minus adjustment costs:

$$\pi_i = p q_i - \gamma(\hat{q}_i - q_i)^2, \quad \gamma > 0, \tag{21.11}$$

Substituting the inverse demand curve (21.10) into (21.11) lets us express the one-period payoff as

$$\pi_i(q_i, q_{-i}, \hat{q}_i) = a_0 q_i - a_1 q_i^2 - a_1 q_i q_{-i} - \gamma(\hat{q}_i - q_i)^2, \tag{21.12}$$

where q_{-i} denotes the output of the firm other than i.

The objective of the firm is to maximize $\sum_{t=0}^{\infty} \beta^t \pi_{it}$.

Firm i chooses a decision rule that sets next period quantity $\hat{q}_i$ as a function f_i of the current state (q_i, q_{-i}).

This completes our review of the duopoly model without concerns for robustness.

Now we activate robustness concerns of both firms.

To map a robust version of the duopoly model into coupled robust linear-quadratic dynamic programming problems, we again define the state and controls as

$$x_t := \begin{bmatrix} 1 \\ q_{1t} \\ q_{2t} \end{bmatrix} \quad \text{and} \quad u_{it} := q_{i,t+1} - q_{it}, \quad i = 1, 2$$

If we write

$$x_t' R_i x_t + u_{it}' Q_i u_{it}$$

where $Q_1 = Q_2 = \gamma$,

$$R_1 := \begin{bmatrix} 0 & -\frac{a_0}{2} & 0 \\ -\frac{a_0}{2} & a_1 & \frac{a_1}{2} \\ 0 & \frac{a_1}{2} & 0 \end{bmatrix} \quad \text{and} \quad R_2 := \begin{bmatrix} 0 & 0 & -\frac{a_0}{2} \\ 0 & 0 & \frac{a_1}{2} \\ -\frac{a_0}{2} & \frac{a_1}{2} & a_1 \end{bmatrix}$$

then we recover the one-period payoffs (21.11) for the two firms in the duopoly model.

The law of motion for the state x_t is $x_{t+1} = Ax_t + B_1 u_{1t} + B_2 u_{2t}$ where

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad B_1 := \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad B_2 := \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

A robust decision rule of firm i will take the form $u_{it} = -F_i x_t$, inducing the following closed-loop system for the evolution of x in the Markov perfect equilibrium:

$$x_{t+1} = (A - B_1 F_1 - B_1 F_2) x_t \tag{21.13}$$

21.3.2 Parameters and Solution

Consider the duopoly model with parameter values of:

- $a_0 = 10$
- $a_1 = 2$
- $\beta = 0.96$
- $\gamma = 12$

From these, we computed the infinite horizon MPE without robustness using the code

```
import numpy as np
import quantecon as qe

# Parameters
a0 = 10.0
a1 = 2.0
β = 0.96
```

(continues on next page)

(continued from previous page)

```python
γ = 12.0

# In LQ form
A = np.eye(3)
B1 = np.array([[0.], [1.], [0.]])
B2 = np.array([[0.], [0.], [1.]])


R1 = [[      0.,      -a0 / 2,          0.],
      [-a0 / 2.,          a1,     a1 / 2.],
      [       0,     a1 / 2.,          0.]]

R2 = [[     0.,           0.,      -a0 / 2],
      [     0.,           0.,      a1 / 2.],
      [-a0 / 2,      a1 / 2.,           a1]]

Q1 = Q2 = γ
S1 = S2 = W1 = W2 = M1 = M2 = 0.0

# Solve using QE's nnash function
F1, F2, P1, P2 = qe.nnash(A, B1, B2, R1, R2, Q1,
                          Q2, S1, S2, W1, W2, M1,
                          M2, beta=β)

# Display policies
print("Computed policies for firm 1 and firm 2:\n")
print(f"F1 = {F1}")
print(f"F2 = {F2}")
print("\n")
```

```
Computed policies for firm 1 and firm 2:

F1 = [[-0.66846615  0.29512482  0.07584666]]
F2 = [[-0.66846615  0.07584666  0.29512482]]
```

Markov Perfect Equilibrium with Robustness

We add robustness concerns to the Markov Perfect Equilibrium model by extending the function `qe.nnash` (link) into a robustness version by adding the maximization operator $\mathcal{D}(P)$ into the backward induction.

The MPE with robustness function is `nnash_robust`.

The function's code is as follows

```python
def nnash_robust(A, C, B1, B2, R1, R2, Q1, Q2, S1, S2, W1, W2, M1, M2,
                 θ1, θ2, beta=1.0, tol=1e-8, max_iter=1000):

    """
    Compute the limit of a Nash linear quadratic dynamic game with
    robustness concern.

    In this problem, player i minimizes
    .. math::
        \sum_{t=0}^{\infty}
```

(continues on next page)

(continued from previous page)

```
        \left\{
            x_t' r_i x_t + 2 x_t' w_i
            u_{it} +u_{it}' q_i u_{it} + u_{jt}' s_i u_{jt} + 2 u_{jt}'
            m_i u_{it}
        \right\}
    subject to the law of motion
    .. math::
        x_{it+1} = A x_t + b_1 u_{1t} + b_2 u_{2t} + C w_{it+1}
    and a perceived control law :math:`u_j(t) = - f_j x_t` for the other
    player.

    The player i also concerns about the model misspecification,
    and maximizes
    .. math::
        \sum_{t=0}^{\infty}
        \left\{
            \beta^{t+1} \theta_{i} w_{it+1}'w_{it+1}
        \right\}

    The solution computed in this routine is the :math:`f_i` and
    :math:`P_i` of the associated double optimal linear regulator
    problem.

    Parameters
    ----------
    A : scalar(float) or array_like(float)
        Corresponds to the MPE equations, should be of size (n, n)
    C : scalar(float) or array_like(float)
        As above, size (n, c), c is the size of w
    B1 : scalar(float) or array_like(float)
        As above, size (n, k_1)
    B2 : scalar(float) or array_like(float)
        As above, size (n, k_2)
    R1 : scalar(float) or array_like(float)
        As above, size (n, n)
    R2 : scalar(float) or array_like(float)
        As above, size (n, n)
    Q1 : scalar(float) or array_like(float)
        As above, size (k_1, k_1)
    Q2 : scalar(float) or array_like(float)
        As above, size (k_2, k_2)
    S1 : scalar(float) or array_like(float)
        As above, size (k_1, k_1)
    S2 : scalar(float) or array_like(float)
        As above, size (k_2, k_2)
    W1 : scalar(float) or array_like(float)
        As above, size (n, k_1)
    W2 : scalar(float) or array_like(float)
        As above, size (n, k_2)
    M1 : scalar(float) or array_like(float)
        As above, size (k_2, k_1)
    M2 : scalar(float) or array_like(float)
        As above, size (k_1, k_2)
    θ1 : scalar(float)
            Robustness parameter of player 1
    θ2 : scalar(float)
```

(continues on next page)

(continued from previous page)

```
            Robustness parameter of player 2
    beta : scalar(float), optional(default=1.0)
          Discount factor
    tol : scalar(float), optional(default=1e-8)
         This is the tolerance level for convergence
    max_iter : scalar(int), optional(default=1000)
        This is the maximum number of iterations allowed

    Returns
    -------
    F1 : array_like, dtype=float, shape=(k_1, n)
        Feedback law for agent 1
    F2 : array_like, dtype=float, shape=(k_2, n)
        Feedback law for agent 2
    P1 : array_like, dtype=float, shape=(n, n)
        The steady-state solution to the associated discrete matrix
        Riccati equation for agent 1
    P2 : array_like, dtype=float, shape=(n, n)
        The steady-state solution to the associated discrete matrix
        Riccati equation for agent 2
    """

    # Unload parameters and make sure everything is a matrix
    params = A, C, B1, B2, R1, R2, Q1, Q2, S1, S2, W1, W2, M1, M2
    params = map(np.asmatrix, params)
    A, C, B1, B2, R1, R2, Q1, Q2, S1, S2, W1, W2, M1, M2 = params


    # Multiply A, B1, B2 by sqrt(β) to enforce discounting
    A, B1, B2 = [np.sqrt(β) * x for x in (A, B1, B2)]

    # Initial values
    n = A.shape[0]
    k_1 = B1.shape[1]
    k_2 = B2.shape[1]

    v1 = np.eye(k_1)
    v2 = np.eye(k_2)
    P1 = np.eye(n) * 1e-5
    P2 = np.eye(n) * 1e-5
    F1 = np.random.randn(k_1, n)
    F2 = np.random.randn(k_2, n)


    for it in range(max_iter):
        # Update
        F10 = F1
        F20 = F2

        I = np.eye(C.shape[1])

        # D1(P1)
        # Note: INV1 may not be solved if the matrix is singular
        INV1 = solve(θ1 * I - C.T @ P1 @ C, I)
        D1P1 =  P1 + P1 @ C @ INV1 @ C.T @ P1
```

(continues on next page)

(continued from previous page)

```
        # D2(P2)
        # Note: INV2 may not be solved if the matrix is singular
        INV2 = solve(θ2 * I - C.T @ P2 @ C, I)
        D2P2 =  P2 + P2 @ C @ INV2 @ C.T @ P2

        G2 = solve(Q2 + B2.T @ D2P2 @ B2, v2)
        G1 = solve(Q1 + B1.T @ D1P1 @ B1, v1)
        H2 = G2 @ B2.T @ D2P2
        H1 = G1 @ B1.T @ D1P1

        # Break up the computation of F1, F2
        F1_left = v1 - (H1 @ B2 + G1 @ M1.T) @ (H2 @ B1 + G2 @ M2.T)
        F1_right = H1 @ A + G1 @ W1.T - \
                   (H1 @ B2 + G1 @ M1.T) @ (H2 @ A + G2 @ W2.T)
        F1 = solve(F1_left, F1_right)
        F2 = H2 @ A + G2 @ W2.T - (H2 @ B1 + G2 @ M2.T) @ F1

        Λ1 = A - B2 @ F2
        Λ2 = A - B1 @ F1
        Π1 = R1 + F2.T @ S1 @ F2
        Π2 = R2 + F1.T @ S2 @ F1
        Γ1 = W1.T - M1.T @ F2
        Γ2 = W2.T - M2.T @ F1

        # Compute P1 and P2
        P1 = Π1 - (B1.T @ D1P1 @ Λ1 + Γ1).T @ F1 + \
             Λ1.T @ D1P1 @ Λ1
        P2 = Π2 - (B2.T @ D2P2 @ Λ2 + Γ2).T @ F2 + \
             Λ2.T @ D2P2 @ Λ2

        dd = np.max(np.abs(F10 - F1)) + np.max(np.abs(F20 - F2))

        if dd < tol:  # success!
            break

    else:
        raise ValueError(f'No convergence: Iteration limit of {max_iter} \
            reached in nnash')

    return F1, F2, P1, P2
```

21.3.3 Some Details

Firm i wants to minimize

$$\sum_{t=t_0}^{t_1-1} \beta^{t-t_0} \left\{ x_t' R_i x_t + u_{it}' Q_i u_{it} + u_{-it}' S_i u_{-it} + 2 x_t' W_i u_{it} + 2 u_{-it}' M_i u_{it} \right\}$$

where

$$x_t := \begin{bmatrix} 1 \\ q_{1t} \\ q_{2t} \end{bmatrix} \quad \text{and} \quad u_{it} := q_{i,t+1} - q_{it}, \quad i = 1, 2$$

and

$$R_1 := \begin{bmatrix} 0 & -\frac{a_0}{2} & 0 \\ -\frac{a_0}{2} & a_1 & \frac{a_1}{2} \\ 0 & \frac{a_1}{2} & 0 \end{bmatrix}, \quad R_2 := \begin{bmatrix} 0 & 0 & -\frac{a_0}{2} \\ 0 & 0 & \frac{a_1}{2} \\ -\frac{a_0}{2} & \frac{a_1}{2} & a_1 \end{bmatrix}, \quad Q_1 = Q_2 = \gamma, \quad S_1 = S_2 = 0, \quad W_1 = W_2 = 0, \quad M_1 = M_2 = 0$$

The parameters of the duopoly model are:

- $a_0 = 10$
- $a_1 = 2$
- $\beta = 0.96$
- $\gamma = 12$

```
# Parameters
a0 = 10.0
a1 = 2.0
β = 0.96
γ = 12.0


# In LQ form
A = np.eye(3)
B1 = np.array([[0.], [1.], [0.]])
B2 = np.array([[0.], [0.], [1.]])


R1 = [[      0.,     -a0 / 2,          0.],
      [-a0 / 2.,          a1,     a1 / 2.],
      [       0,     a1 / 2.,          0.]]

R2 = [[     0.,           0.,      -a0 / 2],
      [     0.,           0.,      a1 / 2.],
      [-a0 / 2,      a1 / 2.,           a1]]

Q1 = Q2 = γ
S1 = S2 = W1 = W2 = M1 = M2 = 0.0
```

Consistency Check

We first conduct a comparison test to check if `nnash_robust` agrees with `qe.nnash` in the non-robustness case in which each $\theta_i \approx +\infty$

```
# Solve using QE's nnash function
F1, F2, P1, P2 = qe.nnash(A, B1, B2, R1, R2, Q1,
                          Q2, S1, S2, W1, W2, M1,
                          M2, beta=β)

# Solve using nnash_robust
F1r, F2r, P1r, P2r = nnash_robust(A, np.zeros((3, 1)), B1, B2, R1, R2, Q1,
                                  Q2, S1, S2, W1, W2, M1, M2, 1e-10,
                                  1e-10, beta=β)

print('F1 and F1r should be the same: ', np.allclose(F1, F1r))
print('F2 and F2r should be the same: ', np.allclose(F1, F1r))
```

(continues on next page)

(continued from previous page)

```
print('P1 and P1r should be the same: ', np.allclose(P1, P1r))
print('P2 and P2r should be the same: ', np.allclose(P1, P1r))
```

```
F1 and F1r should be the same:  True
F2 and F2r should be the same:  True
P1 and P1r should be the same:  True
P2 and P2r should be the same:  True
```

We can see that the results are consistent across the two functions.

Comparative Dynamics under Baseline Transition Dynamics

We want to compare the dynamics of price and output under the baseline MPE model with those under the baseline model under the robust decision rules within the robust MPE.

This means that we simulate the state dynamics under the MPE equilibrium **closed-loop** transition matrix

$$A^o = A - B_1 F_1 - B_2 F_2$$

where F_1 and F_2 are the firms' robust decision rules within the robust markov_perfect equilibrium

- by simulating under the baseline model transition dynamics and the robust MPE rules we are in assuming that at the end of the day firms' concerns about misspecification of the baseline model do not materialize.
- a short way of saying this is that misspecification fears are all 'just in the minds' of the firms.
- simulating under the baseline model is a common practice in the literature.
- note that *some* assumption about the model that actually governs the data has to be made in order to create a simulation.
- later we will describe the (erroneous) beliefs of the two firms that justify their robust decisions as best responses to transition laws that are distorted relative to the baseline model.

After simulating x_t under the baseline transition dynamics and robust decision rules $F_i, i = 1, 2$, we extract and plot industry output $q_t = q_{1t} + q_{2t}$ and price $p_t = a_0 - a_1 q_t$.

Here we set the robustness and volatility matrix parameters as follows:

- $\theta_1 = 0.02$
- $\theta_2 = 0.04$
- $C = \begin{pmatrix} 0 \\ 0.01 \\ 0.01 \end{pmatrix}$

Because we have set $\theta_1 < \theta_2 < +\infty$ we know that

- both firms fear that the baseline specification of the state transition dynamics are incorrect.
- firm 1 fears misspecification more than firm 2.

```
# Robustness parameters and matrix
C = np.asmatrix([[0], [0.01], [0.01]])
θ1 = 0.02
θ2 = 0.04
n = 20
```

(continues on next page)

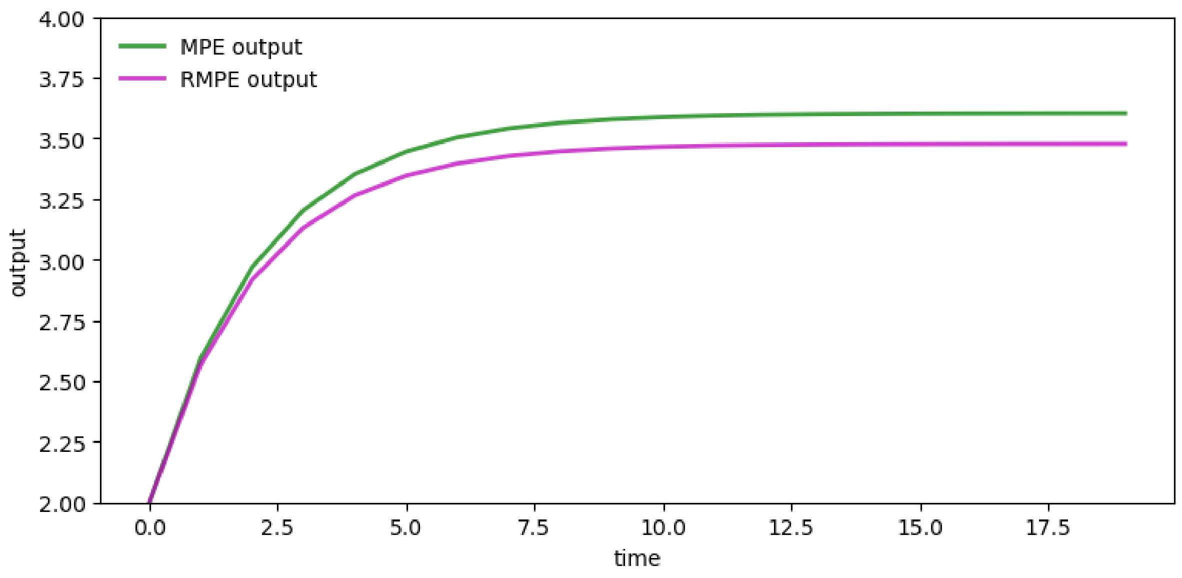
(continued from previous page)

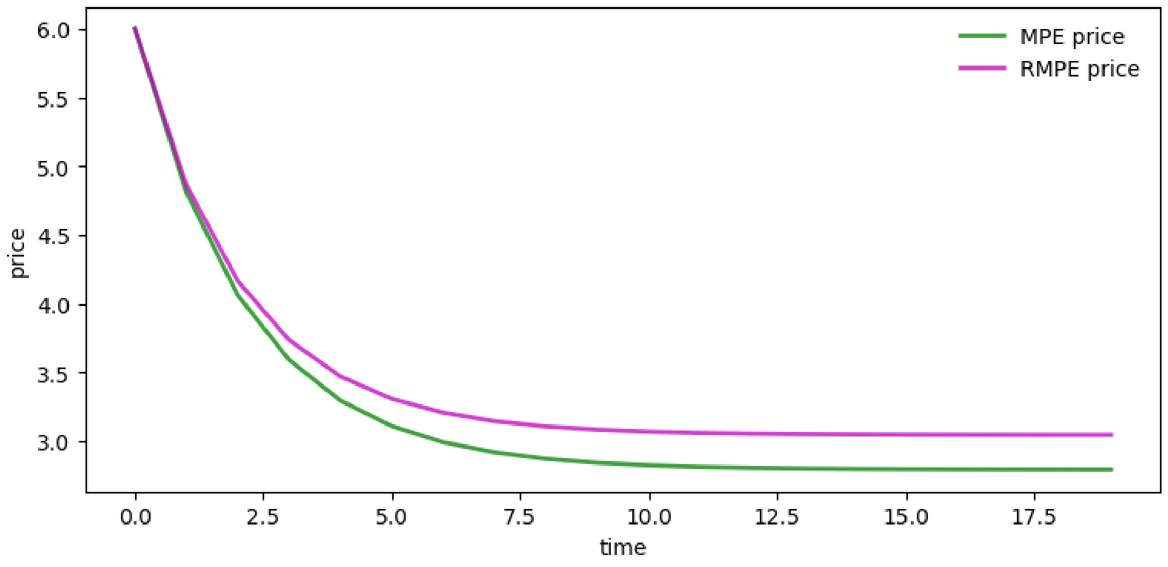
```
# Solve using nnash_robust
F1r, F2r, P1r, P2r = nnash_robust(A, C, B1, B2, R1, R2, Q1,
                                  Q2, S1, S2, W1, W2, M1, M2,
                                  θ1, θ2, beta=β)



# MPE output and price
AF = A - B1 @ F1 - B2 @ F2
x = np.empty((3, n))
x[:, 0] = 1, 1, 1
for t in range(n - 1):
    x[:, t + 1] = AF @ x[:, t]
q1 = x[1, :]
q2 = x[2, :]
q = q1 + q2       # Total output, MPE
p = a0 - a1 * q   # Price, MPE


# RMPE output and price
AO = A - B1 @ F1r - B2 @ F2r
xr = np.empty((3, n))
xr[:, 0] = 1, 1, 1
for t in range(n - 1):
    xr[:, t+1] = AO @ xr[:, t]
qr1 = xr[1, :]
qr2 = xr[2, :]
qr = qr1 + qr2      # Total output, RMPE
pr = a0 - a1 * qr   # Price, RMPE

# RMPE heterogeneous beliefs output and price
I = np.eye(C.shape[1])
INV1 = solve(θ1 * I - C.T @ P1 @ C, I)
K1 =  P1 @ C @ INV1 @ C.T @ P1 @ AO
AOCK1 = AO + C.T @ K1

INV2 = solve(θ2 * I - C.T @ P2 @ C, I)
K2 =  P2 @ C @ INV2 @ C.T @ P2 @ AO
AOCK2 = AO + C.T @ K2
xrp1 = np.empty((3, n))
xrp2 = np.empty((3, n))
xrp1[:, 0] = 1, 1, 1
xrp2[:, 0] = 1, 1, 1
for t in range(n - 1):
    xrp1[:, t + 1] = AOCK1 @ xrp1[:, t]
    xrp2[:, t + 1] = AOCK2 @ xrp2[:, t]
qrp11 = xrp1[1, :]
qrp12 = xrp1[2, :]
qrp21 = xrp2[1, :]
qrp22 = xrp2[2, :]
qrp1 = qrp11 + qrp12       # Total output, RMPE from player 1's belief
qrp2 = qrp21 + qrp22       # Total output, RMPE from player 2's belief
prp1 = a0 - a1 * qrp1      # Price, RMPE from player 1's belief
prp2 = a0 - a1 * qrp2      # Price, RMPE from player 2's belief
```

The following code prepares graphs that compare market-wide output $q_{1t} + q_{2t}$ and the price of the good p_t under equilibrium decision rules $F_i, i = 1, 2$ from an ordinary Markov perfect equilibrium and the decision rules under a Markov perfect equilibrium with robust firms with multiplier parameters $\theta_i, i = 1, 2$ set as described above.

Both industry output and price are under the transition dynamics associated with the baseline model; only the decision rules F_i differ across the two equilibrium objects presented.

```
fig, axes = plt.subplots(2, 1, figsize=(9, 9))

ax = axes[0]
ax.plot(q, 'g-', lw=2, alpha=0.75, label='MPE output')
ax.plot(qr, 'm-', lw=2, alpha=0.75, label='RMPE output')
ax.set(ylabel="output", xlabel="time", ylim=(2, 4))
ax.legend(loc='upper left', frameon=0)

ax = axes[1]
ax.plot(p, 'g-', lw=2, alpha=0.75, label='MPE price')
ax.plot(pr, 'm-', lw=2, alpha=0.75, label='RMPE price')
ax.set(ylabel="price", xlabel="time")
ax.legend(loc='upper right', frameon=0)
plt.show()
```





Under the dynamics associated with the baseline model, the price path is higher with the Markov perfect equilibrium robust decision rules than it is with decision rules for the ordinary Markov perfect equilibrium.

So is the industry output path.

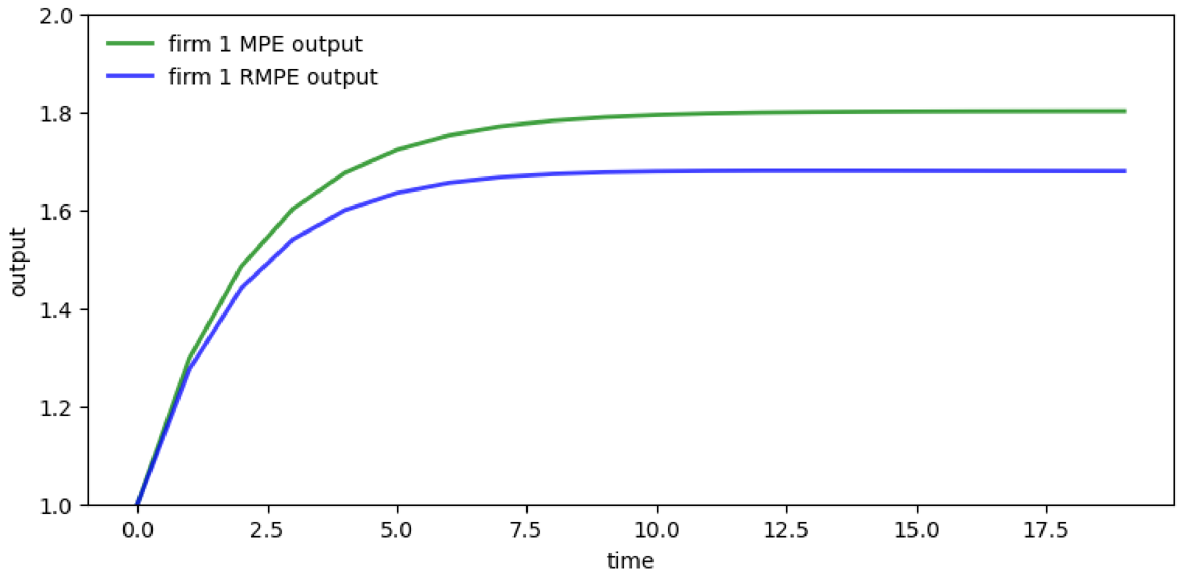
To dig a little beneath the forces driving these outcomes, we want to plot q_{1t} and q_{2t} in the Markov perfect equilibrium with robust firms and to compare them with corresponding objects in the Markov perfect equilibrium without robust firms

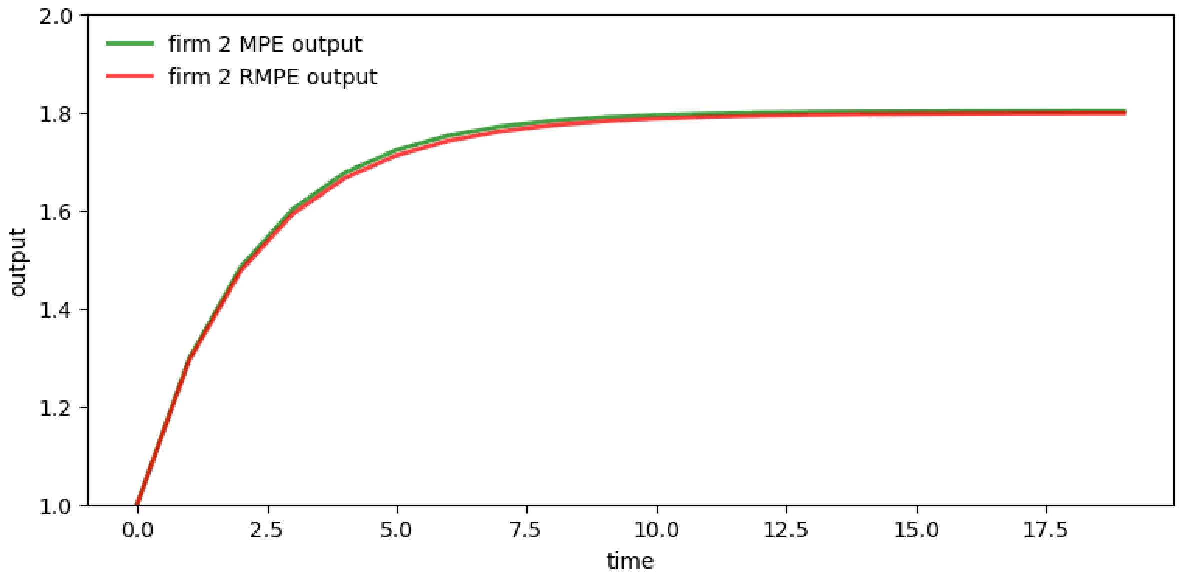
```
fig, axes = plt.subplots(2, 1, figsize=(9, 9))

ax = axes[0]
ax.plot(q1, 'g-', lw=2, alpha=0.75, label='firm 1 MPE output')
ax.plot(qr1, 'b-', lw=2, alpha=0.75, label='firm 1 RMPE output')
ax.set(ylabel="output", xlabel="time", ylim=(1, 2))
ax.legend(loc='upper left', frameon=0)

ax = axes[1]
```

(continues on next page)

(continued from previous page)

```
ax.plot(q2, 'g-', lw=2, alpha=0.75, label='firm 2 MPE output')
ax.plot(qr2, 'r-', lw=2, alpha=0.75, label='firm 2 RMPE output')
ax.set(ylabel="output", xlabel="time", ylim=(1, 2))
ax.legend(loc='upper left', frameon=0)
plt.show()
```





Evidently, firm 1's output path is substantially lower when firms are robust firms while firm 2's output path is virtually the same as it would be in an ordinary Markov perfect equilibrium with no robust firms.

Recall that we have set $\theta_1 = .02$ and $\theta_2 = .04$, so that firm 1 fears misspecification of the baseline model substantially more than does firm 2

- but also please notice that firm 2's behavior in the Markov perfect equilibrium with robust firms responds to the decision rule $F_1 x_t$ employed by firm 1.
- thus it is something of a coincidence that its output is almost the same in the two equilibria.

Larger concerns about misspecification induce firm 1 to be more cautious than firm 2 in predicting market price and the output of the other firm.

To explore this, we study next how *ex-post* the two firms' beliefs about state dynamics differ in the Markov perfect equilibrium with robust firms.

(by *ex-post* we mean *after* extremization of each firm's intertemporal objective)

Heterogeneous Beliefs

As before, let $A^o = A - B\_1F\_1^r - B\_2F\_2^r$, where in a robust MPE, F_i^r is a robust decision rule for firm i.

Worst-case forecasts of x_t starting from $t = 0$ differ between the two firms.

This means that worst-case forecasts of industry output $q_{1t} + q_{2t}$ and price p_t also differ between the two firms.

To find these worst-case beliefs, we compute the following three "closed-loop" transition matrices

- A^o
- $A^o + CK\_1$
- $A^o + CK\_2$

We call the first transition law, namely, A^o, the baseline transition under firms' robust decision rules.

We call the second and third worst-case transitions under robust decision rules for firms 1 and 2.

From $\{x_t\}$ paths generated by each of these transition laws, we pull off the associated price and total output sequences.

The following code plots them

```
print('Baseline Robust transition matrix AO is: \n', np.round(AO, 3))
print('Player 1\'s worst-case transition matrix AOCK1 is: \n', \
np.round(AOCK1, 3))
print('Player 2\'s worst-case transition matrix AOCK2 is: \n', \
np.round(AOCK2, 3))
```

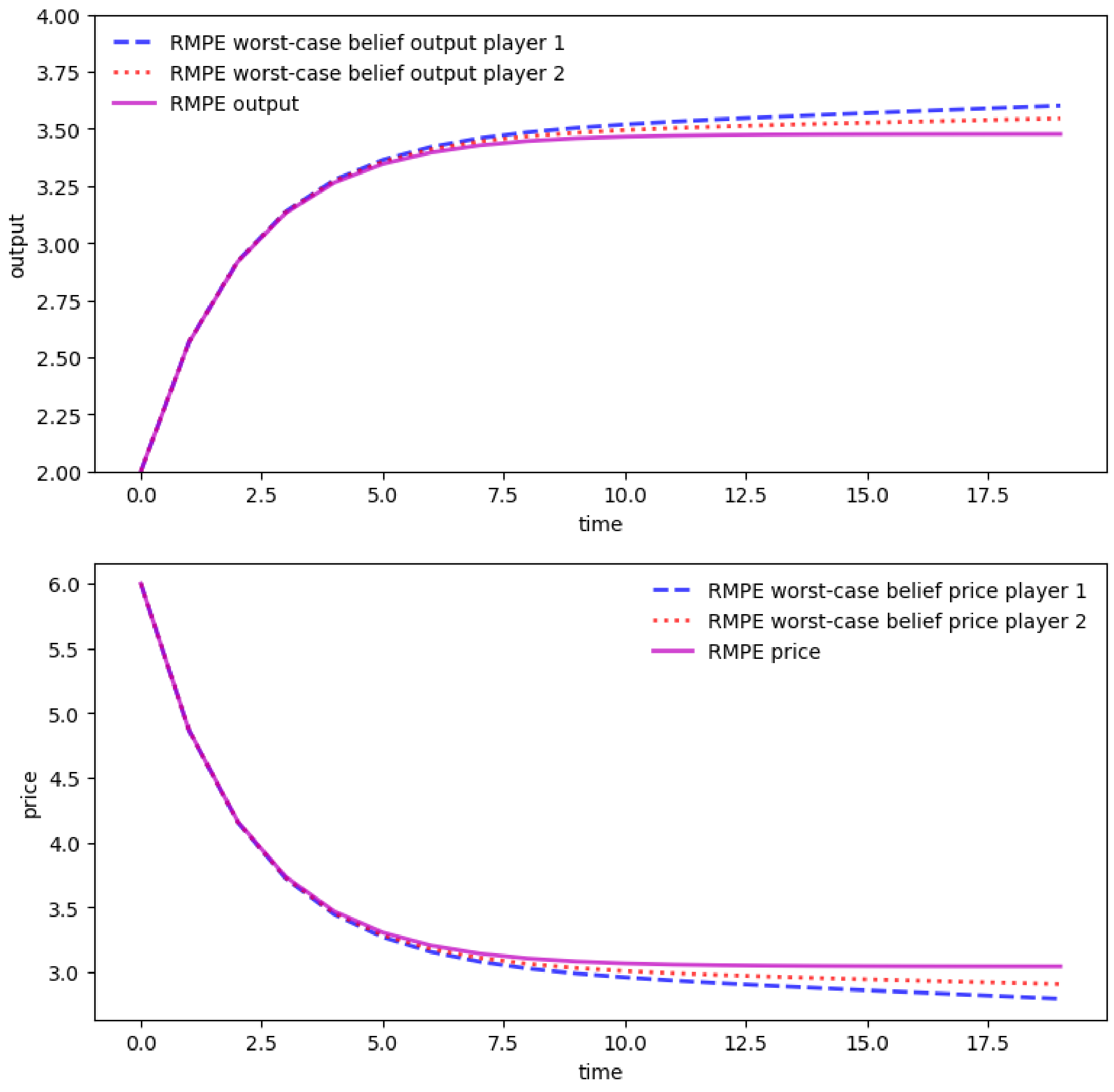
```
Baseline Robust transition matrix AO is:
 [[ 1.     0.     0.   ]
 [ 0.666  0.682 -0.074]
 [ 0.671 -0.071  0.694]]
Player 1's worst-case transition matrix AOCK1 is:
 [[ 0.998  0.002  0.   ]
 [ 0.664  0.685 -0.074]
 [ 0.669 -0.069  0.694]]
Player 2's worst-case transition matrix AOCK2 is:
 [[ 0.999  0.     0.001]
 [ 0.665  0.683 -0.073]
 [ 0.67  -0.071  0.695]]
```

```
# == Plot == #
fig, axes = plt.subplots(2, 1, figsize=(9, 9))

ax = axes[0]
ax.plot(qrp1, 'b--', lw=2, alpha=0.75,
    label='RMPE worst-case belief output player 1')
ax.plot(qrp2, 'r:', lw=2, alpha=0.75,
    label='RMPE worst-case belief output player 2')
```

(continues on next page)

(continued from previous page)

```
ax.plot(qr, 'm-', lw=2, alpha=0.75, label='RMPE output')
ax.set(ylabel="output", xlabel="time", ylim=(2, 4))
ax.legend(loc='upper left', frameon=0)

ax = axes[1]
ax.plot(prp1, 'b--', lw=2, alpha=0.75,
    label='RMPE worst-case belief price player 1')
ax.plot(prp2, 'r:', lw=2, alpha=0.75,
    label='RMPE worst-case belief price player 2')
ax.plot(pr, 'm-', lw=2, alpha=0.75, label='RMPE price')
ax.set(ylabel="price", xlabel="time")
ax.legend(loc='upper right', frameon=0)
plt.show()
```



We see from the above graph that under robustness concerns, player 1 and player 2 have heterogeneous beliefs about total output and the goods price even though they share the same baseline model and information

- firm 1 thinks that total output will be higher and price lower than does firm 2
- this leads firm 1 to produce less than firm 2

These beliefs justify (or **rationalize**) the Markov perfect equilibrium robust decision rules.

This means that the robust rules are the unique **optimal** rules (or best responses) to the indicated worst-case transition dynamics.

([Hansen and Sargent, 2008] discuss how this property of robust decision rules is connected to the concept of *admissibility* in Bayesian statistical decision theory)

Part VI

Other

CHAPTER

TWENTYTWO

TROUBLESHOOTING

This page is for readers experiencing errors when running the code from the lectures.

22.1 Fixing Your Local Environment

The basic assumption of the lectures is that code in a lecture should execute whenever

1. it is executed in a Jupyter notebook and
2. the notebook is running on a machine with the latest version of Anaconda Python.

You have installed Anaconda, haven't you, following the instructions in this lecture?

Assuming that you have, the most common source of problems for our readers is that their Anaconda distribution is not up to date.

Here's a useful article on how to update Anaconda.

Another option is to simply remove Anaconda and reinstall.

You also need to keep the external code libraries, such as QuantEcon.py up to date.

For this task you can either

- use conda install -y quantecon on the command line, or
- execute !conda install -y quantecon within a Jupyter notebook.

If your local environment is still not working you can do two things.

First, you can use a remote machine instead, by clicking on the Launch Notebook icon available for each lecture

Second, you can report an issue, so we can try to fix your local set up.

We like getting feedback on the lectures so please don't hesitate to get in touch.

22.2 Reporting an Issue

One way to give feedback is to raise an issue through our issue tracker.

Please be as specific as possible. Tell us where the problem is and as much detail about your local set up as you can provide.

Another feedback option is to use our discourse forum.

Finally, you can provide direct feedback to contact@quantecon.org

CHAPTER

TWENTYTHREE

REFERENCES

CHAPTER

TWENTYFOUR

EXECUTION STATISTICS

This table contains the latest execution statistics.

Document	Modified	Method	Run Time (s)	Status
additive_functionals	2024-06-13 03:26	cache	18.63	✅
arma	2024-06-13 03:26	cache	8.31	✅
classical_filtering	2024-06-13 03:26	cache	1.28	✅
discrete_dp	2024-06-13 03:27	cache	30.75	✅
eig_circulant	2024-06-13 03:27	cache	3.39	✅
entropy	2024-06-13 03:27	cache	0.99	✅
estspec	2024-06-13 03:27	cache	5.88	✅
finite_markov	2024-06-13 03:27	cache	6.52	✅
five_preferences	2024-06-13 03:27	cache	14.95	✅
intro	2024-06-13 03:27	cache	1.02	✅
linear_algebra	2024-06-13 03:27	cache	2.01	✅
lp_intro	2024-06-13 03:28	cache	1.52	✅
lu_tricks	2024-06-13 03:28	cache	2.06	✅
newton_method	2024-06-13 03:29	cache	66.12	✅
opt_transport	2024-06-13 03:29	cache	16.31	✅
qr_decomp	2024-06-13 03:29	cache	1.04	✅
rob_markov_perf	2024-06-13 03:29	cache	5.44	✅
robustness	2024-06-13 03:29	cache	6.48	✅
stationary_densities	2024-06-13 03:29	cache	10.0	✅
status	2024-06-13 03:29	cache	4.44	✅
svd_intro	2024-06-13 03:29	cache	1.4	✅
troubleshooting	2024-06-13 03:27	cache	1.02	✅
var_dmd	2024-06-13 03:27	cache	1.02	✅
von_neumann_model	2024-06-13 03:29	cache	2.02	✅
zreferences	2024-06-13 03:27	cache	1.02	✅

These lectures are built on `linux` instances through `github actions`.

These lectures are using the following python version

```
!python --version
```

```
Python 3.11.7
```

and the following package versions

```
!conda list
```

BIBLIOGRAPHY

[AHS03] Evan W. Anderson, Lars Peter Hansen, and Thomas J. Sargent. A Quartet of Semigroups for Model Specification, Robustness, Prices of Risk, and Model Detection. *Journal of the European Economic Association*, 1(1):68–123, March 2003. URL: https://ideas.repec.org/a/tpr/jeurec/v1y2003i1p68-123.html, doi:.

[AP91] Papoulis Athanasios and S Unnikrishna Pillai. *Probability, random variables, and stochastic processes*. McGraw Hill, 1991.

[BCZ14] David Backus, Mikhail Chernov, and Stanley Zin. Sources of Entropy in Representative Agent Models. *Journal of Finance*, 69(1):51–99, February 2014. URL: https://ideas.repec.org/a/bla/jfinan/v69y2014i1p51-99.html, doi:.

[BHS09] Francisco Barillas, Lars Peter Hansen, and Thomas J. Sargent. Doubts or variability? *Journal of Economic Theory*, 144(6):2388–2418, November 2009. URL: https://ideas.repec.org/a/eee/jetheo/v144y2009i6p2388-2418.html, doi:.

[Ber97] J. N. Bertsimas, D. & Tsitsiklis. *Introduction to linear optimization*. Athena Scientific, 1997.

[BK19] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

[BK22] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering, Second Edition*. Cambridge University Press, New York, 2022.

[Buc04] James A. Bucklew. *An Introduction to Rare Event Simulation*. Springer Verlag, New York, 2004.

[CC08] J. D. Cryer and K-S. Chan. *Time Series Analysis*. Springer, 2nd edition edition, 2008.

[DSS58] Robert Dorfman, Paul A. Samuelson, and Robert M. Solow. *Linear Programming and Economic Analysis: Revised Edition*. McGraw Hill, New York, 1958.

[DLP13] Y E Du, Ehud Lehrer, and A D Y Pauzner. Competitive economy as a ranking device over networks. submitted, 2013.

[Gal89] David Gale. *The theory of linear economic models*. University of Chicago press, 1989.

[Gal16] Alfred Galichon. *Optimal Transport Methods in Economics*. Princeton University Press, Princeton, New Jersey, 2016.

[GS89] Itzhak Gilboa and David Schmeidler. Maxmin Expected Utility with Non-Unique Prior. *Journal of Mathematical Economics*, 18(2):141–153, apr 1989.

[HTW67] Michael J Hamburger, Gerald L Thompson, and Roman L Weil. Computation of expansion rates for the generalized von neumann model of an expanding economy. *Econometrica*, pages 542–547, 1967.

[Ham05] James D Hamilton. What's real about the business cycle? *Federal Reserve Bank of St. Louis Review*, pages 435–452, 2005.

[HS08a] L P Hansen and T J Sargent. *Robustness*. Princeton University Press, 2008.

[Han12] Lars Peter Hansen. Dynamic Valuation Decomposition Within Stochastic Economies. *Econometrica*, 80(3):911–967, May 2012. URL: https://ideas.repec.org/a/ecm/emetrp/v80y2012i3p911-967.html, doi:10.3982/ECTA8070.

[HJ91] Lars Peter Hansen and Ravi Jagannathan. Implications of Security Market Data for Models of Dynamic Economies. *Journal of Political Economy*, 99(2):225–262, April 1991. URL: https://ideas.repec.org/a/ucp/jpolec/v99y1991i2p225-62.html, doi:10.1086/261749.

[HS80] Lars Peter Hansen and Thomas J Sargent. Formulating and estimating dynamic linear rational expectations models. *Journal of Economic Dynamics and control*, 2:7–46, 1980.

[HS00] Lars Peter Hansen and Thomas J Sargent. Wanting robustness in macroeconomics. *Manuscript, Department of Economics, Stanford University.*, 2000.

[HS08b] Lars Peter Hansen and Thomas J Sargent. *Robustness*. Princeton University Press, 2008.

[HS01] Lars Peter Hansen and Thomas J. Sargent. Robust control and model uncertainty. *American Economic Review*, 91(2):60–66, 2001.

[HS24] Lars Peter Hansen and Thomas J. Sargent. Risk, uncertainty, and value. University of Chicago and NYU manuscript, 2024.

[HST99] Lars Peter Hansen, Thomas J. Sargent, and Thomas D. Tallarini. Robust Permanent Income and Pricing. *Review of Economic Studies*, 66(4):873–907, 1999. URL: https://ideas.repec.org/a/oup/restud/v66y1999i4p873-907..html, doi:.

[HLL96] O Hernandez-Lerma and J B Lasserre. *Discrete-Time Markov Control Processes: Basic Optimality Criteria*. Number Vol 1 in Applications of Mathematics Stochastic Modelling and Applied Probability. Springer, 1996.

[HR93] Hugo A Hopenhayn and Richard Rogerson. Job Turnover and Policy Evaluation: A General Equilibrium Analysis. *Journal of Political Economy*, 101(5):915–938, 1993.

[Hu18] Y. Hu, Y. & Guo. *Operations research*. Tsinghua University Press, 5th edition, 2018.

[Haggstrom02] Olle Häggström. *Finite Markov chains and algorithmic applications*. Volume 52. Cambridge University Press, 2002.

[Janich94] K Jänich. *Linear Algebra*. Springer Undergraduate Texts in Mathematics and Technology. Springer, 1994.

[Jac73] D. H. Jacobson. Optimal stochastic linear systems with exponential performance criteria and their relation to differential games. *IEEE Transactions on Automatic Control*, 18(2):124–131, 1973.

[KMT56] John G Kemeny, Oskar Morgenstern, and Gerald L Thompson. A generalization of the von neumann model of an expanding economy. *Econometrica*, pages 115–135, 1956.

[Kni21] Frank H. Knight. *Risk, Uncertainty, and Profit*. Houghton Mifflin, 1921.

[KBBWP16] J. N. Kutz, S. L. Brunton, Brunton B. W, and J. L. Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

[LM94] A Lasota and M C MacKey. *Chaos, Fractals, and Noise: Stochastic Aspects of Dynamics*. Applied Mathematical Sciences. Springer-Verlag, 1994.

[LS18] L Ljungqvist and T J Sargent. *Recursive Macroeconomic Theory*. MIT Press, 4 edition, 2018.

[Luc87] Robert E Lucas. *Models of business cycles*. Volume 26. Oxford Blackwell, 1987.

[MMR06] Fabio Maccheroni, Massimo Marinacci, and Aldo Rustichini. Ambiguity Aversion, Robustness, and the Variational Representation of Preferences. *Econometrica*, 74(6):1147–1498, 2006.

[MT09] S P Meyn and R L Tweedie. *Markov Chains and Stochastic Stability*. Cambridge University Press, 2009.

[MF02] Mario J Miranda and P L Fackler. *Applied Computational Economics and Finance*. Cambridge: MIT Press, 2002.

[Mut60] John F Muth. Optimal properties of exponentially weighted forecasts. *Journal of the american statistical association*, 55(290):299–306, 1960.

[Orf88] Sophocles J Orfanidis. *Optimum Signal Processing: An Introduction*. McGraw Hill Publishing, New York, New York, 1988.

[Put05] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley \\& Sons, 2005.

[Roz67] Y. A. Rozanov. *Stationary Random Processes*. Holden-Day, San Francisco, 1967.

[Rus96] John Rust. Numerical dynamic programming in economics. *Handbook of computational economics*, 1:619–729, 1996.

[Sar87] Thomas J Sargent. *Macroeconomic Theory*. Academic Press, New York, 2nd edition, 1987.

[Sch10] Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

[SW49] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, 1949.

[Shi95] A N Shiriaev. *Probability*. Graduate texts in mathematics. Springer. Springer, 2nd edition, 1995.

[SLP89] N L Stokey, R E Lucas, and E C Prescott. *Recursive Methods in Economic Dynamics*. Harvard University Press, 1989.

[Tal00] Thomas D Tallarini. Risk-sensitive real business cycles. *Journal of Monetary Economics*, 45(3):507–532, June 2000.

[Tau86] George Tauchen. Finite state markov-chain approximations to univariate and vector autoregressions. *Economics Letters*, 20(2):177–181, 1986.

[TRL+14] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz. On dynamic mode decomposition: theory and applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014.

[vN28] John von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320, 1928.

[vN37] John von Neumann. Uber ein okonomsiches gleichungssystem und eine verallgemeinering des browerschen fixpunktsatzes. In *Erge. Math. Kolloq.*, volume 8, 73–83. 1937.

[Whi63] Peter Whittle. *Prediction and regulation by linear least-square methods*. English Univ. Press, 1963.

[Whi81] Peter Whittle. Risk-sensitive linear/quadratic/gaussian control. *Advances in Applied Probability*, 13(4):764–777, 1981.

[Whi83] Peter Whittle. *Prediction and Regulation by Linear Least Squares Methods*. University of Minnesota Press, Minneapolis, Minnesota, 2nd edition, 1983.

[Whi90] Peter Whittle. *Risk-Sensitive Optimal Control*. Wiley, New York, 1990.

INDEX