Covariance Stationary Processes

Thomas J. Sargent and John Stachurski

June 23, 2021

1 Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

```
In [1]: !pip install --upgrade quantecon
```

2 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.

2.1 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.

2.2 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.

2.3 Other Reading

For supplementary reading, see

- [2], chapter 2
- [3], chapter 11
- John Cochrane's notes on time series analysis, chapter 8
- [4], chapter 6
- [1], all

Let's start with some imports:

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        import quantecon as qe
```

```
   /home/ubuntu/anaconda3/lib/python3.7/site-packages/numba/np/ufunc/parallel.py:
↪355:
NumbaWarning: The TBB threading layer requires TBB version 2019.5 or later i.e.,

TBB_INTERFACE_VERSION >= 11005. Found TBB_INTERFACE_VERSION = 11004. The TBB
↪threading

layer is disabled.
  warnings.warn(problem)
```

3 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 $X_t, X_{t+1}, ...$ 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).

3.1 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.

3.2 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.

3.3 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*

$$X_t = \sum_{j=0}^{\infty} \psi_j \epsilon_{t-j}, \qquad t \in \mathbb{Z} \tag{1}$$

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 (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 (1) is

$$\gamma(k) = \sigma^2 \sum_{j=0}^{\infty} \psi_j \psi_{j+k} \tag{2}$$

By the Cauchy-Schwartz inequality, one can show that $\gamma(k)$ satisfies equation (2).

Evidently, $\gamma(k)$ does not depend on t.

3.4 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 $X_{t-1}, X_{t-2}, \ldots$
- 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 [3], p. 286.

3.5 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

$$X_t = \phi X_{t-1} + \epsilon_t \quad \text{where} \quad |\phi| < 1 \quad \text{and } \{\epsilon_t\} \text{ is white noise} \tag{3}$$

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.

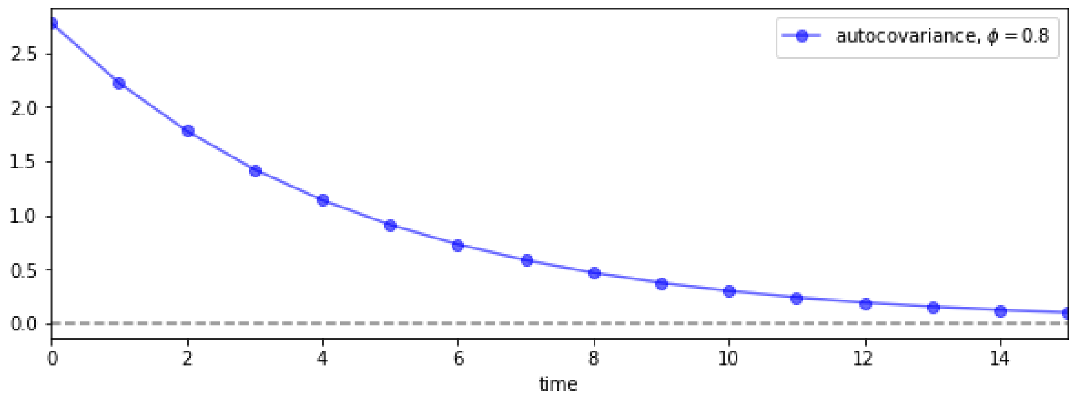
Applying (2) to the previous expression for X_t, we get the AR(1) autocovariance function

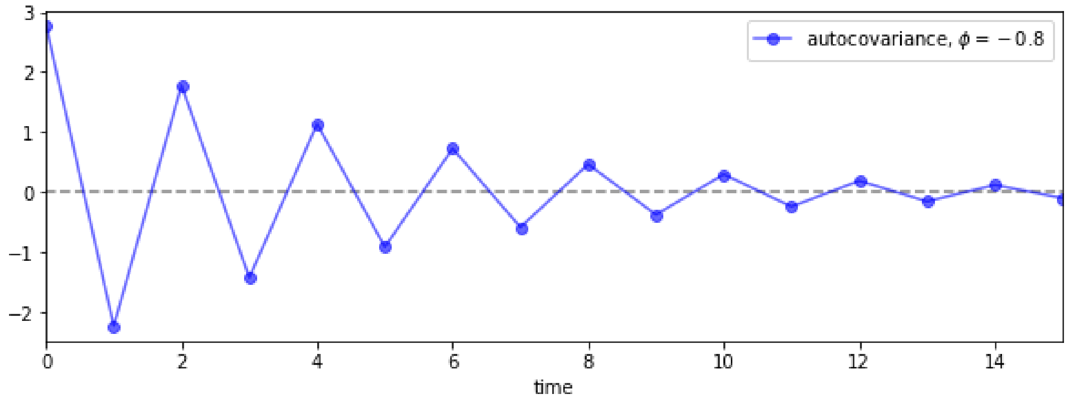
$$\gamma(k) = \phi^k \frac{\sigma^2}{1 - \phi^2}, \qquad k = 0, 1, \ldots \tag{4}$$

The next figure plots an example of this function for $\phi = 0.8$ and $\phi = -0.8$ with $\sigma = 1$.

```
In [3]: 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

3.6 ARMA Processes

A stochastic process $\{X_t\}$ is called an *autoregressive moving average process*, or ARMA(p, q), if it can be written as

$$X_t = \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \tag{5}$$

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 (5) as

$$L^0 X_t - \phi_1 L^1 X_t - \cdots - \phi_p L^p X_t = L^0 \epsilon_t + \theta_1 L^1 \epsilon_t + \cdots + \theta_q L^q \epsilon_t \tag{6}$$

If we let $\phi(z)$ and $\theta(z)$ be the polynomials

$$\phi(z) := 1 - \phi_1 z - \cdots - \phi_p z^p \quad \text{and} \quad \theta(z) := 1 + \theta_1 z + \cdots + \theta_q z^q \tag{7}$$

then (6) becomes

$$\phi(L) X_t = \theta(L) \epsilon_t \tag{8}$$

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 [1].

The function $t \mapsto \psi_t$ is often called the *impulse response function*.

4 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.

4.1 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 $\tan(\omega) = y/x$.

4.2 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

$$f(\omega) = \gamma(0) + 2 \sum_{k \geq 1} \gamma(k) \cos(\omega k) \tag{9}$$

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 $[0, \pi]$.

4.3 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")

4.4 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

$$f(\omega) = \sigma^2(1 + 2\theta\cos(\omega) + \theta^2) \tag{10}$$

With a bit more effort, it's possible to show (see, e.g., p. 261 of [3]) that the spectral density of the AR(1) process $X_t = \phi X_{t-1} + \epsilon_t$ is

$$f(\omega) = \frac{\sigma^2}{1 - 2\phi\cos(\omega) + \phi^2} \tag{11}$$

More generally, it can be shown that the spectral density of the ARMA process (5) is

$$f(\omega) = \left|\frac{\theta(e^{i\omega})}{\phi(e^{i\omega})}\right|^2 \sigma^2 \tag{12}$$

where

- σ is the standard deviation of the white noise process $\{\epsilon_t\}$.
- the polynomials $\phi(\cdot)$ and $\theta(\cdot)$ are as defined in (7).

The derivation of (12) uses the fact that convolutions become products under Fourier transformations.

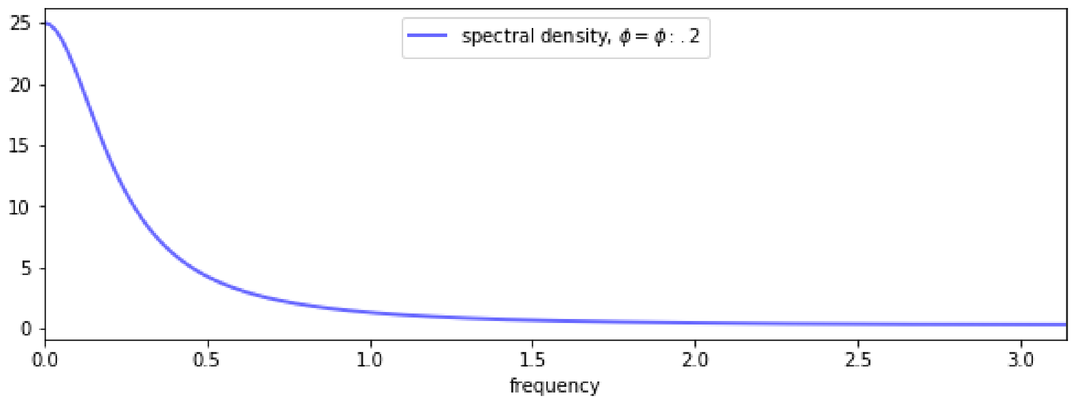
The proof is elegant and can be found in many places — see, for example, [3], chapter 11, section 4.

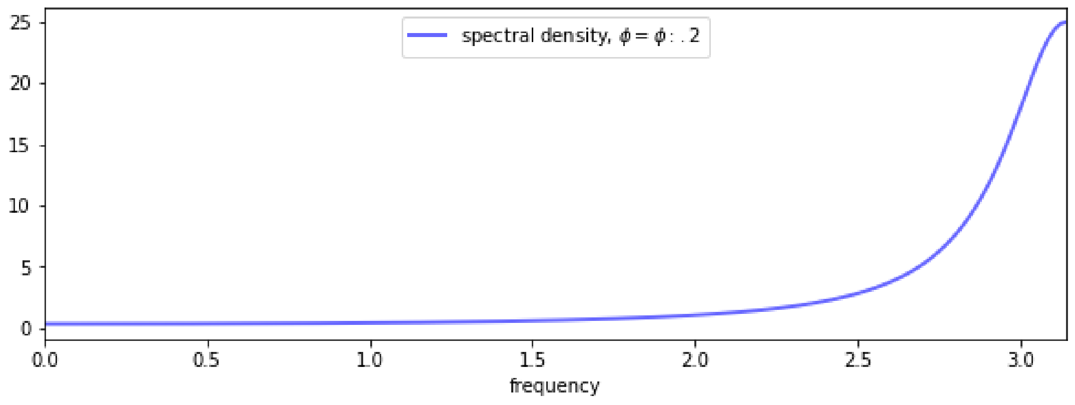
It's a nice exercise to verify that (10) and (11) are indeed special cases of (12).

4.5 Interpreting the Spectral Density

Plotting (11) reveals the shape of the spectral density for the AR(1) model when ϕ takes the values 0.8 and -0.8 respectively.

```
In [4]: 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

$$f(\omega) = \gamma(0) + 2 \sum_{k \geq 1} \gamma(k) \cos(\omega k) = \gamma(0) + 2 \sum_{k \geq 1} (-0.8)^k \cos(\omega k) \tag{13}$$

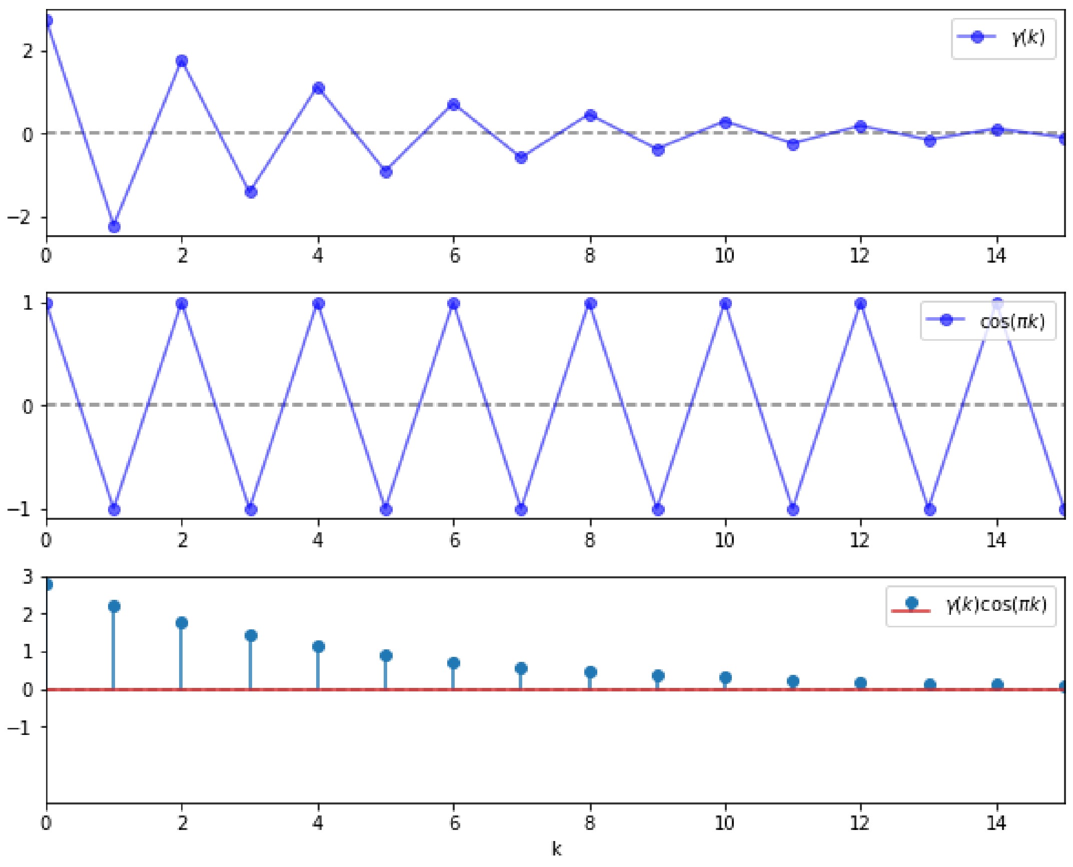
When we evaluate this at $\omega = \pi$, we get a large number because $\cos(\pi k)$ is large and positive when $(-0.8)^k$ is positive, and large in absolute value and negative when $(-0.8)^k$ is negative.

Hence the product is always large and positive, and hence the sum of the products on the right-hand side of (13) is large.

These ideas are illustrated in the next figure, which has k on the horizontal axis.

```
In [5]: 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]
        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 ☒ = -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()
```

```
   /home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:27:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
```

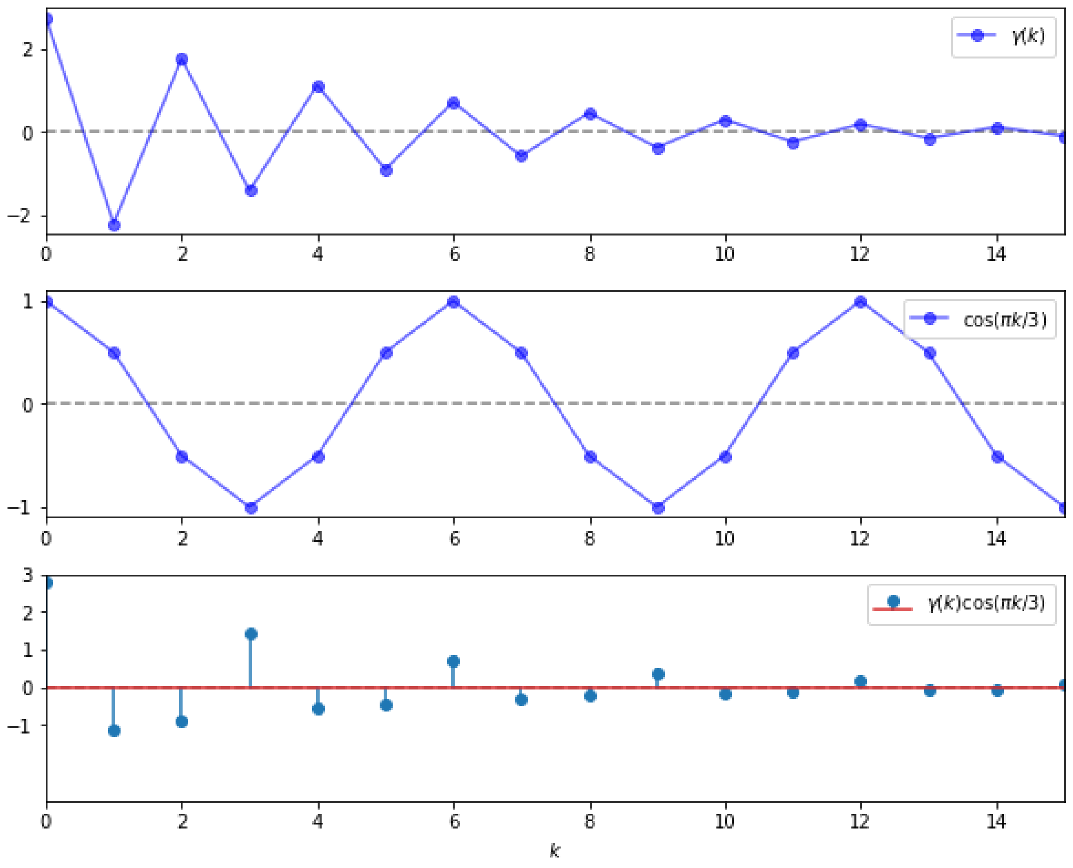


On the other hand, if we evaluate $f(\omega)$ at $\omega = \pi/3$, then the cycles are not matched, the sequence $\gamma(k)\cos(\omega k)$ contains both positive and negative terms, and hence the sum of these terms is much smaller.

In [6]:
```
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='$\cos(\pi k/3)$')
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()
```

```
    /home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:27:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
```



In summary, the spectral density is large at frequencies ω where the autocovariance function exhibits damped cycles.

4.6 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

$$\gamma(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(\omega) e^{i\omega k} d\omega \tag{14}$$

This is convenient in situations where the spectral density is easier to calculate and manipulate than the autocovariance function.

(For example, the expression (12) for the ARMA spectral density is much easier to work with than the expression for the ARMA autocovariance)

4.7 Mathematical Theory

This section is loosely based on [3], 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

$$f = \sum_k \alpha_k h_k \quad \text{where} \quad \alpha_k := \langle f, h_k \rangle \tag{15}$$

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

$$T\gamma = \sum_{k \in \mathbb{Z}} \gamma(k) \frac{e^{i\omega k}}{\sqrt{2\pi}} = \frac{1}{\sqrt{2\pi}} f(\omega) \tag{16}$$

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 (16) gives (14), justifying our earlier expression for the inverse transform.

5 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

5.1 Application

Let's use this code to replicate the plots on pages 68–69 of [2].

Here are some functions to generate the plots

```
In [7]: 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):
            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=(12, 8))
            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()
```

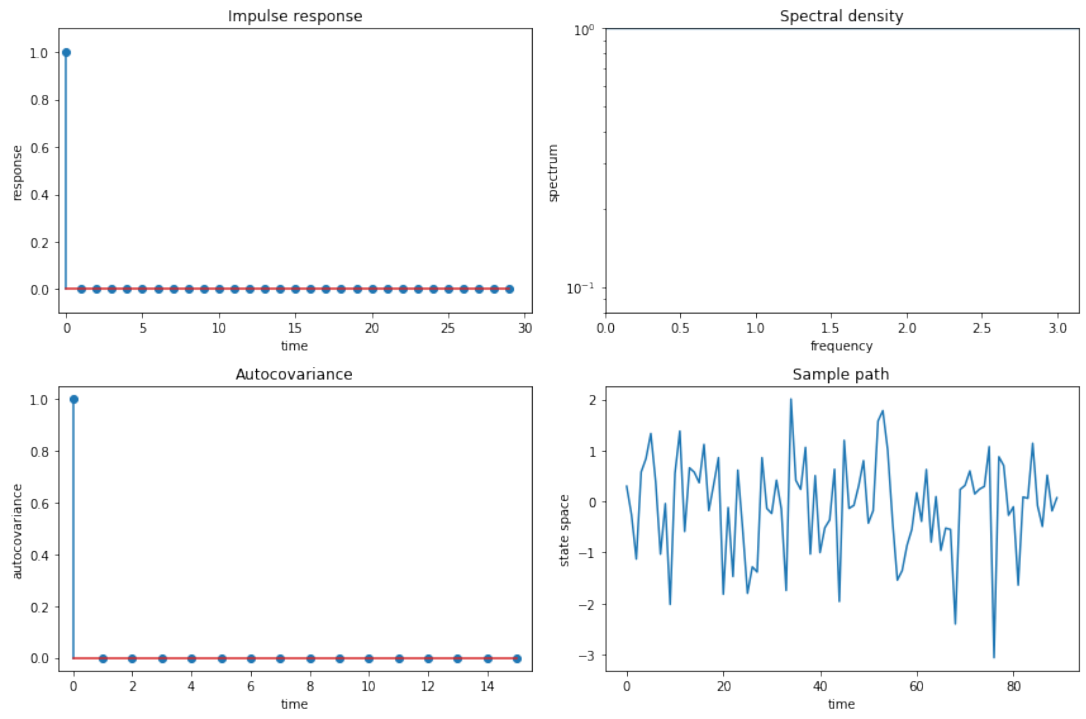
Now let’s call these functions to generate plots.

As a warmup, let’s make sure things look right when we for the pure white noise model $X_t = \epsilon_t$.

```
In [8]: ϕ = 0.0
```

```
θ = 0.0
arma = qe.ARMA(ϕ, θ)
quad_plot(arma)
```

```
    /home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
  """
/home/ubuntu/anaconda3/lib/python3.7/site-packages/numpy/core/_asarray.py:85:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:16:
UserWarning: Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
  app.launch_new_instance()
/home/ubuntu/anaconda3/lib/python3.7/site-packages/matplotlib/transforms.py:923:
ComplexWarning: Casting complex values to real discards the imaginary part
  self._points[:, 1] = interval
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
```



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.

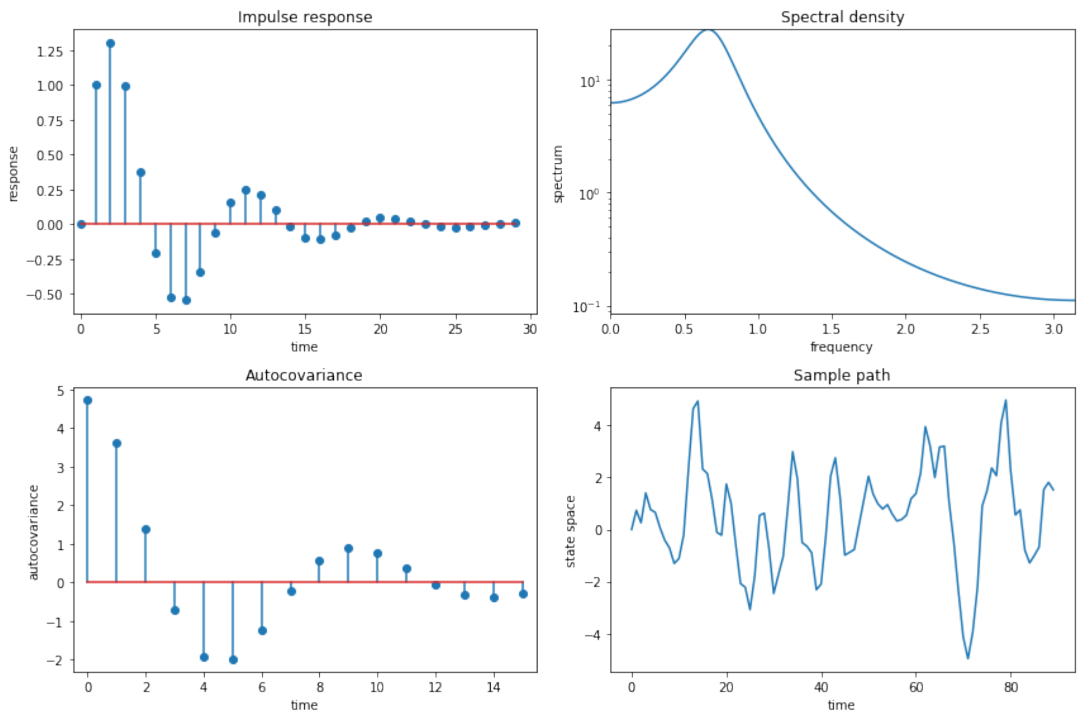
- it is actually challenging to visualize a time series realization of white noise – a sequence of surprises – but this too looks pretty good.

To get some more examples, as our laboratory we'll replicate quartets of graphs that [2] use to teach "how to read spectral densities".

Ljunqvist and Sargent's first model is $X_t = 1.3X_{t-1} - .7X_{t-2} + \epsilon_t$

```
In [9]: ϕ = 1.3, -.7
        θ = 0.0
        arma = qe.ARMA(ϕ, θ)
        quad_plot(arma)
```

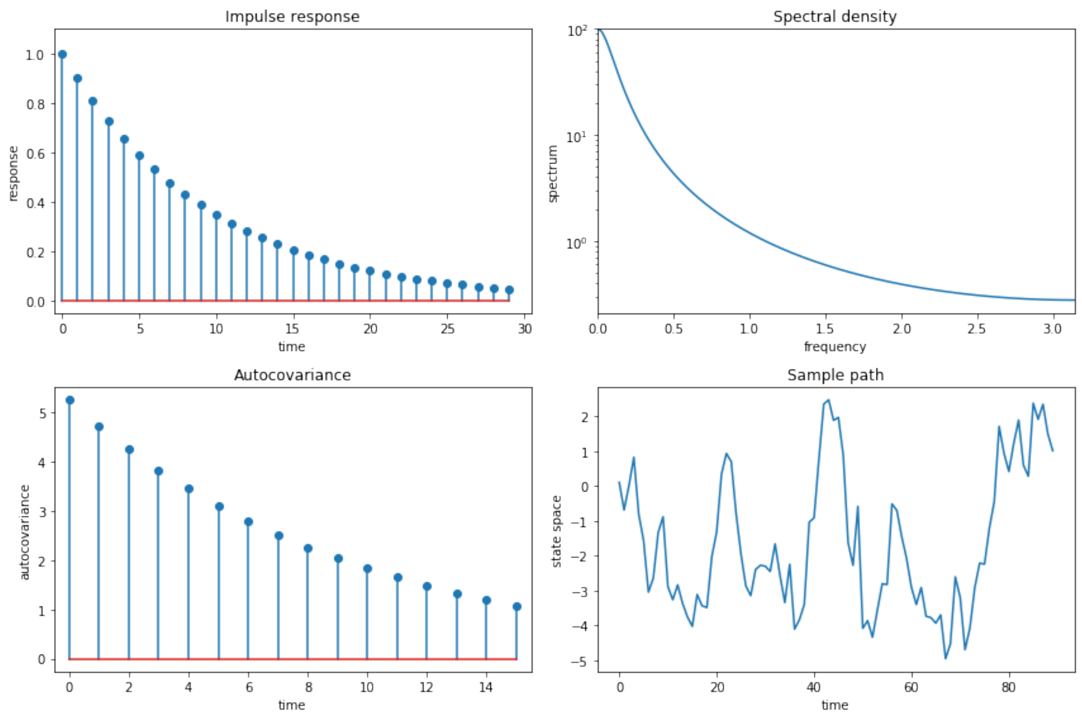
```
    /home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
  """
/home/ubuntu/anaconda3/lib/python3.7/site-packages/numpy/core/_asarray.py:85:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:16:
UserWarning: Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
  app.launch_new_instance()
/home/ubuntu/anaconda3/lib/python3.7/site-packages/matplotlib/transforms.py:923:
ComplexWarning: Casting complex values to real discards the imaginary part
  self._points[:, 1] = interval
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
```



Ljungqvist and Sargent's second model is $X_t = .9X_{t-1} + \epsilon_t$

```
In [10]: ϕ = 0.9
         θ = -0.0
         arma = qe.ARMA(ϕ, θ)
         quad_plot(arma)
```

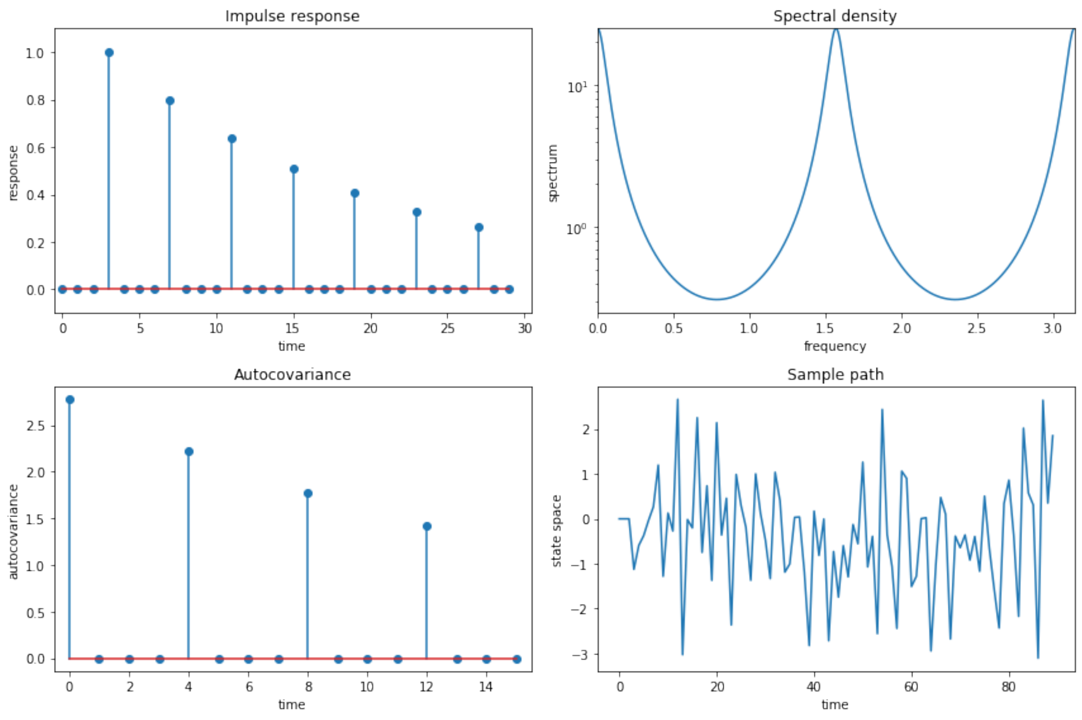
```
    /home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
  """
/home/ubuntu/anaconda3/lib/python3.7/site-packages/numpy/core/_asarray.py:85:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:16:
UserWarning: Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
  app.launch_new_instance()
/home/ubuntu/anaconda3/lib/python3.7/site-packages/matplotlib/transforms.py:923:
ComplexWarning: Casting complex values to real discards the imaginary part
  self._points[:, 1] = interval
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
```



Ljungqvist and Sargent's third model is $X_t = .8X_{t-4} + \epsilon_t$

```
In [11]: ϕ = 0., 0., 0., .8
         θ = -0.0
         arma = qe.ARMA(ϕ, θ)
         quad_plot(arma)
```

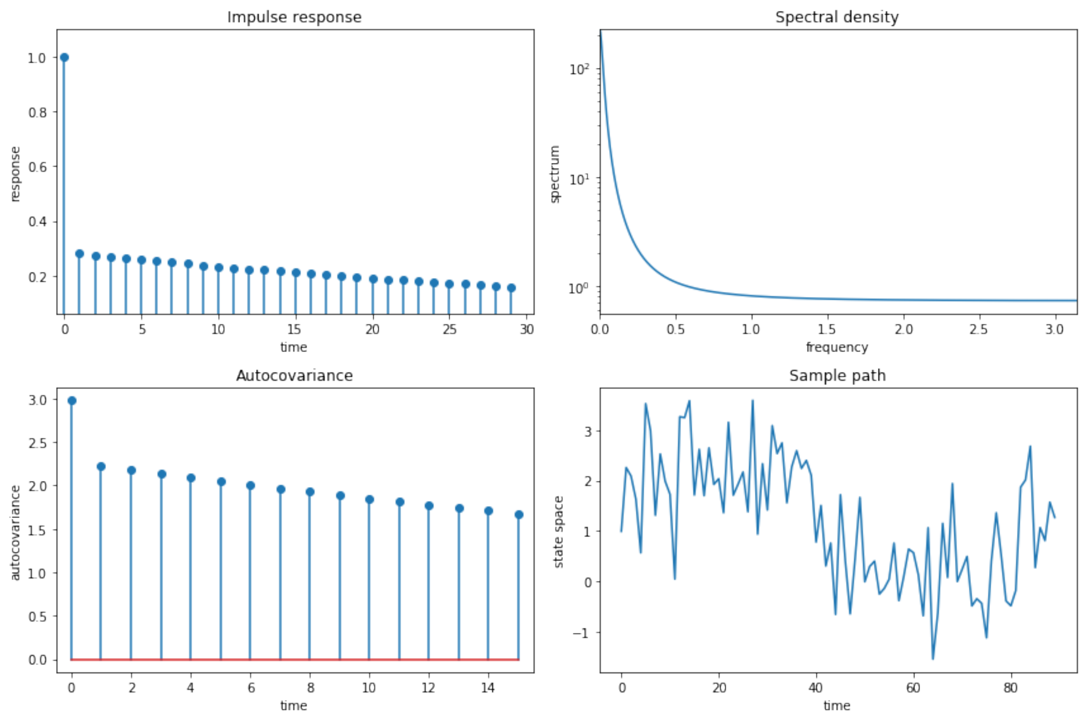
```
  /home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
  """
/home/ubuntu/anaconda3/lib/python3.7/site-packages/numpy/core/_asarray.py:85:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:16:
UserWarning: Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
  app.launch_new_instance()
/home/ubuntu/anaconda3/lib/python3.7/site-packages/matplotlib/transforms.py:923:
ComplexWarning: Casting complex values to real discards the imaginary part
  self._points[:, 1] = interval
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
```



Ljungqvist and Sargent's fourth model is $X_t = .98X_{t-1} + \epsilon_t - .7\epsilon_{t-1}$

```
In [12]: ϕ = .98
         θ = -0.7
         arma = qe.ARMA(ϕ, θ)
         quad_plot(arma)
```

```
    /home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
  """
/home/ubuntu/anaconda3/lib/python3.7/site-packages/numpy/core/_asarray.py:85:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:16:
UserWarning: Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
  app.launch_new_instance()
/home/ubuntu/anaconda3/lib/python3.7/site-packages/matplotlib/transforms.py:923:
ComplexWarning: Casting complex values to real discards the imaginary part
  self._points[:, 1] = interval
/home/ubuntu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:23:
UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a
LineCollection instead of individual lines. This significantly improves the
performance of a stem plot. To remove this warning and switch to the new behaviour,
set the "use_line_collection" keyword argument to True.
```



5.2 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 (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 (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.

5.3 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 (14), we have now shown that, for n sufficiently large, $a_k \approx \gamma(k)$ — which is exactly what we want to compute.

References

[1] J. D. Cryer and K-S. Chan. *Time Series Analysis.* Springer, 2nd edition edition, 2008.

[2] L Ljungqvist and T J Sargent. *Recursive Macroeconomic Theory.* MIT Press, 4 edition, 2018.

[3] Thomas J Sargent. *Macroeconomic Theory.* Academic Press, New York, 2nd edition, 1987.

[4] A N Shiriaev. *Probability.* Graduate texts in mathematics. Springer. Springer, 2nd edition, 1995.