AR1 Processes

Thomas J. Sargent and John Stachurski

September 12, 2020

1 Contents

• Overview 2
• The AR(1) Model 3
• Stationarity and Asymptotic Stability 4
• Ergodicity 5
• Exercises 6
• Solutions 7

2 Overview

In this lecture we are going to study a very simple class of stochastic models called AR(1) processes.

These simple models are used again and again in economic research to represent the dynamics of series such as

• labor income
• dividends
• productivity, etc.

AR(1) processes can take negative values but are easily converted into positive processes when necessary by a transformation such as exponentiation.

We are going to study AR(1) processes partly because they are useful and partly because they help us understand important concepts.

Let’s start with some imports:

In [1]: import numpy as np
   import matplotlib.pyplot as plt
   %matplotlib inline
3 The AR(1) Model

The AR(1) model (autoregressive model of order 1) takes the form

\[ X_{t+1} = aX_t + b + cW_{t+1} \]  

(1)

where \(a, b, c\) are scalar-valued parameters.

This law of motion generates a time series \(\{X_t\}\) as soon as we specify an initial condition \(X_0\). This is called the state process and the state space is \(\mathbb{R}\).

To make things even simpler, we will assume that

- the process \(\{W_t\}\) is IID and standard normal,
- the initial condition \(X_0\) is drawn from the normal distribution \(N(\mu_0, \nu_0)\) and
- the initial condition \(X_0\) is independent of \(\{W_t\}\).

3.1 Moving Average Representation

Iterating backwards from time \(t\), we obtain

\[ X_t = aX_{t-1} + b + cW_t = a^2X_{t-2} + ab + acW_{t-1} + b + cW_t = \cdots \]

If we work all the way back to time zero, we get

\[ X_t = a^tX_0 + b \sum_{j=0}^{t-1} a^j + c \sum_{j=0}^{t-1} a^jW_{t-j} \]  

(2)

Equation (2) shows that \(X_t\) is a well defined random variable, the value of which depends on

- the parameters,
- the initial condition \(X_0\) and
- the shocks \(W_1, \ldots, W_t\) from time \(t = 1\) to the present.

Throughout, the symbol \(\psi_t\) will be used to refer to the density of this random variable \(X_t\).

3.2 Distribution Dynamics

One of the nice things about this model is that it’s so easy to trace out the sequence of distributions \(\{\psi_t\}\) corresponding to the time series \(\{X_t\}\).

To see this, we first note that \(X_t\) is normally distributed for each \(t\).

This is immediate form (2), since linear combinations of independent normal random variables are normal.

Given that \(X_t\) is normally distributed, we will know the full distribution \(\psi_t\) if we can pin down its first two moments.

Let \(\mu_t\) and \(\nu_t\) denote the mean and variance of \(X_t\) respectively.

We can pin down these values from (2) or we can use the following recursive expressions:
\[ \mu_{t+1} = a \mu_t + b \quad \text{and} \quad v_{t+1} = a^2 v_t + c^2 \]  

These expressions are obtained from (1) by taking, respectively, the expectation and variance of both sides of the equality.

In calculating the second expression, we are using the fact that \( X_t \) and \( W_{t+1} \) are independent. (This follows from our assumptions and (2).)

Given the dynamics in (2) and initial conditions \( \mu_0, v_0 \), we obtain \( \mu_t, v_t \) and hence

\[ \psi_t = N(\mu_t, v_t) \]

The following code uses these facts to track the sequence of marginal distributions \( \{\psi_t\} \).

The parameters are

In [2]: \( a, b, c = 0.9, 0.1, 0.5 \)

\( \mu, v = -3.0, 0.6 \) # initial conditions \( \mu_0, v_0 \)

Here’s the sequence of distributions:

In [3]: from scipy.stats import norm

\[
sim\_length = 10
grid = np.linspace(-5, 7, 120)
\]

fig, ax = plt.subplots()

for t in range(sim_length):
    mu = a * mu + b
    v = a**2 * mu + c**2
    ax.plot(grid, norm.pdf(grid, loc=mu, scale=np.sqrt(v)),
            label=f"\$\psi\_\{t\}\$",
            alpha=0.7)

ax.legend(bbox_to_anchor=[1.05, 1], loc=2, borderaxespad=1)
plt.show()
4 Stationarity and Asymptotic Stability

Notice that, in the figure above, the sequence \( \{ \psi_t \} \) seems to be converging to a limiting distribution.

This is even clearer if we project forward further into the future:

```
In [4]: def plot_density_seq(ax, mu_0=-3.0, v_0=0.6, sim_length=60):
    mu, v = mu_0, v_0
    for t in range(sim_length):
        mu = a * mu + b
        v = a**2 * v + c**2
        ax.plot(grid,
                norm.pdf(grid, loc=mu, scale=np.sqrt(v)),
                alpha=0.5)

fig, ax = plt.subplots()
plot_density_seq(ax)
plt.show()
```
Moreover, the limit does not depend on the initial condition.

For example, this alternative density sequence also converges to the same limit.

In [5]: fig, ax = plt.subplots()
   plot_density_seq(ax, mu_0=3.0)
   plt.show()

In fact it’s easy to show that such convergence will occur, regardless of the initial condition, whenever $|a| < 1$. 
To see this, we just have to look at the dynamics of the first two moments, as given in (3).

When $|a| < 1$, these sequence converge to the respective limits

$$
\mu^* := \frac{b}{1-a} \quad \text{and} \quad v^* = \frac{c^2}{1-a^2}
$$

(See our lecture on one dimensional dynamics for background on deterministic convergence.)

Hence

$$
\psi_t \to \psi^* = N(\mu^*, v^*) \quad \text{as} \ t \to \infty
$$

We can confirm this is valid for the sequence above using the following code.

```python
In [6]: fig, ax = plt.subplots()
plot_density_seq(ax, mu_0=3.0)

mu_star = b / (1 - a)
std_star = np.sqrt(c**2 / (1 - a**2))  # square root of v_star
psi_star = norm.pdf(grid, loc=mu_star, scale=std_star)
ax.plot(grid, psi_star, 'k-', lw=2, label="$\psi^*$")
ax.legend()
plt.show()
```

As claimed, the sequence $\{\psi_t\}$ converges to $\psi^*$. 

6
4.1 Stationary Distributions

A stationary distribution is a distribution that is a fixed point of the update rule for distributions.

In other words, if \( \psi_t \) is stationary, then \( \psi_{t+j} = \psi_t \) for all \( j \) in \( \mathbb{N} \).

A different way to put this, specialized to the current setting, is as follows: a density \( \psi \) on \( \mathbb{R} \) is stationary for the AR(1) process if

\[
X_t \sim \psi \implies aX_t + b + cW_{t+1} \sim \psi
\]

The distribution \( \psi^* \) in (5) has this property — checking this is an exercise.

(Of course, we are assuming that \( |a| < 1 \) so that \( \psi^* \) is well defined.)

In fact, it can be shown that no other distribution on \( \mathbb{R} \) has this property.

Thus, when \( |a| < 1 \), the AR(1) model has exactly one stationary density and that density is given by \( \psi^* \).

5 Ergodicity

The concept of ergodicity is used in different ways by different authors.

One way to understand it in the present setting is that a version of the Law of Large Numbers is valid for \( \{X_t\} \), even though it is not IID.

In particular, averages over time series converge to expectations under the stationary distribution.

Indeed, it can be proved that, whenever \( |a| < 1 \), we have

\[
\frac{1}{m} \sum_{t=1}^{m} h(X_t) \to \int h(x)\psi^*(x)dx \quad \text{as} \quad m \to \infty
\]

whenever the integral on the right hand side is finite and well defined.

Notes:

- In (6), convergence holds with probability one.
- The textbook by [1] is a classic reference on ergodicity.

For example, if we consider the identity function \( h(x) = x \), we get

\[
\frac{1}{m} \sum_{t=1}^{m} X_t \to \int x\psi^*(x)dx \quad \text{as} \quad m \to \infty
\]

In other words, the time series sample mean converges to the mean of the stationary distribution.

As will become clear over the next few lectures, ergodicity is a very important concept for statistics and simulation.
6 Exercises

6.1 Exercise 1

Let \( k \) be a natural number. The \( k \)-th central moment of a random variable is defined as

\[
M_k := \mathbb{E}[(X - \mathbb{E}X)^k]
\]

When that random variable is \( N(\mu, \sigma^2) \), it is known that

\[
M_k = \begin{cases} 
0 & \text{if } k \text{ is odd} \\
\sigma^k(k-1)!! & \text{if } k \text{ is even}
\end{cases}
\]

Here \( n!! \) is the double factorial.

According to (6), we should have, for any \( k \in \mathbb{N} \),

\[
\frac{1}{m} \sum_{t=1}^{m} (X_t - \mu^*)^k \approx M_k
\]

when \( m \) is large.

Confirm this by simulation at a range of \( k \) using the default parameters from the lecture.

6.2 Exercise 2

Write your own version of a one dimensional kernel density estimator, which estimates a density from a sample.

Write it as a class that takes the data \( X \) and bandwidth \( h \) when initialized and provides a method \( f \) such that

\[
f(x) = \frac{1}{hn} \sum_{i=1}^{n} K \left( \frac{x - X_i}{h} \right)
\]

For \( K \) use the Gaussian kernel (\( K \) is the standard normal density).

Write the class so that the bandwidth defaults to Silverman’s rule (see the “rule of thumb” discussion on this page). Test the class you have written by going through the steps

1. simulate data \( X_1, \ldots, X_n \) from distribution \( \phi \)
2. plot the kernel density estimate over a suitable range
3. plot the density of \( \phi \) on the same figure

for distributions \( \phi \) of the following types

- beta distribution with \( \alpha = \beta = 2 \)
- beta distribution with \( \alpha = 2 \) and \( \beta = 5 \)
- beta distribution with \( \alpha = \beta = 0.5 \)

Use \( n = 500 \).

Make a comment on your results. (Do you think this is a good estimator of these distributions?)
6.3 Exercise 3

In the lecture we discussed the following fact: for the AR(1) process

\[ X_{t+1} = aX_t + b + cW_{t+1} \]

with \( \{W_t\} \) iid and standard normal,

\[ \psi_t = N(\mu, s^2) \Rightarrow \psi_{t+1} = N(a\mu + b, a^2s^2 + c^2) \]

Confirms this, at least approximately, by simulation. Let

- \( a = 0.9 \)
- \( b = 0.0 \)
- \( c = 0.1 \)
- \( \mu = -3 \)
- \( s = 0.2 \)

First, plot \( \psi_t \) and \( \psi_{t+1} \) using the true distributions described above.

Second, plot \( \psi_{t+1} \) on the same figure (in a different color) as follows:

1. Generate \( n \) draws of \( X_t \) from the \( N(\mu, s^2) \) distribution
2. Update them all using the rule \( X_{t+1} = aX_t + b + cW_{t+1} \)
3. Use the resulting sample of \( X_{t+1} \) values to produce a density estimate via kernel density estimation.

Try this for \( n = 2000 \) and confirm that the simulation based estimate of \( \psi_{t+1} \) does converge to the theoretical distribution.

7 Solutions

7.1 Exercise 1

In [7]: from numba import njit
from scipy.special import factorial2

@njit
def sample_moments_ar1(k, m=100_000, mu_0=0.0, sigma_0=1.0, seed=1234):
    np.random.seed(seed)
    sample_sum = 0.0
    x = mu_0 + sigma_0 * np.random.randn()
    for t in range(m):
        sample_sum += (x - mu_star)**k
        x = a * x + b + c * np.random.randn()
    return sample_sum / m

def true_moments_ar1(k):
    if k % 2 == 0:
        return std_star**k * factorial2(k - 1)
    else:
return 0

k_vals = np.arange(6) + 1
sample_moments = np.empty_like(k_vals)
true_moments = np.empty_like(k_vals)

for k_idx, k in enumerate(k_vals):
    sample_moments[k_idx] = sample_moments_ar1(k)
    true_moments[k_idx] = true_moments_ar1(k)

fig, ax = plt.subplots()
ax.plot(k_vals, true_moments, label="true moments")
ax.plot(k_vals, sample_moments, label="sample moments")
ax.legend()

plt.show()

7.2 Exercise 2

Here is one solution:

In [8]: K = norm.pdf

class KDE:
    
    def __init__(self, x_data, h=None):
        
        if h is None:
            c = x_data.std()
            n = len(x_data)
            h = 1.06 * c * n**(-1/5)

10
self.h = h
self.x_data = x_data

def f(self, x):
    if np.isscalar(x):
        return K((x - self.x_data) / self.h).mean() * (1/self.h)
    else:
        y = np.empty_like(x)
        for i, x_val in enumerate(x):
            y[i] = K((x_val - self.x_data) / self.h).mean() * (1/self.h)
        return y

In [9]:
   def plot_kde(ϕ, x_min=-0.2, x_max=1.2):
       x_data = ϕ.rvs(n)
       kde = KDE(x_data)

       x_grid = np.linspace(-0.2, 1.2, 100)
       fig, ax = plt.subplots()
       ax.plot(x_grid, kde.f(x_grid), label="estimate")
       ax.plot(x_grid, ϕ.pdf(x_grid), label="true density")
       ax.legend()
       plt.show()

In [10]:
   from scipy.stats import beta

   n = 500
   parameter_pairs = (2, 2), (2, 5), (0.5, 0.5)
   for α, β in parameter_pairs:
       plot_kde(beta(α, β))
We see that the kernel density estimator is effective when the underlying distribution is smooth but less so otherwise.

### 7.3 Exercise 3

Here is our solution
In [11]:
a = 0.9
b = 0.0
c = 0.1
μ = -3
s = 0.2

In [12]:
μ_next = a * μ + b
s_next = np.sqrt(a**2 * s**2 + c**2)

In [13]:
ψ = lambda x: K((x - μ) / s)
ψ_next = lambda x: K((x - μ_next) / s_next)

In [14]:
ψ = norm(μ, s)
ψ_next = norm(μ_next, s_next)

In [15]:
n = 2000
x_draws = ψ.rvs(n)
x_draws_next = a * x_draws + b + c * np.random.randn(n)
kde = KDE(x_draws_next)

x_grid = np.linspace(μ - 1, μ + 1, 100)
fig, ax = plt.subplots()

ax.plot(x_grid, ψ.pdf(x_grid), label="ψ_t")
ax.plot(x_grid, ψ_next.pdf(x_grid), label="ψ_{t+1}")
ax.plot(x_grid, kde.f(x_grid), label="estimate of ψ_{t+1}")

ax.legend()
plt.show()

The simulated distribution approximately coincides with the theoretical distribution, as predicted.
References