Introducing an AR Model

TIME SERIES ANALYSIS IN PYTHON

Rob Reider
Adjunct Professor, NYU-Courant
Consultant, Quantopian

Mathematical Description of AR(1) Model

Today's value equals a mean, plus a fraction phi, of yesterday's value, plus noise.

AR = Auto Regressive Model

$$R_t = \mu + \phi R_{t-1} + \epsilon_t$$

- Since only one lagged value on right hand side, this is called:
  - AR model of order 1, or
  - AR(1) model
- AR parameter is ϕ <— If Phi = 1 then the series is a random walk, if phi is 0 then the process is white noise.
- For stationarity, $-1 < \phi < 1$ <— In order for the process ot be stables, phi has to be between 1 and -1

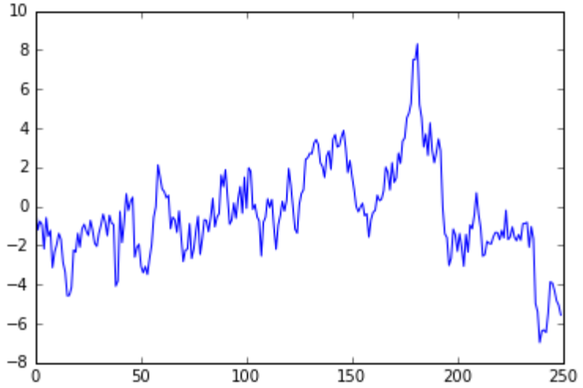
Interpretation of AR(1) Parameter

$$R_t = \mu + \phi R_{t-1} + \epsilon_t$$

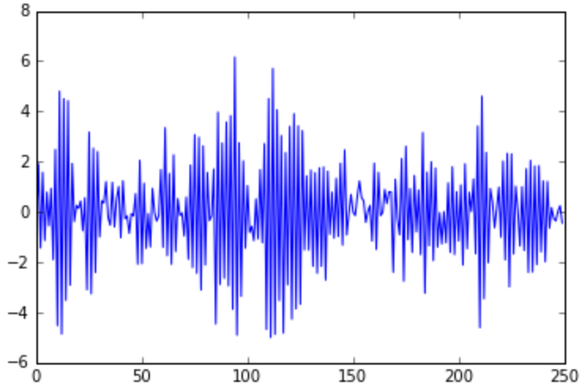
- Negative ϕ: Mean Reversion
- Positive ϕ: Momentum

Comparison of AR(1) Time Series

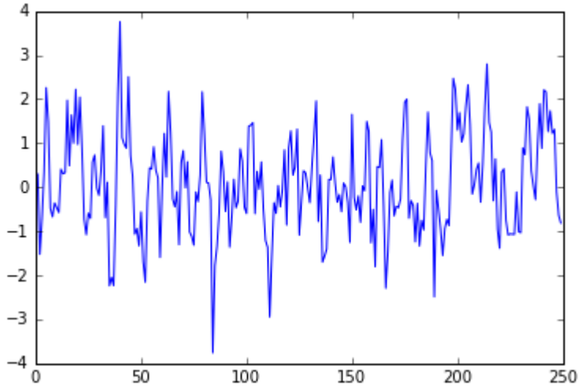
- $\phi = 0.9$



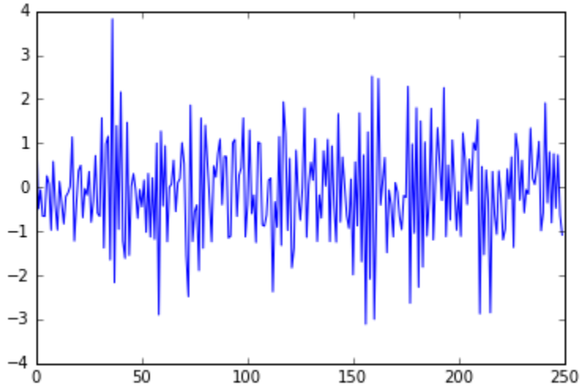
- $\phi = -0.9$



- $\phi = 0.5$

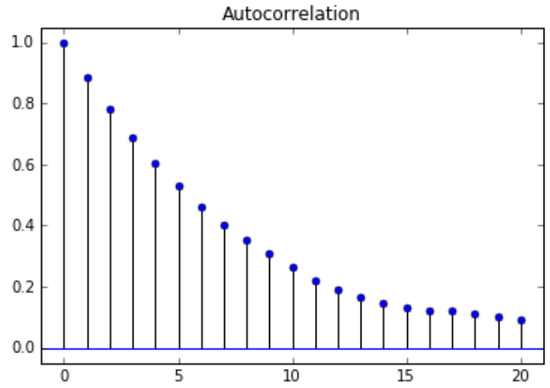


- $\phi = -0.5$

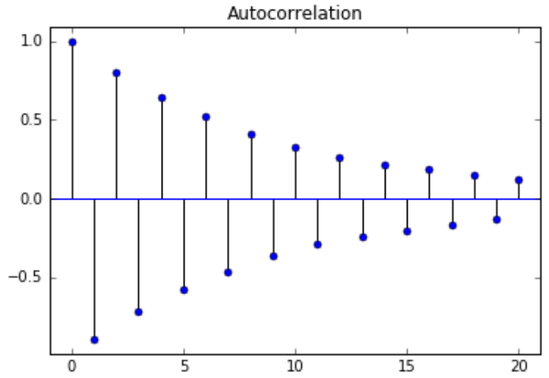


Comparison of AR(1) Autocorrelation Functions

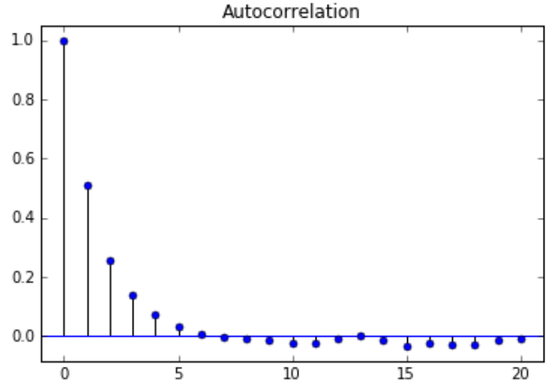
- $\phi = 0.9$



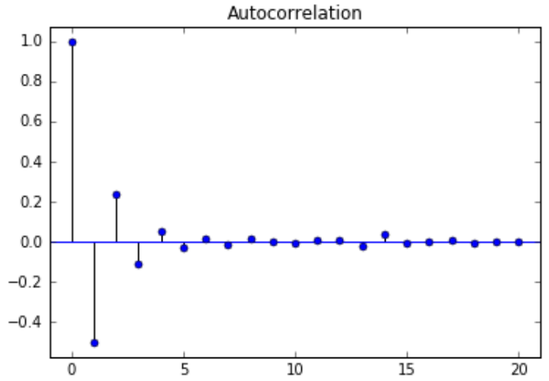
- $\phi = -0.9$



- $\phi = 0.5$



- $\phi = -0.5$



Higher Order AR Models

- AR(1)

$$R_t = \mu + \phi_1 R_{t-1} + \epsilon_t$$

  - AR(2)

$$R_t = \mu + \phi_1 R_{t-1} + \phi_2 R_{t-2} + \epsilon_t$$

  - AR(3)

$$R_t = \mu + \phi_1 R_{t-1} + \phi_2 R_{t-2} + \phi_3 R_{t-3} + \epsilon_t$$

  - ...

Simulating an AR Process

```
from statsmodels.tsa.arima_process import ArmaProcess
ar = np.array([1, -0.9]) <— You must include the 0 lag coefficient of 1, and the sign of the other coefficient is the opposite of what we've been using.
ma = np.array([1])
AR_object = ArmaProcess(ar, ma)
simulated_data = AR_object.generate_sample(nsample=1000)
plt.plot(simulated_data)
```

Ex: For a AR(1) process with phi equal to +0.9, the second element of the AR array should be the opposite sign: -0.9

```
# import the module for simulating data
from statsmodels.tsa.arima_process import ArmaProcess

# Plot 1: AR parameter = +0.9
plt.subplot(2,1,1)
ar1 = np.array([1, -0.9])
ma1 = np.array([1])
AR_object1 = ArmaProcess(ar1, ma1)
simulated_data_1 = AR_object1.generate_sample(nsample=1000)
plt.plot(simulated_data_1)

# Plot 2: AR parameter = -0.9
plt.subplot(2,1,2)
ar2 = np.array([1, 0.9])
ma2 = np.array([1])
AR_object2 = ArmaProcess(ar2, ma2)
simulated_data_2 = AR_object2.generate_sample(nsample=1000)
plt.plot(simulated_data_2)
plt.show()
```

The two AR parameters produce very different looking time series plots, but in the next exercise you'll really be able to distinguish the time series.

Let's practice!

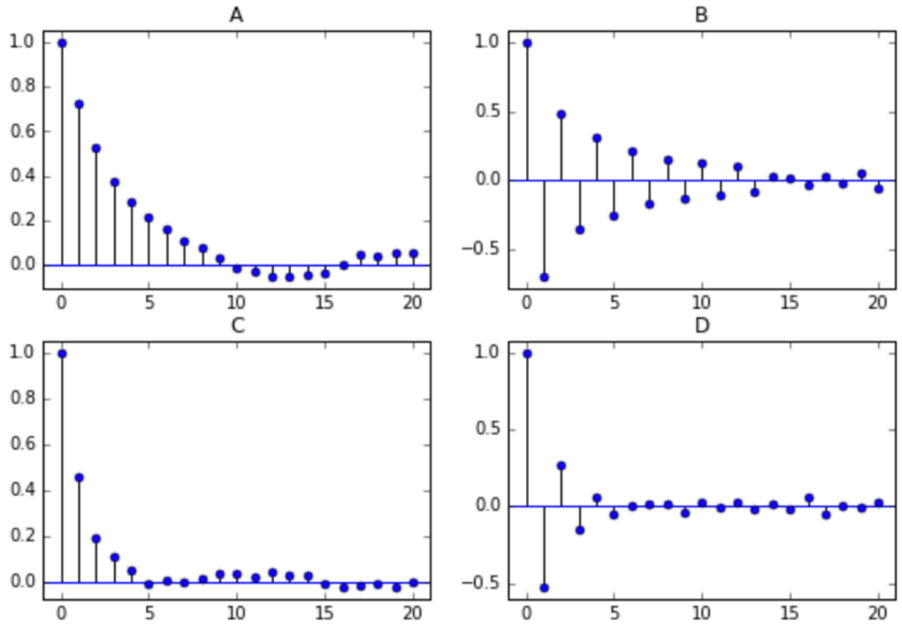
TIME SERIES ANALYSIS IN PYTHON

```
# Import the plot_acf module from statsmodels
from statsmodels.graphics.tsaplots import plot_acf

# Plot 1: AR parameter = +0.9
plot_acf(simulated_data_1, alpha=1, lags=20)
plt.show()

# Plot 2: AR parameter = -0.9
plot_acf(simulated_data_2, alpha=1, lags=20)
plt.show()

# Plot 3: AR parameter = +0.3
plot_acf(simulated_data_3, alpha=1, lags=20)
plt.show()
```

The autocorrelation function decays exponentially for an AR time series at a rate of the AR parameter. For example, if the AR parameter, ϕ = +0.9, the first-lag autocorrelation will be 0.9, the second-lag will be (0.9) ^^ 2=0.81, the third-lag will be (0.9) ^^ 3 = 0.729, etc. A smaller AR parameter will have a steeper decay, and for a negative AR parameter, say -0.9, the decay will flip signs, so the first-lag autocorrelation will be -0.9, the second-lag will be (–0.9) ^^ 2 = 0.81 , the third-lag will be (–0.9) ^^ 3 = –0.729, etc.

Match AR Model with ACF

Here are four Autocorrelation plots:



Which figure corresponds to an AR(1) model with an AR parameter of -0.5?

Answer the question 50 XP

Possible Answers

- A — press 1
- B — press 2
- C — press 3
- D — press 4

Estimating and Forecasting an AR Model

TIME SERIES ANALYSIS IN PYTHON

Rob Reider
Adjunct Professor, NYU-Courant
Consultant, Quantopian

Estimating an AR Model

- To estimate parameters from data (simulated)

```
from statsmodels.tsa.arima_model import ARMA
mod = ARMA(simulated_data, order=(1,0))
result = mod.fit()
```

Estimating an AR Model

- Full output (true $\mu = 0$ and $\phi = 0.9$)

```
print(result.summary())
```

```
                            ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 5000
Model:                     ARMA(1, 0)   Log Likelihood               -7178.386
Method:                       css-mle   S.D. of innovations              1.017
Date:                Fri, 01 Dec 2017   AIC                          14362.772
Time:                        15:34:50   BIC                          14382.324
Sample:                             0   HQIC                         14369.625

==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const         -0.0361      0.152     -0.238      0.812        -0.333     0.261
ar.L1.y        0.9054      0.006    151.020      0.000         0.894     0.917
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.1045           +0.0000j            1.1045            0.0000
-----------------------------------------------------------------------------
```

Estimating an AR Model

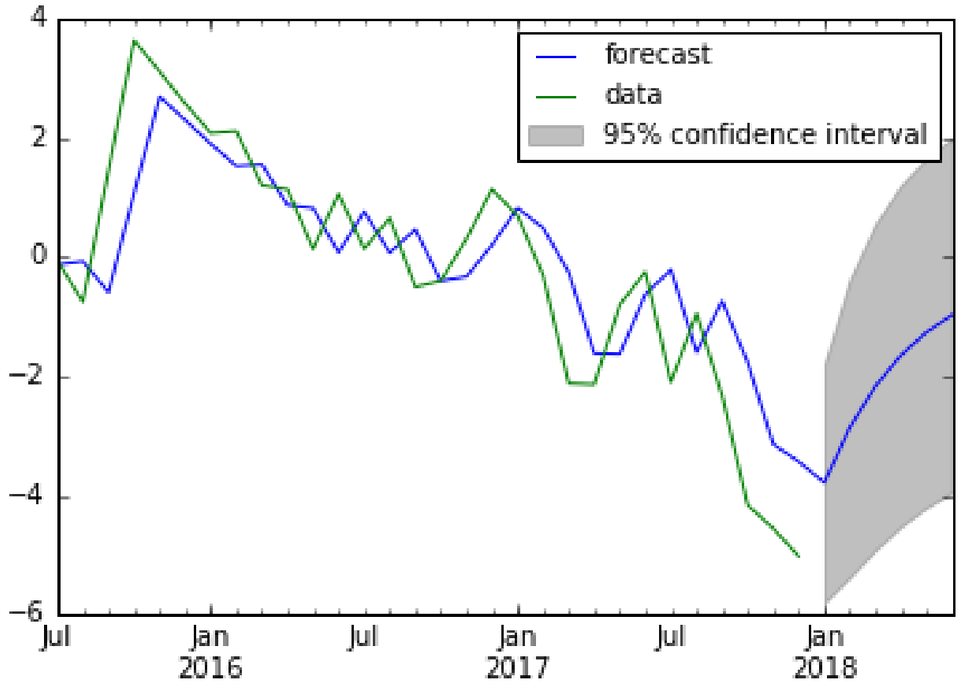
- Only the estimates of μ and ϕ (true $\mu = 0$ and $\phi = 0.9$)

```
print(result.params)
```

```
array([-0.03605989,  0.90535667])
```

Forecasting an AR Model

```
from statsmodels.tsa.arima_model import ARMA
mod = ARMA(simulated_data, order=(1,0))
res = mod.fit()
res.plot_predict(start='2016-07-01', end='2017-06-01')
plt.show()
```



The confidence interval gets wider the further out the forecast is.

```
# Import the ARMA module from statsmodels
from statsmodels.tsa.arima_model import ARMA

# Fit an AR(1) model to the first simulated data
mod = ARMA(simulated_data_1, order=(1,0))
res = mod.fit()

# Print out summary information on the fit
print(res.summary())

# Print out the estimate for the constant and for phi
print("When the true phi=0.9, the estimate of phi (and the constant) are:")
print(res.params)
```

Notice how close the estimated parameter is to the true parameter.

```
# Import the ARMA module from statsmodels
from statsmodels.tsa.arima_model import ARMA

# Forecast the first AR(1) model
mod = ARMA(simulated_data_1, order=(1,0))
res = mod.fit()
res.plot_predict(start=990, end=1010)
plt.show()
```

Notice how, when phi is high like here, the forecast gradually moves to the long term mean of zero, but if phi were low, it would move much quicker to the long term mean.

Let's practice!

TIME SERIES ANALYSIS IN PYTHON

You will now use the forecasting techniques you learned in the last exercise and apply it to real data rather than simulated data. You will revisit a dataset from the first chapter: the annual data of 10-year interest rates going back 56 years, which is in a Series called interest_rate_data. Being able to forecast interest rates is of enormous importance, not only for bond investors but also for individuals like new homeowners who must decide between fixed and floating rate mortgages.

You saw in the first chapter that there is some mean reversion in interest rates over long horizons. In other words, when interest rates are high, they tend to drop and when they are low, they tend to rise over time. Currently they are below long-term rates, so they are expected to rise, but an AR model attempts to quantify how much they are expected to rise.

```
# Import the ARMA module from statsmodels
from statsmodels.tsa.arima_model import ARMA

# Forecast interest rates using an AR(1) model
mod = ARMA(interest_rate_data, order=(1,0))
res = mod.fit()

# Plot the original series and the forecasted series
res.plot_predict(start = 0, end = '2022')
plt.legend(fontsize=8)
plt.show()
```

Sometimes it is difficult to distinguish between a time series that is slightly mean reverting and a time series that does not mean revert at all, like a random walk. You will compare the ACF for the slightly mean-reverting interest rate series of the last exercise with a simulated random walk with the same number of observations.

You should notice when plotting the autocorrelation of these two series side-by-side that they look very similar.

```
# Import the plot_acf module from statsmodels
from statsmodels.graphics.tsaplots import plot_acf

# Plot the interest rate series and the simulated random walk series side-by-side
fig, axes = plt.subplots(2,1)

# Plot the autocorrelation of the interest rate series in the top plot
fig = plot_acf(interest_rate_data, alpha=1, lags=12, ax=axes[0])

# Plot the autocorrelation of the simulated random walk series in the bottom plot
fig = plot_acf(simulated_data, alpha=1, lags=12, ax=axes[1])

# Label axes
axes[0].set_title("Interest Rate Data")
axes[1].set_title("Simulated Random Walk Data")
plt.show()
```

Choosing the Right Model

TIME SERIES ANALYSIS IN PYTHON

Rob Reider
Adjunct Professor, NYU-Courant
Consultant, Quantopian

Identifying the Order of an AR Model

- The order of an AR(p) model will usually be unknown
- Two techniques to determine order
  - Partial Autocorrelation Function
  - Information criteria

Partial Autocorrelation Function (PACF)

Measures the incremental benefit of adding another lag.

$$R_t = \phi_{0,1} + \phi_{1,1} R_{t-1} + \epsilon_{1t}$$

$$R_t = \phi_{0,2} + \phi_{1,2} R_{t-1} + \phi_{2,2} R_{t-2} + \epsilon_{2t}$$

$$R_t = \phi_{0,3} + \phi_{1,3} R_{t-1} + \phi_{2,3} R_{t-2} + \phi_{3,3} R_{t-3} + \epsilon_{3t}$$

$$R_t = \phi_{0,4} + \phi_{1,4} R_{t-1} + \phi_{2,4} R_{t-2} + \phi_{3,4} R_{t-3} + \phi_{4,4} R_{t-4} + \epsilon_{4t}$$

$$\vdots$$

Phi 4, 4 shows how significant adding a fourth lag is when you already have 3 lags in your model.

Plot PACF in Python

- Same as ACF, but use `plot_pacf` instead of `plt_acf`
- Import module

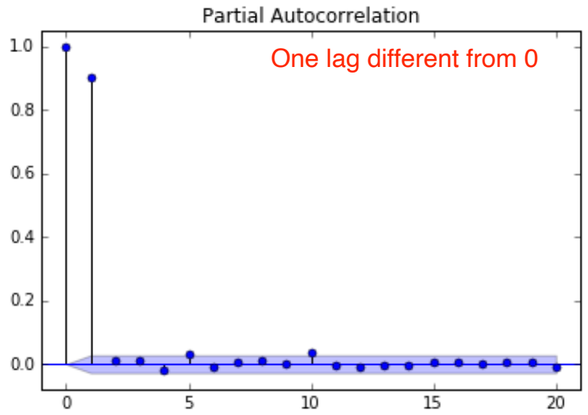
```
from statsmodels.graphics.tsaplots import plot_pacf
```

- Plot the PACF

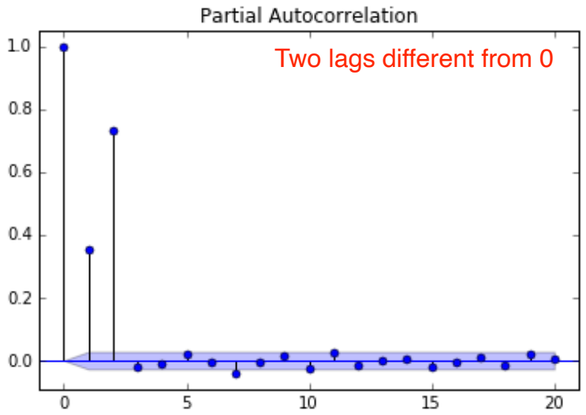
```
plot_pacf(x, lags= 20, alpha=0.05)
```

Comparison of PACF for Different AR Models

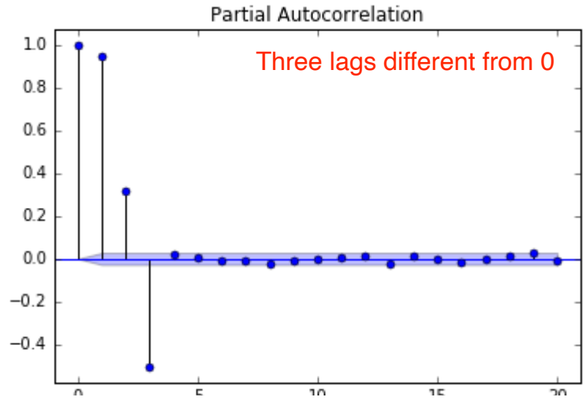
- AR(1)



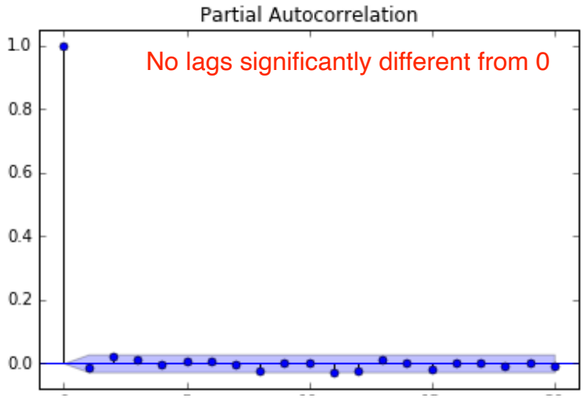
- AR(2)



- AR(3)



- White Noise



Information Criteria

- Information criteria: adjusts goodness-of-fit for number of parameters
- Two popular adjusted goodness-of-fit measures
  - AIC (Akaike Information Criterion)
  - BIC (Bayesian Information Criterion)

Information Criteria

- Estimation output

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 2500
Model:                     ARMA(2, 0)   Log Likelihood               -3536.481
Method:                       css-mle   S.D. of innovations              0.996
Date:                Fri, 29 Dec 2017   AIC                           7080.963
Time:                        22:53:24   BIC                           7104.259
Sample:                             0   HQIC                          7089.420

==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          0.0054      0.010      0.517      0.605        -0.015     0.026
ar.L1.y       -0.6130      0.019    -32.243      0.000        -0.650    -0.576
ar.L2.y       -0.3109      0.019    -16.351      0.000        -0.348    -0.274
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -0.9859           -1.4982j            1.7935           -0.3426
AR.2           -0.9859           +1.4982j            1.7935            0.3426
-----------------------------------------------------------------------------
```

Getting Information Criteria From `statsmodels`

- You learned earlier how to fit an AR model

```
from statsmodels.tsa.arima_model import ARMA
mod = ARMA(simulated_data, order=(1,0))
result = mod.fit()
```

- And to get full output

```
result.summary()
```

- Or just the parameters

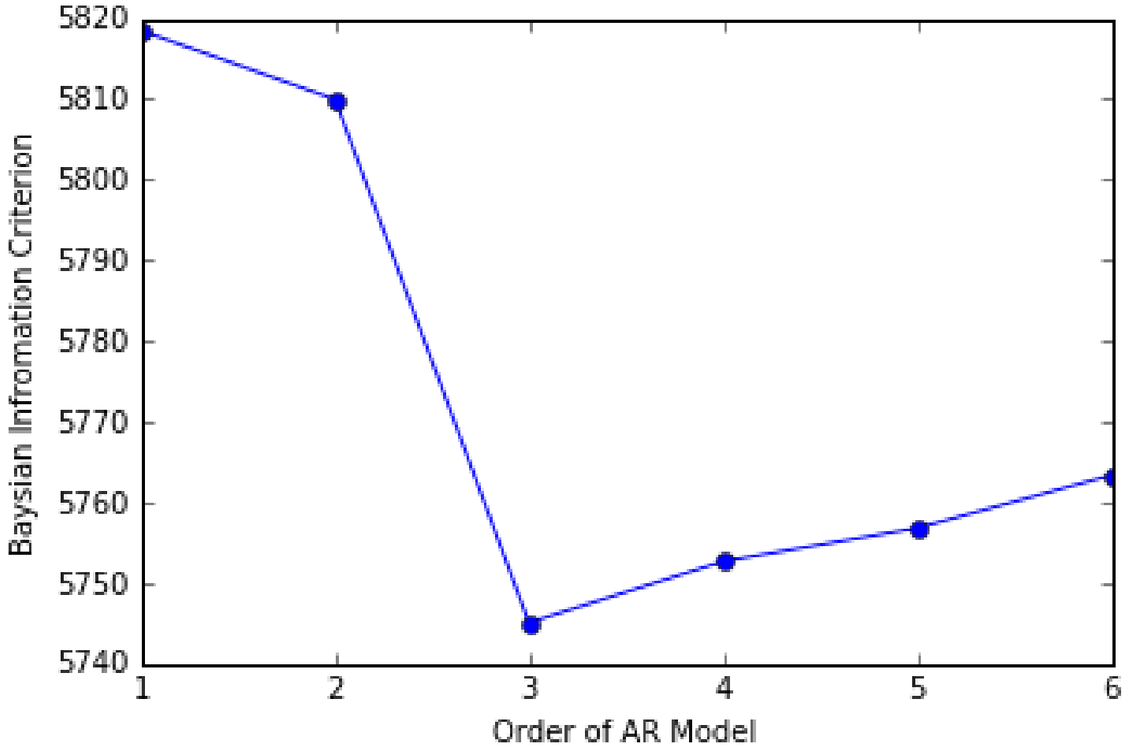
```
result.params
```

- To get the AIC and BIC

```
result.aic
result.bic
```

Information Criteria

- Fit a simulated AR(3) to different AR(p) models
- Choose p with the lowest BIC



One useful tool to identify the order of an AR model is to look at the Partial Autocorrelation Function (PACF). In this exercise, you will simulate two time series, an AR(1) and an AR(2), and calculate the sample PACF for each. You will notice that for an AR(1), the PACF should have a significant lag-1 value, and roughly zeros after that. And for an AR(2), the sample PACF should have significant lag-1 and lag-2 values, and zeros after that.

```
# Import the modules for simulating data and for plotting the PACF
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.graphics.tsaplots import plot_pacf

# Simulate AR(1) with phi=+0.6
ma = np.array([1])
ar = np.array([1, -0.6])
AR_object = ArmaProcess(ar, ma)
simulated_data_1 = AR_object.generate_sample(nsample=5000)

# Plot PACF for AR(1)
plot_pacf(simulated_data_1, lags=20)
plt.show()

# Simulate AR(2) with phi1=+0.6, phi2=+0.3
ma = np.array([1])
ar = np.array([1, -0.6, -0.3])
AR_object = ArmaProcess(ar, ma)
simulated_data_2 = AR_object.generate_sample(nsample=5000)

# Plot PACF for AR(2)
plot_pacf(simulated_data_2, lags=20)
plt.show()
```

Notice that the number of significant lags for the PACF indicate the order of the AR model

Let's practice!

TIME SERIES ANALYSIS IN PYTHON

Another tool to identify the order of a model is to look at the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). These measures compute the goodness of fit with the estimated parameters, but apply a penalty function on the number of parameters in the model. You will take the AR(2) simulated data from the last exercise, saved as simulated_data_2, and compute the BIC as you vary the order, p, in an AR(p) from 0 to 6.

```
# Import the module for estimating an ARMA model
from statsmodels.tsa.arima_model import ARMA

# Fit the data to an AR(p) for p = 0,...,6 , and save the BIC
BIC = np.zeros(7)
for p in range(7):
    mod = ARMA(simulated_data_2, order=(p,0))
    res = mod.fit()
# Save BIC for AR(p)
    BIC[p] = res.bic

# Plot the BIC as a function of p
plt.plot(range(1,7), BIC[1:7], marker='o')
plt.xlabel('Order of AR Model')
plt.ylabel('Bayesian Information Criterion')
plt.show()
```

For an AR(2), the BIC achieves its minimum at p=2, which is what we expect.