`

Chapter 5

Time Series Analysis: Concepts and Applications

As alluded to before, time series analysis (TSA) refers to study of time series signals from processes without exogenous inputs. Before we dive into the relatively more complex world of input-output modeling, it would be prudent to first get acquainted with how to model stochastic variations in signals and compactly describe the dependencies among adjacent observations in a time series. The study of time series is not just for pedagogical convenience. SysID derives many concepts from time series analysis and therefore, a strong foundation in modeling time series is important for mastering the art of SysID.

Time series analysis can help us to characterize I/O model residuals and get clues regarding further model refinement. If your process has measured disturbance signals (inputs that can't be manipulated; for example, ambient temperature), then they can be modeled via TSA to provide better forecasts and control. Furthermore, TSA can be used to model controlled process variables and build process monitoring tools. In this chapter, we will work through case-studies illustrating these applications.

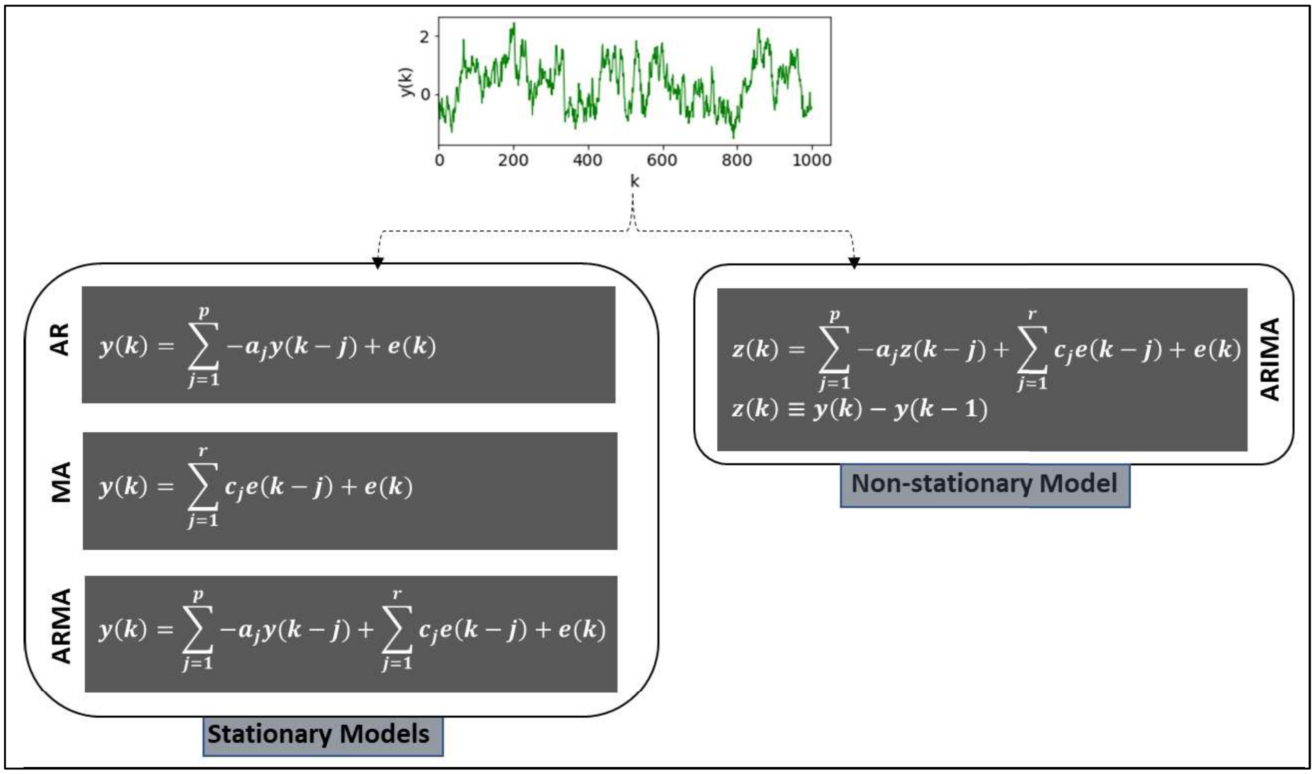
Apart from setting a strong foundation of digital signal processing, this chapter will also introduce the backshift notation and transfer function operator. These constructs are quite useful for compact description and algebraic manipulation of the difference equations. Specifically, the following topics are covered

- Introduction to AR, MA, and ARMA models for stationary signals
- Using ACF and PACF for model structure selection
- Monitoring controlled process variable in a CSTR using ARMA models
- Introduction to ARIMA models for nonstationary signals
- Forecasting measured signals using ARIMA models

`

5.1 Time Series Analysis: An Introduction

Time series analysis refers to the study and modeling of dependencies among sequential data points in time series signals. Several approaches exist for modeling time series. In this chapter, we will look at four of the most common models as shown in Figure 5.1. While AR, MA, and ARMA models are used to describe stationary signals, ARIMA is used for (homogeneous) nonstationary signals. In this chapter, we will understand the differences, similarities, and the inherent connections between these models.

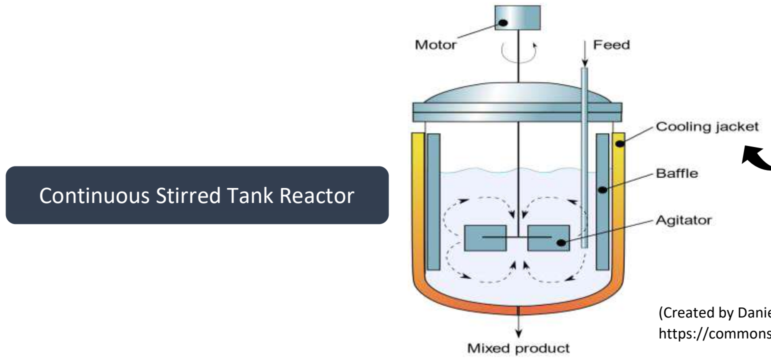


y(k)
k
AR
$y(k) = \sum_{j=1}^{p} -a_j y(k-j) + e(k)$
MA
$y(k) = \sum_{j=1}^{r} c_j e(k-j) + e(k)$
ARMA
$y(k) = \sum_{j=1}^{p} -a_j y(k-j) + \sum_{j=1}^{r} c_j e(k-j) + e(k)$
Stationary Models
ARIMA
$z(k) = \sum_{j=1}^{p} -a_j z(k-j) + \sum_{j=1}^{r} c_j e(k-j) + e(k)$
$z(k) \equiv y(k) - y(k-1)$
Non-stationary Model

Figure 5.1: Commonly employed univariate time-series models

To further motivate the study of time series analysis, let's briefly look at two of the use cases that were mentioned previously. The illustration below shows a CSTR unit that has measured disturbance signals (feed concentration and temperature). An accurate model for these signals can help generate accurate forecasts and control the vessel temperature more effectively. Also, it is easy to show that if the setpoint of the controllers do not change, then the controlled variables (here, output product temperature and concentration) depend on process disturbances alone. Therefore, a TSA model for these variables can be built to eventually look for significant mismatch between measurements and model predictions as an indicator of process faults.

`



Cooling jacket temperature is used to control the reactor/product temperature at optimal set-point

(Created by Daniele Pugliesi under Creative Commons Attribution-Share Alike 3.0, https://commons.wikimedia.org/wiki/File:Agitated_vessel.svg)

5.2 Autoregressive (AR) Models: An Introduction

AR models are among the simplest and easiest to understand time series models, wherein the current state is written as a linear combination of past measurements as shown below[38]

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) \cdots - a_p y(k-p) + e(k) \qquad \text{eq. 1}$$
$$= \sum_{j=1}^{p} -a_j y(k-j) + e(k)$$

The above model uses *p* lagged/past values and therefore is referred to as an AR(p) model or an autoregressive model of order *p*. In spite of its simplicity, an AR model can generate varied patterns of signals as shown below. In process industry, processes with inherent autoregressive behavior are fairly common.

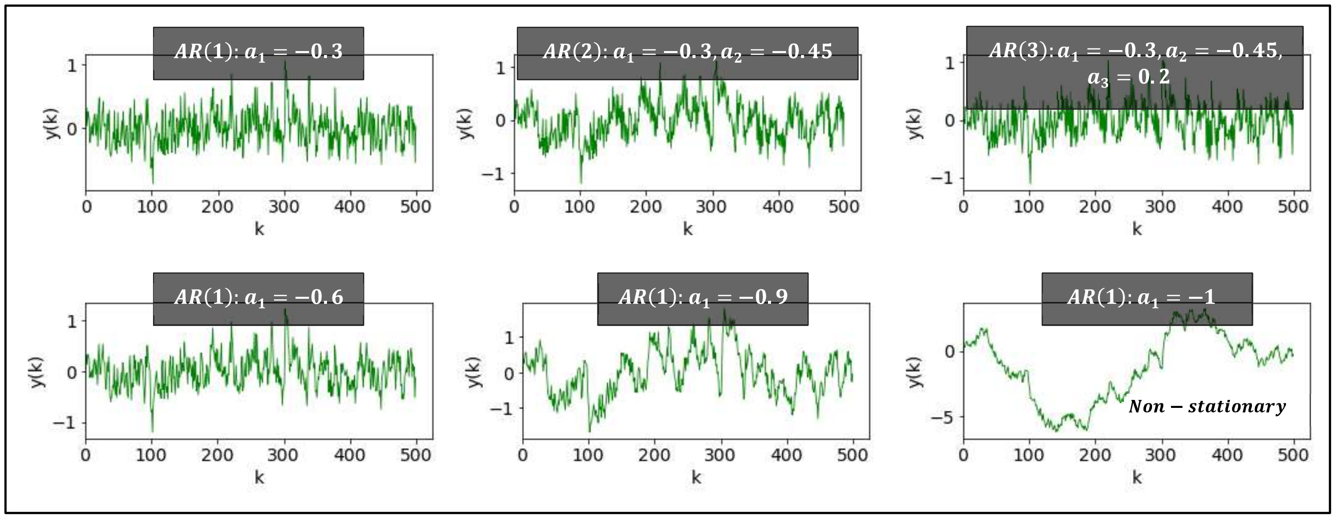


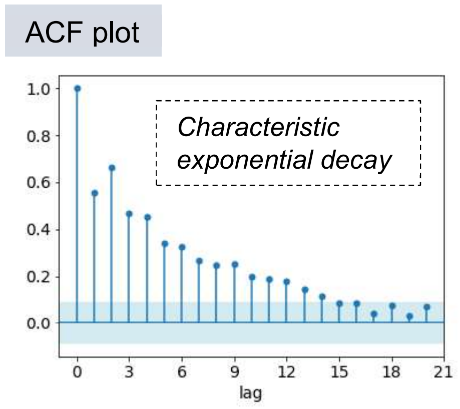
Figure 5.2: Time-series signals from AR processes

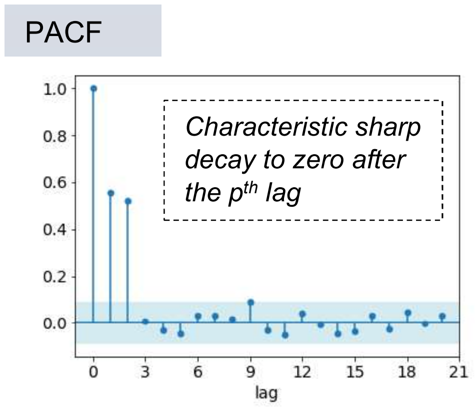
[38] You will see soon why we have used '$-a_j$' and not 'a_j' in the equation

`

Model order selection

Given any time series data, the suitability of an AR model and its potential order can be judged using the ACF[39] and PACF plots as shown below for the AR(2) plot in Figure 5.2.



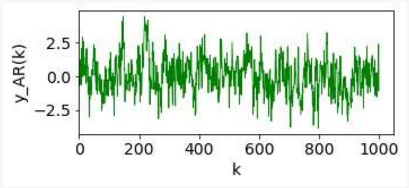


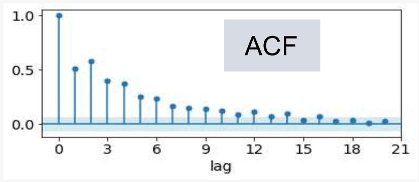
Let's see an example which shows how to generate data from an AR process and how to fit an AR model.

Example 5.1:

We will create an AR(2) signal with 1000 samples. Thereafter, guided by ACF/PACF, a model will be fitted to the data followed by residual diagnostics.

```
# import packages
import numpy as np, matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# generate data for AR(2) process using statsmodels.tsa package
ar_coeffs = np.array([1, -0.3, -0.45]) # a₁ = -0.3, a₂ = -0.45
ARprocess = ArmaProcess(ar_coeffs)
y_AR = ARprocess.generate_sample(nsample=1000)

# generate ACF and PACF plots for y_AR
conf_int = 2/np.sqrt(len(y_AR))
plot_acf(y_AR, lags= 20, alpha=None, title='')
plt.gca().axhspan(-conf_int, conf_int)
```

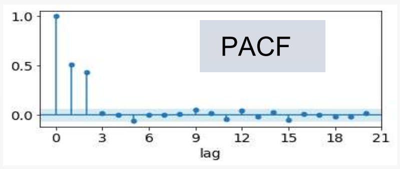




[39] In general, the ACF plot for an AR process may also show damped sine waves in addition to damped exponentials

`

```
plot_pacf(y_AR, lags= 20, alpha=None, title='')
plt.gca().axhspan(-conf_int, conf_int)
```



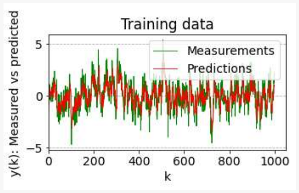
The PACF plot clearly indicates an AR(2) process. We will again use the *statsmodels.tsa* package to fit an AR(2) model.

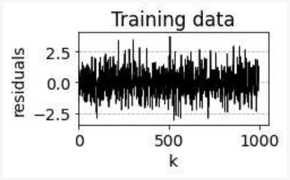
```
# Fit an AR(2) model
y_AR_centered = y_AR - np.mean(y_AR)
model = ARIMA(y_AR_centered, order=(2, 0, 0)) # order = (p,d,r)
results = model.fit()

# Print out the estimate for the parameters a1 and a2
print('[a1, a2] = ', -results.arparams)

>>> [a1, a2] =  [-0.27 -0.50]
```

AR model is linear-in-parameters and therefore, OLS method can be used to estimate the unknown model parameters.

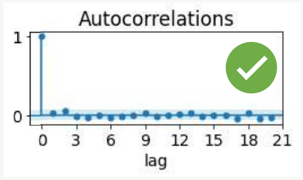
The estimated parameters are close to the true values (the inaccuracy in the estimates diminishes for larger sample size). With the obtained model, let's generate the model residuals and confirm whiteness of the residual sequence.

```
# get model predictions and residuals on training dataset
y_AR_centered_pred = results.predict()
residuals = y_AR_centered - y_AR_centered_pred

plt.figure(), plt.title('Training data')
plt.plot(y_AR_centered, 'g', label='Measurements')
plt.plot(y_AR_centered_pred, 'r', label='Predictions')
plt.ylabel('y(k): Measured vs predicted'), plt.xlabel('k'), plt.legend()

plt.figure(), plt.plot(residuals, 'black', linewidth=0.8)
plt.title('Training data'), plt.ylabel('residuals'), plt.xlabel('k')
```





```
# ACF residuals
plot_acf(residuals, lags= 20, alpha=None, title='')
plt.gca().axhspan(-conf_int, conf_int, facecolor='lightblue', alpha=0.5)
plt.xlabel('lag'), plt.title('Autocorrelations')
```



Shift operator and transfer function operator

The original form of the difference equations that we have worked with so far is not very convenient when combining different models and analyzing model properties. For a more compact and useful representation, the concept of backshift operator (*q*$^{-1}$) is introduced. It is defined as

$$y(k-1) = q^{-1}y(k)$$

Evidently, the backshift operator moves the signal one step back. Multiple applications shift the signal back further

$$q^{-1}\big(q^{-1}y(k)\big) = q^{-2}y(k) = y(k-2)$$

Let's rewrite Eq. 1 using the shift operator as follows

$$\begin{aligned}
& y(k) + \textstyle\sum_{j=1}^{p} a_j y(k-j) = e(k) \\
\Rightarrow\ & y(k) + \big(\textstyle\sum_{j=1}^{p} a_j\, q^{-j}\big) y(k) = e(k) \\
\Rightarrow\ & \big(1 + \textstyle\sum_{j=1}^{p} a_j\, q^{-j}\big) y(k) = e(k) \\
\Rightarrow\ & y(k) = \frac{1}{\left(1+\sum_{j=1}^{p} a_j q^{-j}\right)}\, e(k) \equiv H(q^{-1})e(k)
\end{aligned}$$

Let's see how we would represent a simple ARX model $y(k) + a_1 y(k-1) + a_2 y(k-2) = b_1 u(k-1) + b_2 u(k-2) + e(k)$ using the *q* notation

$$(1 + a_1 q^{-1} + a_2 q^{-2}) y(k) = q^{-1}(b_1 + b_2 q^{-1}) u(k) + e(k)$$

$$\Rightarrow y(k) = \underbrace{G(q^{-1})}_{\dfrac{q^{-1}(b_1 + b_2 q^{-1})}{1 + a_1 q^{-1} + a_2 q^{-2}}} u(k) + \underbrace{H(q^{-1})}_{\dfrac{1}{1 + a_1 q^{-1} + a_2 q^{-2}}} e(k) \qquad \text{eq. 2}$$

Note that *G(q*$^{-1}$*)* and *H(q*$^{-1}$*)* are (input and noise transfer) operators (and not multipliers) that operate on *u(k)* and *e(k)*, respectively, and determine how the incoming signals are 'transferred' to the output. Therefore, *G(q*$^{-1}$*)* and *H(q*$^{-1}$*)* are called transfer operators. Equation 2 can be written further as

$$y(k) = \underbrace{y_u(k)}_{\text{deterministic output}} + \underbrace{v(k)}_{\text{disturbance signal}}$$

`

where, $y_u(k) = G(q^{-1})u(k)$ eq. 3

$$v(k) = H(q^{-1})e(k)$$

Note that Eq. 3 clearly shows that in an ARX model, the process disturbance *v(k)* is a colored signal and not a white noise (*e(k)*). It is evident, therefore, that the transfer operator form makes it easy to dissect the stochastic and deterministic parts of the model.

The *q* notation may not seem appealing to you immediately, but soon enough you will begin to appreciate the convenience it offers! For now, just note that $G(q^{-1})$ and $H(q^{-1})$ can be manipulated like polynomials of q^{-1} as you will soon see.

5.3 Moving Average (MA) Models: An Introduction

In MA models the current state is a summation of current and past white noise error values as shown below

$$y(k) = c_1 e(k-1) + c_2 e(k-2) \cdots + c_r e(k-r) + e(k) \qquad \text{eq. 4}$$

$$= \sum_{j=1}^{r} c_j e(k-j) + e(k)$$

$$= \left(1 + \sum_{j=1}^{r} c_j q^{-j}\right) e(k)$$

The above model is referred to as an MA(r) model. An MA model can be interpreted as a counterpart of FIR models for time series signals where the effect of past random shocks/noise are propagated directly to future values of the time series. Plots below provide a sample of the patterns generated from MA models.

`

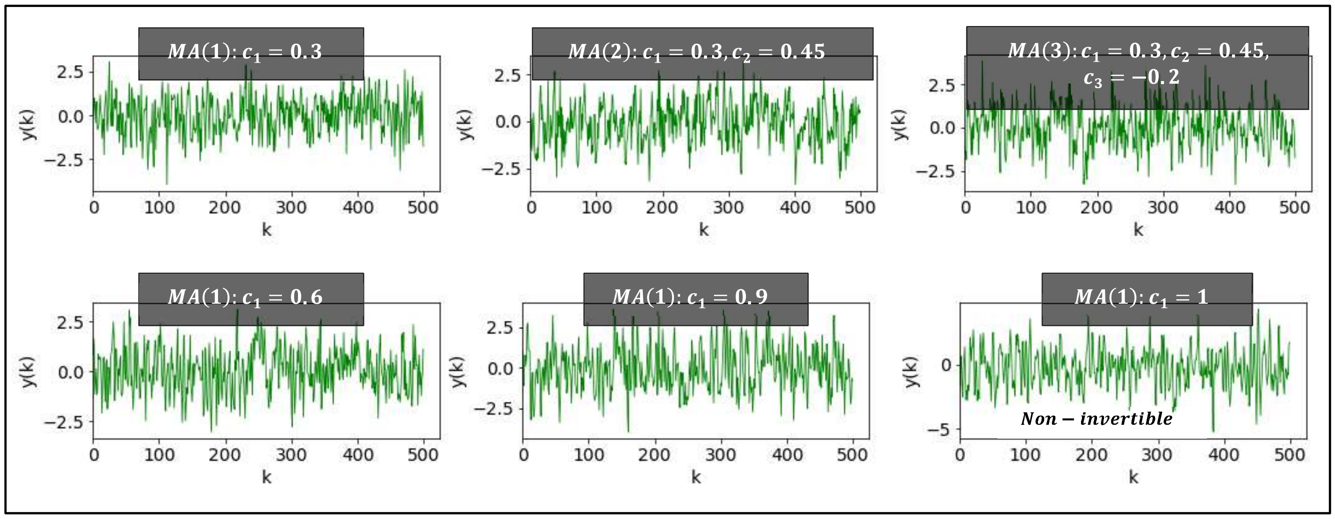
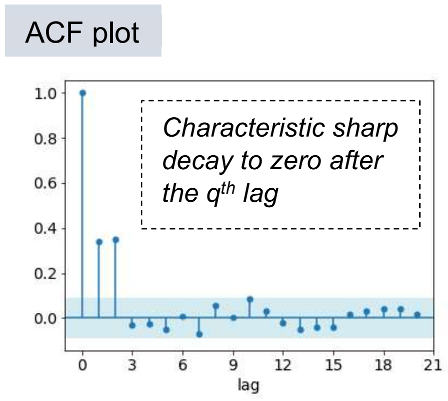
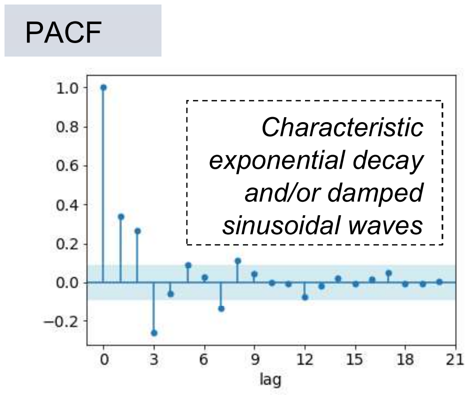


Figure 5.3: Time-series signals from MA processes

Model order selection

Just like for AR model, ACF and PACF plots provide direct clues about the order of the model as shown below for the MA(2) plot in Figure 5.3.

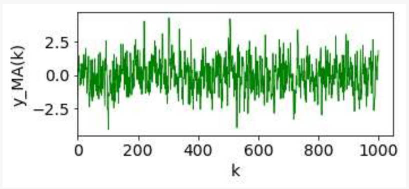




Let's concretize our understanding of the MA process through an example.

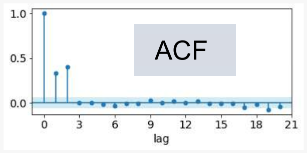
Example 5.2: We will create an MA(2) signal with 1000 samples. Thereafter, guided by ACF/PACF, a model will be fitted to the data followed by residual diagnostics. The imported packages are the same as that in Example 5.1

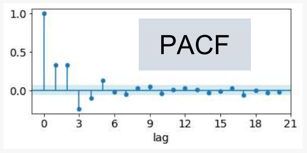
```
# generate data for MA(2) process using statsmodels.tsa package
 ma_coeffs = np.array([1, 0.3, 0.45]) # [1, c1, c2]
MAprocess = ArmaProcess(ma = ma_coeffs)
y_MA = MAprocess.generate_sample(nsample=1000)
```



`

```
# generate ACF and PACF plots for y_MA
conf_int = 2/np.sqrt(len(y_MA))
plot_acf(y_MA, lags= 20, alpha=None, title='')
plt.gca().axhspan(-conf_int, conf_int)

plot_pacf(y_MA, lags= 20, alpha=None, title='')
plt.gca().axhspan(-conf_int, conf_int)
```





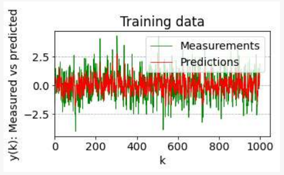
The ACF plot clearly indicates an MA(2) process. We will again use the *statsmodels.tsa* package to fit an MA(2) model.

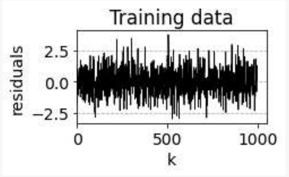
```
# Fit an MA(2) model
y_MA_centered = y_MA - np.mean(y_MA)
model = ARIMA(y_MA_centered, order=(0, 0, 2)) # order = (p,d,r)
results = model.fit()

# Print out the estimate for the parameters c1 and c2
print('[c1, c2] = ', results.maparams)

>>> [c1, c2] =  [0.30 0.53 ]
```

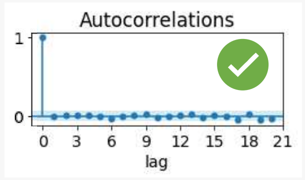
Estimation of MA parameters is not as easy as that for AR parameters because the unknowns, past shocks (*e*) and coefficients (*c*), are non-linearly present in the model. Therefore, iterative numerical procedure is employed for parameter estimation.

The estimated parameters are close to the true values (the inaccuracy in the estimates diminishes for larger sample size). With the obtained model, let’s generate the model residuals and confirm whiteness of the residual sequence.

```
# get model predictions and residuals on training dataset
y_MA_centered_pred = results.predict()
residuals = y_MA_centered - y_MA_centered_pred

plt.figure(), plt.title('Training data')
plt.plot(y_MA_centered, 'g', label='Measurements')
plt.plot(y_MA_centered_pred, 'r', label='Predictions')
plt.ylabel('y(k): Measured vs predicted'), plt.xlabel('k'), plt.legend()

plt.figure(), plt.plot(residuals, 'black', linewidth=0.8)
plt.title('Training data'), plt.ylabel('residuals'), plt.xlabel('k')

# ACF residuals
plot_acf(residuals, lags= 20, alpha=None, title='')
plt.gca().axhspan(-conf_int, conf_int, facecolor='lightblue', alpha=0.5)
plt.xlabel('lag'), plt.title('Autocorrelations')
```







`

Equivalence between AR and MA models

AR and MA models may appear to be very different, but, under some conditions, they are inter-convertible. Any stationary AR model can be rewritten as an MA(∞) model and any invertible (definition of invertibility will be provided soon) MA model can be equivalently rewritten as an AR(∞) model. For example, consider the MA(1) model $y(k) = c_1 e(k-1) + e(k)$, which can be rewritten as

$$\begin{aligned} y(k) &= c_1(y(k-1) - c_1 e(k-2)) + e(k) \\ &= c_1 y(k-1) - c_1^2 e(k-2) + e(k) \end{aligned}$$

$\vdots$ after repeated substitutions

AR(∞) model! $$y(k) = -\sum_{j=1}^{\infty}(-c_1)^j y(k-j) + e(k) \qquad \text{eq. 5}$$

An alternative and quicker derivation:

$$y(k) = (1 + c_1 q^{-1})e(k)$$
$$\Rightarrow (1 + c_1 q^{-1})^{-1} y(k) = e(k)$$
$$\Rightarrow \left(\textstyle\sum_{j=0}^{\infty}(-c_1)^j q^{-j}\right) y(k) = e(k)$$

using polynomial long division

Note that Eq. 5 represents a valid/sensible model only if $|c_1| < 1$ (otherwise c_1^j grows unbounded). This is also the condition of invertibility for a MA(1) model. An AR(1) model can be rewritten as MA(∞) model via recursive substitution of *y(k-1)*, *y(k-2)*, and so on.

ACF/PACF plots may sometime suggest both AR and MA models as adequate representations of the underlying process. Given the equivalence between the two class of models, you can choose either. Just keep a note that while AR parameters are easy to estimate, MA model may be more parsimonious (i.e., require much fewer parameters).

`

5.4 Autoregressive Moving Average (ARMA) Models: An Introduction

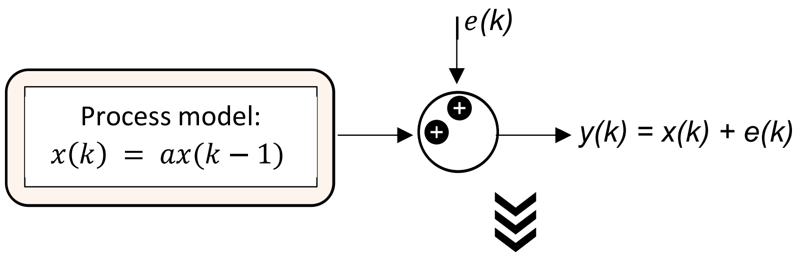
Sometimes a more compact and parsimonious model (compared to AR or MA models) can be built by combining the autoregressive and moving average terms. Such models are called ARMA models and take the following form

$$y(k) = -a_1 y(k-1) \cdots - a_p y(k-p) + c_1 e(k-1) \cdots + c_r e(k-r) + e(k)$$

$$= \underbrace{\left(-\sum_{j=1}^{p} a_j q^{-j}\right) y(k)}_{\text{AR terms}} + \underbrace{\left(\sum_{j=1}^{r} c_j q^{-j}\right) e(k)}_{\text{MA terms}} + e(k) \qquad \text{eq. 6}$$

The above model is called ARMA(p,r) due to the *p*th order autoregressive component and *r*th order moving average component. It is apparent that AR and MA models are special cases of ARMA model. Additionally, any stationary and invertible ARMA model can equivalently be rewritten an AR(∞) or a MA(∞) model.

ARMA models are not just for mathematical convenience but are commonly encountered in process industry. To see the reason, consider an example scenario below where output from a pure AR process is contaminated by a random white noise disturbance



$$y(k) = a\big(y(k-1) - e(k-1)\big) + e(k)$$

$$\Rightarrow y(k) = ay(k-1) - ae(k-1) + e(k)$$

Overall, an ARMA(1,1) process!

`

Plots below provide a sample of patterns generated from ARMA models.

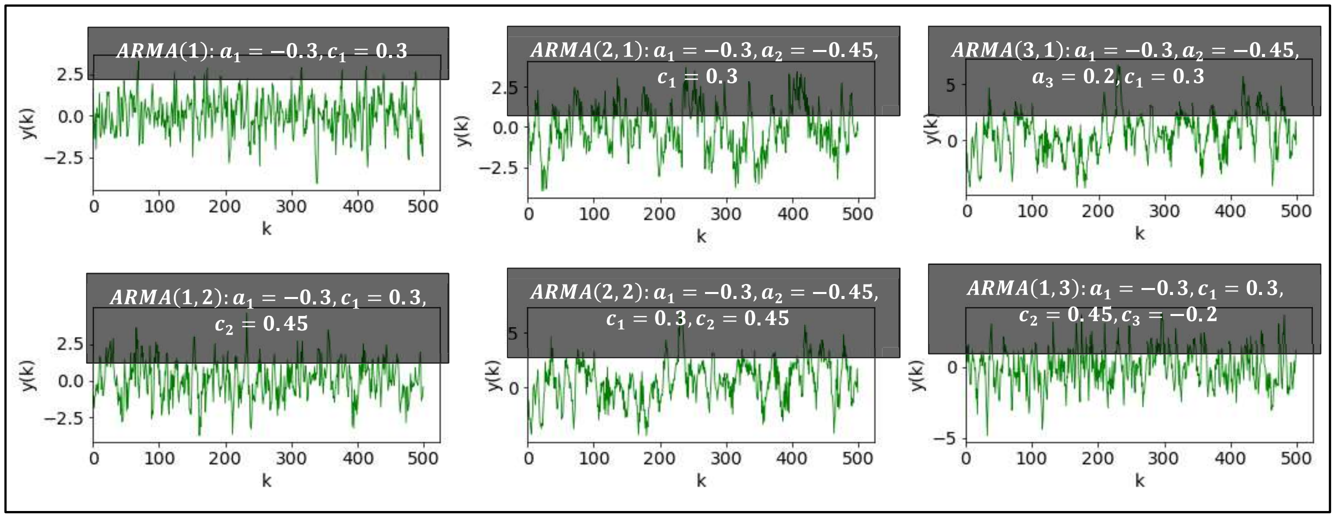
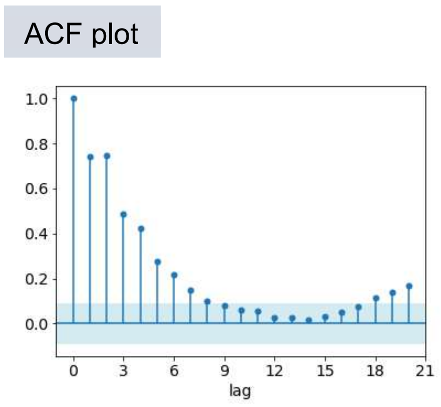
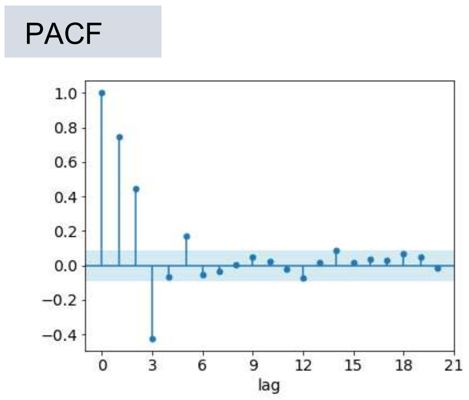


Figure 5.4: Time-series signals from ARMA processes

Model order selection

ACF and PACF plots for the ARMA(2,2) plot in Figure 5.5 are provided below.





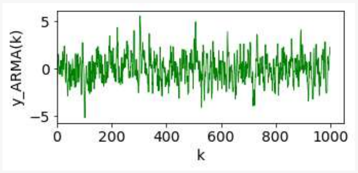
Both ACF and PACF plots tails off in exponential-like pattern (with mixture of damped sine waves) which is characteristic of ARMA process

As is apparent, not much help is provided regarding the specific values of *p* and *r*. The general recourse is to use model order selection techniques introduced in Chapter 4.

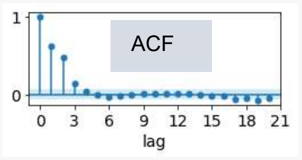
`

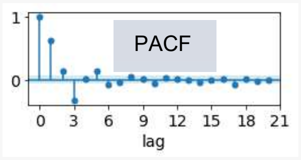
Example 5.3: We will create an ARMA(1,2) signal with 1000 samples. Thereafter, the AR and MA orders will be automatically determined via AIC and the corresponding model will be fitted to the data. The imported packages are the same as that in Example 5.1

```
# generate data for ARMA(1,2) process
ARMAprocess = ArmaProcess(ar=[1, -0.3], ma=[1, 0.3, 0.45])
y_ARMA = ARMAprocess.generate_sample(nsample=1000)
```



```
# generate ACF and PACF plots for y_ARMA
plot_acf(y_ARMA, lags= 20, alpha=None, title='')
plot_pacf(y_ARMA, lags= 20, alpha=None, title='')
```





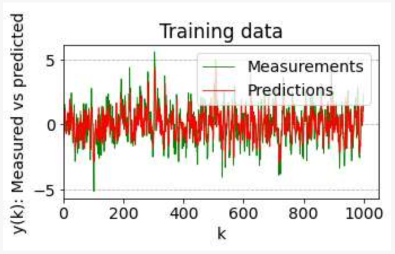
The ACF and PACF plots are not very conclusive. Fortunately, the *tsa* package provides a function arma_order_select_ic that can help estimate the orders via AIC.

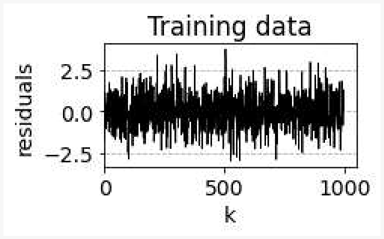
```
# Determine the optimal AR and MA orders
from statsmodels.tsa.stattools import arma_order_select_ic

y_ARMA_centered = y_ARMA - np.mean(y_ARMA)
res = arma_order_select_ic(y_ARMA_centered, max_ma=4, ic=["aic"])
p, r = res.aic_min_order
print('(p, r) = ', res.aic_min_order)

>>> (p, r) =  (1, 2)
```

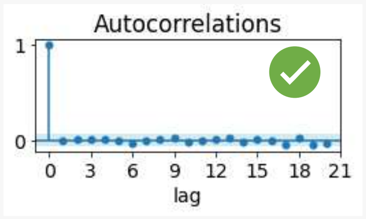
Orders correctly identified!

```
# Fit an ARMA(p,r) model
model = ARIMA(y_ARMA_centered, order=(p, 0, r))
results = model.fit()
print('[a1] = ', -results.arparams); print('[c1, c2] = ', results.maparams)

>>> [a1] =  [-0.29]
[c1, c2] =  [0.30 0.53]
```

The estimated parameters are close to the true values. Next, the residuals are generated as in the previous examples and whiteness of the residual sequence is confirmed.







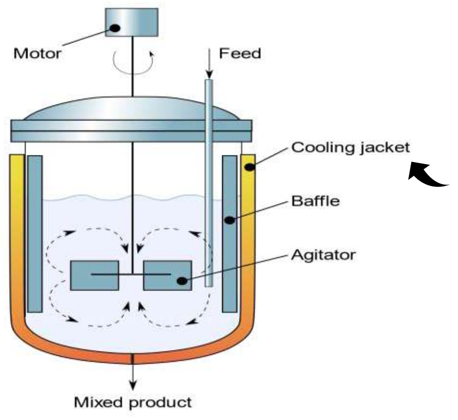
5.5 Monitoring Controlled Variables in a CSTR using ARMA Models

To illustrate a practical application of time series analysis, we will consider the continuous stirred-tank reactor described in the chapter introduction. Here, the product temperature will be monitored to ensure it is not deviating away from its optimal setpoint due to any process fault. The datafile *CSTR_controlledTemperature.csv* contains 4 hours of temperature measurements[40] sampled at 0.1 seconds (totaling 2401 samples). A process fault occurs at 3.5 hours (around sample # 2100) causing a 20% decrease in the heat transfer coefficient. Let's see if this process fault causes any deviations in the product temperature and if our monitoring methodology can detect these deviations.

Continuous Stirred Tank Reactor

Monitoring objective: *Catch unexpected deviations in product temperature as soon as possible as a leading indicator of process and/or controller fault*

Scenario: *A fault in the cooling jacket leads to a step change (decrease) in the heat transfer coefficient*



Cooling jacket temperature is used to control the reactor/product temperature at optimal set-point

Figure 5.5: Process fault detection using TSA: case-study set-up[41]

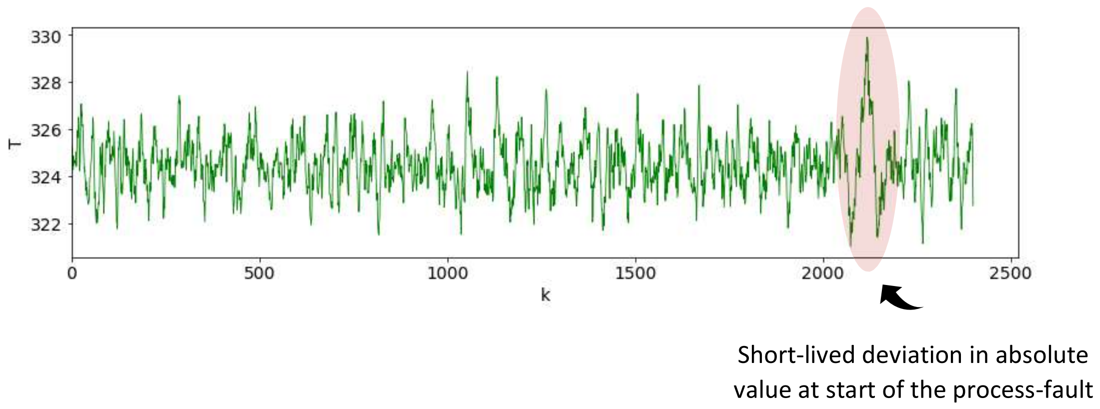
Let's start by importing the required packages and exploring the provided I/O dataset. The first 3 hours of data will be used to build a model and the last 1 hour will be used as test data.

```
# import packages
import matplotlib.pyplot as plt, numpy as np, control
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import arma_order_select_ic
from statsmodels.tsa.arima.model import ARIMA
```

[40] The CSTR model provided at https://github.com/APMonitor/pdc/blob/master/CSTR_Control.ipynb is used to generate the data.
[41] CSTR diagram created by Daniele Pugliesi under Creative Commons Attribution-Share Alike 3.0, https://commons.wikimedia.org/wiki/File:Agitated_vessel.svg

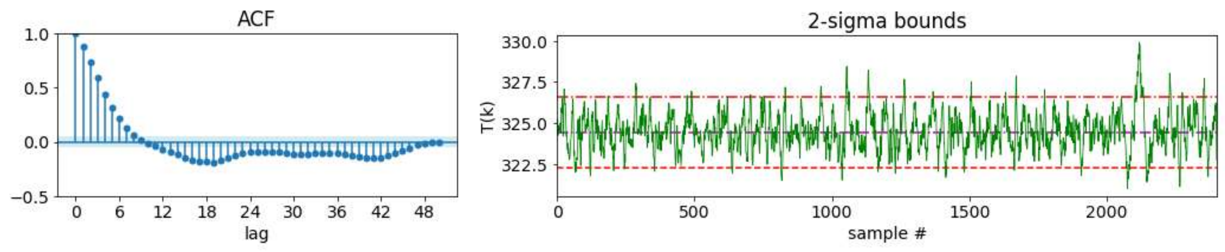
`

```
# read data and split into training and test data
T = np.loadtxt('CSTR_controlledTemperature.csv', delimiter=',')
T_train = T[:1800]; T_test = T[1800:]
```



The plot above shows that the temperature seems to show only a short-lived[42] deviation in its absolute value due to the process fault. Therefore, on the basis of this time-plot alone, plant operators could very well ignore this as an unimportant fluctuation and the process fault could go unattended.

The trivial univariate monitoring approach of using $mean \pm 2 * standard\ deviation$ bounds would also be inadequate because this approach assumes that the samples are independent of each-other, which as shown by the below ACF plot of *T_train* data, is certainly not true for the product outlet temperature (and dynamic processes in general). The control chart shown below shows the failure of this approach as the controlled variable does not show any sustained unusual violations of the control bounds (bounds are computed using the training data only).



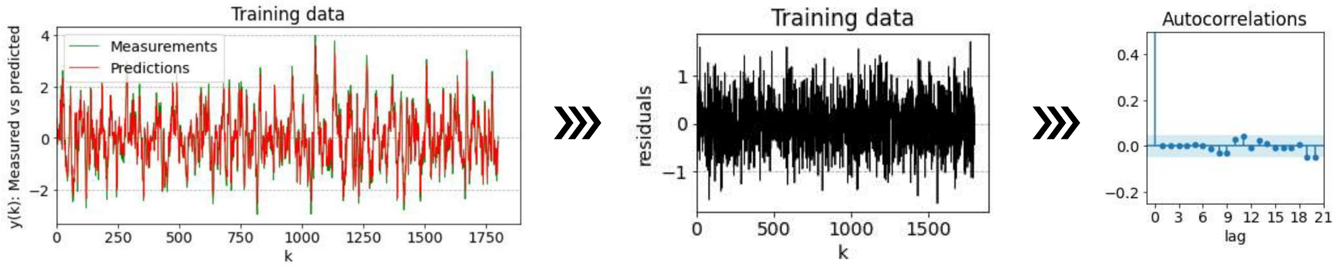
Let’s now see if the time-series models can do a better job of fault detection. We will attempt to fit an ARMA model.

[42] The controller adjusts the jacket temperature to compensate for the reduction in the heat transfer coefficient.

`

```
# Determine the optimal AR and MA orders
T_train_centered = T_train - np.mean(T_train); T_test_centered = T_test - np.mean(T_train)
res = arma_order_select_ic(T_train_centered, max_ar=5, max_ma=5, ic=["aic"])
p, r = res.aic_min_order
print('(p, r) = ', res.aic_min_order)

>>> (p, r) =  (2, 4)

# Fit an ARMA(p,r) model and get residuals
model = ARIMA(T_train_centered, order=(p, 0, r))
results = model.fit()
T_train_centered_pred = results.fittedvalues # same as results.predict()
residuals_train = T_train_centered - T_train_centered_pred # same as results.resid
```

The residuals pass the model quality check.



It is time now to use the fitted ARMA model for fault detection. The rationale is straight-forward: if any process-fault leads to changes in process behavior then the model fitted using 'normal' operating condition data won't be very accurate with faulty data and therefore, the residuals will show higher values. Consequently, the residuals are monitored for fault detection as summarized in Figure 5.6.

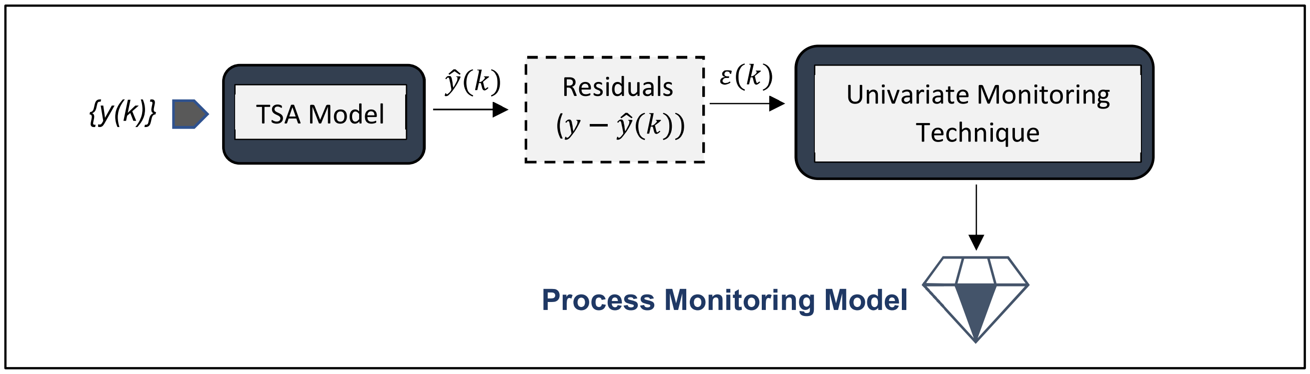
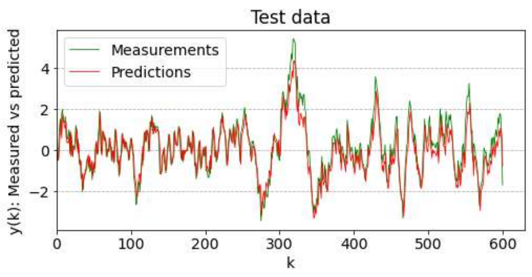
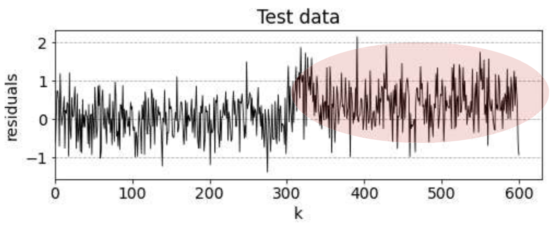


Figure 5.6: Process monitoring methodology via monitoring of time-series analysis-based residuals.

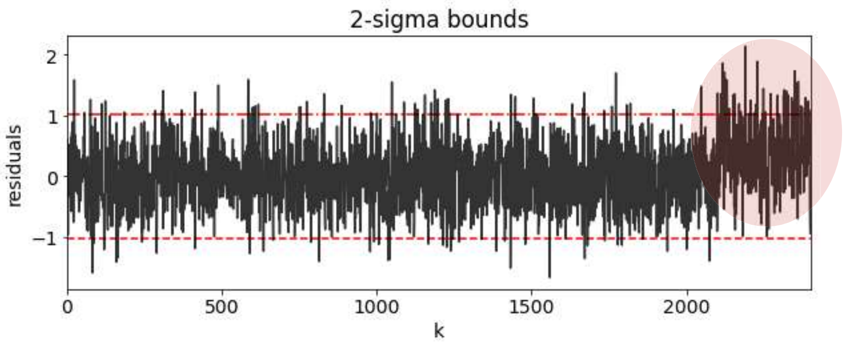
`

```
# 1-step ahead predictions for test data
results_test = results.apply(T_test_centered)
T_test_centered_pred = results_test.fittedvalues # same as results_test.predict()
residuals_test = T_test_centered - T_test_centered_pred
```





The above plots show that the residuals exhibit an increase in the mean value after the onset of process fault (after sample # 300 for test dataset). The control chart for the whole dataset makes the sustained unusual values of the residuals more evident. The fault diagnosis exercise can now be carried out to isolate the root cause of the fault.



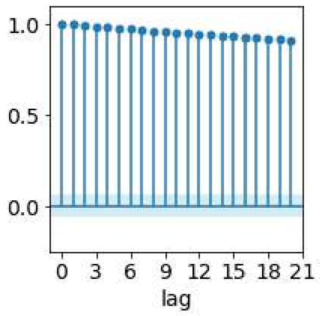
Fault detection via monitoring of model parameter estimates

We saw in the above example how process faults lead to change in the behavior of process variables. Therefore, as alternative to residual monitoring, another popular approach for process fault detection is to build new dynamic models for the process at regular intervals and check for significant changes in model parameters. Significant parameter changes can be good indicators of process fault.

Stationarity revisited and introduction to invertibility

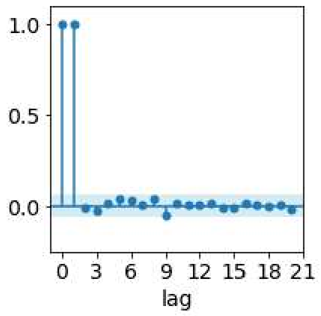
In Chapter 4, we had seen how homogeneous non-stationary signals can be made stationary by differencing. However, it is not always easy to infer homogeneous non-stationarity from simple time-plots. Fortunately, ACF, PACF, and PSD plots can provide crucial clues. The characteristic indicators of non-stationary behavior for these graphical plots are provided below. You should be easily able to explain the rationale behind these characteristics based on the concepts we have covered so far.

ACF plot



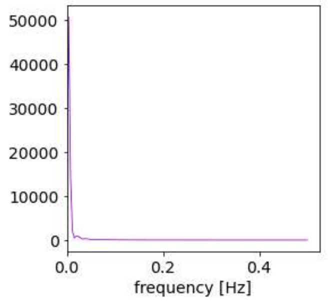
Slowly decaying/nearly constant

PACF plot



Near-unity value at lag 1

PSD



Significant power at low frequencies

Checking stationarity using noise transfer operator *H(q⁻¹)*

Transfer operator for the ARMA model in Eq. 6 can be written as

$$H(q^{-1}) = \frac{1 + \sum_{j=1}^{r} c_j q^{-j}}{1 + \sum_{j=1}^{p} a_j q^{-j}} = \frac{C(q^{-1})}{A(q^{-1})}$$

numerator polynomial in q^{-1}

denominator polynomial in q^{-1}

An ARMA model represents a stationary process if the absolute of the roots of $A(q^{-1})$ is greater than 1. For example, for the AR(1) process *y(k) = ay(k-1) + e(k)*,

$$A(q^{-1}) = 1 - aq^{-1} \;\Rightarrow root = \frac{1}{a}$$

$$\Rightarrow stationarity\ if \left|\frac{1}{a}\right| > 1 \;\Rightarrow |a| < 1$$

In SysID literature, roots of $A(q^{-1})$ are called poles. Note that if degree of $A(q^{-1})$ or the order of autoregressive part is > 1, then poles can be complex numbers. Therefore, 'poles outside the unit circle' is the correct way of stating the condition of stationarity.

`

Poles inside the unit circle lead to explosive non-stationarity and poles on the unit circle causes homogeneous non-stationarity (such as random walk process). MA processes have $A(q^{-1}) = 1$ and are always stationary (for finite *r* and *c*$_j$).

<u>Invertibility</u>

Another condition imposed on disturbance model to represent realistic processes is invertibility. The condition requires that the roots of the numerator polynomial ($C(q^{-1})$), also called zeros of the transfer operator, lie outside the unit circle. AR models have $C(q^{-1}) = 1$ and therefore, are always invertible. Without going into mathematical details, it suffices to know that for non-invertible models, it is possible to write *e(k)* as a summation of future values of *y(k)*; such models are called non-causal models and do not represent realistic processes.

5.6 Autoregressive Integrated Moving Average (ARIMA) Models: An Introduction

In Chapter 4 we saw that non-stationary signals may be modeled via stationary ARMA models after suitable differencing. Such difference-stationary processes are formalized as ARIMA processes[43] and are represented as

$$\underbrace{(1-q^{-1})^d y(k)}_{\substack{\text{Integrating part} \\ \text{with } d \text{ differences}}} = \frac{C(q^{-1})}{A(q^{-1})} e(k) = \underbrace{\frac{1+\sum_{j=1}^{r} c_j q^{-j}}{1+\sum_{j=1}^{p} a_j q^{-j}}\, e(k)}_{\text{ARMA part}}$$

$$\Downarrow$$

$$z(k) = \frac{C(q^{-1})}{A(q^{-1})} e(k) \qquad \text{where } \underbrace{z(k)}_{\text{differenced signal}} = (1-q^{-1})^d y(k)$$

[43] Sometimes, modelers include a constant term θ_o in ARIMA model which then takes the following form $(1-q^{-1})^d y(k) = \theta_o + \frac{C(q^{-1})}{A(q^{-1})} e(k)$. In Chapter 4, we had shown that a deterministic trend of degree *d* can be removed with differencing *d* times. Therefore, the inclusion of θ_o in the model is equivalent to explicitly including a deterministic trend in the model.

For example, consider a signal that becomes ARMA(1,1) stationary after first differencing

$$(1-q^{-1})y(k) = \frac{1-cq^{-1}}{1-aq^{-1}}e(k) \longleftarrow \text{transfer operator form}$$

$$\Rightarrow (1-aq^{-1})(y(k)-y(k-1)) = e(k) - ce(k-1)$$

$$\Rightarrow y(k) = (1+a)y(k-1) - ay(k-2) - ce(k-1) + e(k) \longleftarrow \text{difference equation form}$$

It should have occurred to you by now that *q* notation makes it much easier to represent and work with complicated models.

The above model is called ARIMA(p,d,r) due to the *d^th^* degree of differencing and ARMA(p,r) modeling. You can see that with suitable choice of *p, d*, and *r*, MA, AR, and ARMA models become special cases of ARIMA model.

Random walk model

An ARIMA(p,1,r) disturbance model is equivalent to saying that there are random (and temporally correlated) step changes occurring in the disturbance signal. A pure integrating model of order 1 (ARIMA(0,1,0), a.k.a. random walk model) takes the following form

$$(1-q^{-1})y(k) = e(k) \Rightarrow y(k) = y(k-1) + e(k)$$

The integrating model is called as such because the noise values are continuously added to the output signal.

Model order selection

The graphical tools can be used to check if differencing is needed. If the differenced series shows signs of AR or MA or ARMA, then no further differencing is required. Very rarely, you would need to choose *p, d*, or *r* greater than 2.

Unit-root tests

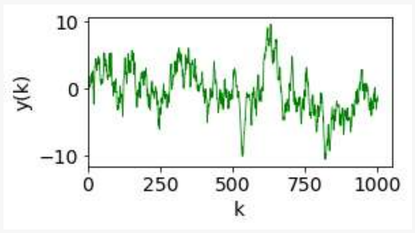
A rigorous way of checking the presence of unit pole (and therefore the need of differencing) is unit-root test. To understand this test, consider a simple AR(1) process *y(k) = ay(k-1)* + *e(k)*. To warrant differencing, '*a*' needs to be equal to 1 and therefore, in the regression model *z(k) = (a-1)y(k-1)* + *e(k)* where *z(k) = y(k) – y(k-1)*, the coefficient *a-1* would equal 0. The unit-root test is simply a hypothesis test on this coefficient being equal to zero. This is called Dickey-Fuller test.

If you have some experience with hypothesis testing, then you would know that if the *p* value of the test is above a specified threshold/significance level[44], then the (null) hypothesis (on the series under study being non-stationary) is not rejected. For more generic series of integrating order > 1, a more generic augmented Dickey-Fuller test is used. However, the test process remains the same.

Example 5.4:

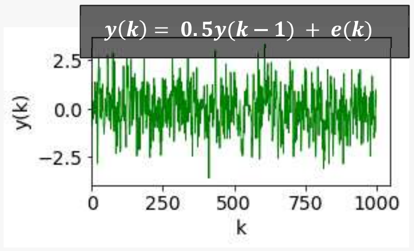
Generate data from AR process and perform ADF test to check for unit root.

```
# import packages
import numpy as np, matplotlib.pyplot as plt
from statsmodels.tsa.arima_process import ArmaProcess

# generate data
ar_coeffs = np.array([1, -0.96]) # y(k) = 0.96y(k-1) + e(k)
ARprocess = ArmaProcess(ar=ar_coeffs)
y = ARprocess.generate_sample(nsample=1000)

# perform augmented Dickey-Fuller test
from statsmodels.tsa.stattools import adfuller
result = adfuller(y)
print('p-value: %f' % result[1])
```

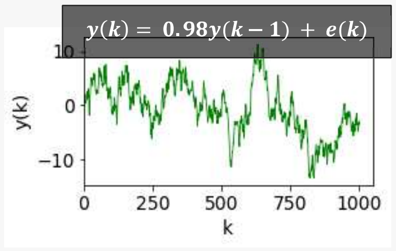


```
>>> p-value: 0.000187
```

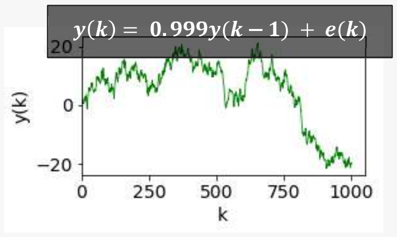
Since the p-value is less than 0.05, the null hypothesis is rejected $\Rightarrow$ the time series is stationary



p-value: 0



p-value: 0.027



p-value: 0.904

[44] The threshold is commonly set to 5% or 0.05

`

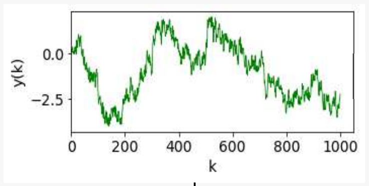
Example 5.5:

In this example we will utilize a given a time series generated from the following process. With guidance from the graphical plots and ADF test, we will infer the correct model orders. Thereafter, an ARIMA model will be fitted to find the model parameters.

$$(1 - q^{-1})y(k) = \frac{1 - 0.7q^{-1}}{1 - 0.5q^{-1}}e(k)$$

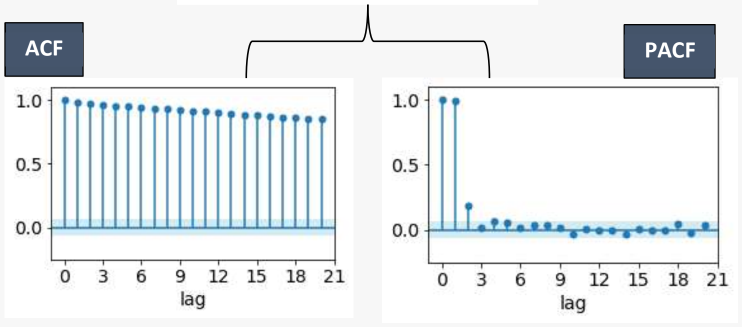
Note that packages like *pmdarima* exist which can be used to automatically estimate the order of differencing and the AR/MA orders.

```
# import packages and read data
import numpy as np, matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
y = np.loadtxt('ARIMA_data.txt')
```

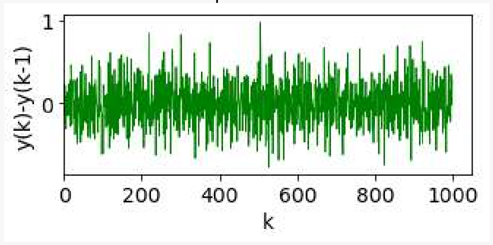


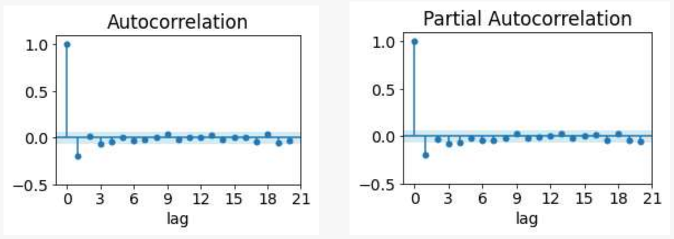
p-value: 0.325

The correlation plots and the ADF test clearly indicate the presence of unit-root. Therefore, we know that differencing is required atleast once. Let's check the ACF and PACF plots of the difference signal.



p-value: 0





The ADF test indicates that the differenced signal is stationary and therefore no further differencing is required. Moreover, the ACF and PACF plots clearly indicate the AR and MA orders to be 1 (both plots show sharp decline in value after lag 1). We can therefore fit an ARIMA(1,1,1) model to the given data.

```
# fit ARIMA(1,1,1) model
y_centered = y - np.mean(y)
model = ARIMA(y_centered, order=(1, 1, 1))
results = model.fit()
print(results.summary())
```

```
=====================================
                    coef    std err
-------------------------------------
ar.L1             0.4883      0.096
ma.L1            -0.6723      0.082
```

Parameter estimates are close to actual

5.7 Forecasting Signals using ARIMA

Previously we saw how time series models can be used to build process monitoring solutions. As alluded to before, another common use of such models is in forecasting of stochastic signals. Although the *Statsmodels* package provides native support for generating forecasts, it would be instructive to see what the forecasting work process actually entails. We will use a simple ARIMA(1,1,1) model for this illustration. Let the model be the following

$$(1 - q^{-1})y(k) = \frac{1 - 0.7q^{-1}}{1 - 0.5q^{-1}}e(k)$$

Let's expand the above into a difference equation with only *y(k)* on the L.H.S.

$$y(k) = 1.5y(k-1) - 0.5y(k-2) - 0.7e(k-1) + e(k) \qquad \text{eq. 7}$$

To derive the expression for the forecasts from Eq. 7, the following notation/rules apply:

- $\hat{y}(k+l|k) \equiv \hat{y}(l)$ denotes the *l*th step-ahead forecast/prediction using all information available at time *k*
- Any *y* on R.H.S. of Eq. 7 that has not yet occurred is replaced with its forecast y
- Any future noise term is replaced with zero and past noise term with the 1-step ahead residuals (for example, $e(k-1) = y(k-1) - \hat{y}(k-1|k-2)$)

Let's see how the forecast expressions look like for $l = 1, 2, 3, \ldots$

$$\hat{y}(1) = 1.5y(k) - 0.5y(k-1) - 0.7e(k)$$

to be replaced with $y(k) - \hat{y}(k|k-1)$

$$\hat{y}(2) = 1.5\hat{y}(1) - 0.5y(k)$$

$$\hat{y}(3) = 1.5\hat{y}(2) - 0.5\hat{y}(1)$$

$$\vdots$$

both error terms in Eq. 7 replaced with zero

A somewhat incomplete aspect in the above forecast expressions is computation of *e(k)*. The annotation text suggests making use of $\hat{y}(k|k-1)$, but it would involve the term *e(k-1)*. This leads to a recursive approach to find the past noise values. The established approach is to start from time *p* + *d*, set estimated noise values for times ≤ p + d to zero and then recursively estimate the noise values until time *k* using the forecast expression for the 1-step ahead forecast. For our ARIMA(1,1,1) model, the computation would look like the following:

`

$$\hat{y}(2|1) = 1.5y(1) - 0.5y(0) - \overset{\text{(approximation)}}{\cancelto{0}{0.7}}e(1) + \overset{\text{(replacement with conditional mean)}}{\cancelto{0}{e}}(2)$$

$$\Downarrow$$

$$e(2) = y(2) - \hat{y}(2)$$

$$\Downarrow$$

$$\hat{y}(3|2) = 1.5y(2) - 0.5y(1) - 0.7e(2) + \cancelto{0}{e}(3)$$

$$\Downarrow$$

$$e(3) = y(3) - \hat{y}(3)$$

$$\vdots$$

This completes our quick overview of time-series analysis and its applications. Hopefully, you have gained an in-depth understanding of the concepts behind time-series analysis and can appreciate its usefulness in describing process systems.

Summary

In this chapter we looked at the four common models used to describe time series, namely, AR, MA, ARMA, and ARIMA models. We studied their model forms and procedures behind the corresponding model order selection. We looked at how ACF and PACF tools provide useful clues regarding model structure. We also looked into more details the stationarity requirements of TSA models and ways of assessing stationarity of signals. We will build upon the concepts covered in this chapter and look at models used to describe input-output dynamic signals in the next chapter.