1 Module 10: Time Series and Forecasting

Contents

1.1 The Forecasting Problem

Given some existing time-series data, try to forecast trends which will occur in the future. The modelling may be context-agnostic (i.e. 100 yrs of data collected yearly == 100 seconds of data collected per-second)

Methodology:

- 1. Collect historical data
- 2. Train a model
- 3. Use the model to make a forecast
- 4. Evaluate the performance of the model on updated data

Notation:

- Value at time t: y_t
- Historical data: $y_{t-h:t} = [y_{t-h+1}, y_{t-h+2}, \ldots, y_t]$
- Forecast: $\hat{y}_{t:t+f} = [\hat{y}_{t+1}, \hat{y}_{t+2}, \ldots, \hat{y}_{t+f}]$
- Future data: $y_{t:t+f} = [y_{t+1}, y_{t+2}, \ldots, y_{t+f}]$
- Forecast error: $e_t = y_{t:t+f} - \hat{y}_{t:t+f}$

e_t is an array, so we reduce it to a minimizeable number:

- MAE: $||e_t||_1 = \sum_{\tau=t+1}^{t+f} |e(\tau)|$
- RMSE: $||e_t||_2 = \sqrt{\frac{1}{f} \sum_{\tau=t+1}^{t+f} e(\tau)^2}$

1.2 The Stochastic Process

A sequence of random variables.

A time series is a single sample of a stochastic process.

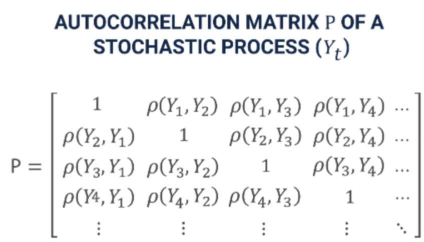
$$(Y_t)_{1:T} = (Y_1, Y_2, \ldots, Y_T)$$

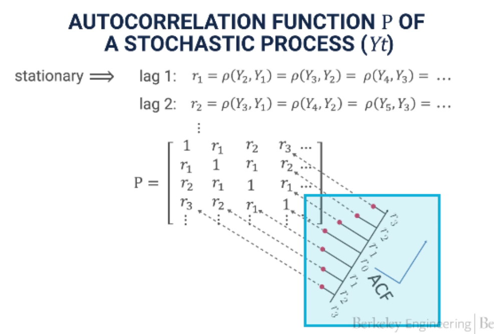
for T the length of the stochastic process.

Stationary process: a process' statistical properties remain constant over time (mean, variance, etc.), regardless of where you put the window of time or its scale.

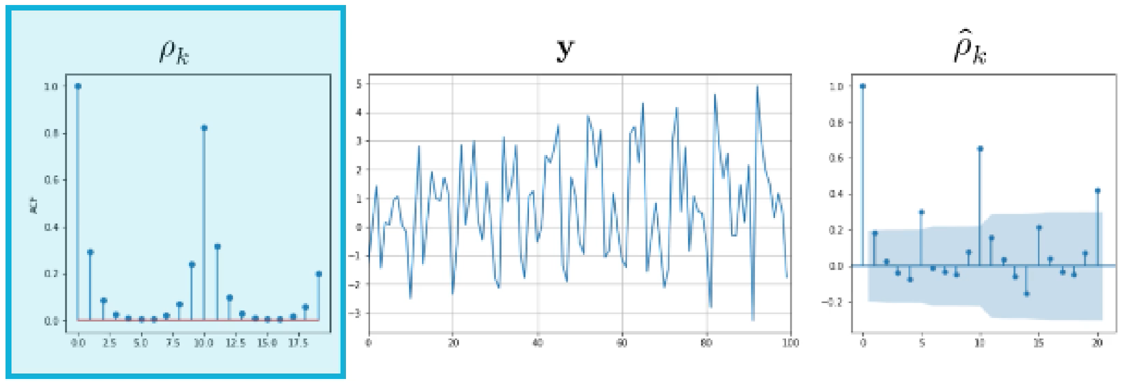
Independent process: when all its constituent random variables (Y_t) are mutually independent, i.e. $p(Y_1, Y_2, \ldots, Y_T) = \prod_{t=1}^{T} p(Y_t)$ (NOTE that the values of the past are irrelevant to the current or future values.)

IID (Independent and Identically Distributed): a process both stationary and independent e.g. the Gaussian white noise process.





In a stationary process, the diagonals will each represent a different timestep $r1, r2, \ldots \rightarrow$ "lags" $\rightarrow$ autocorrelation function (ACF)

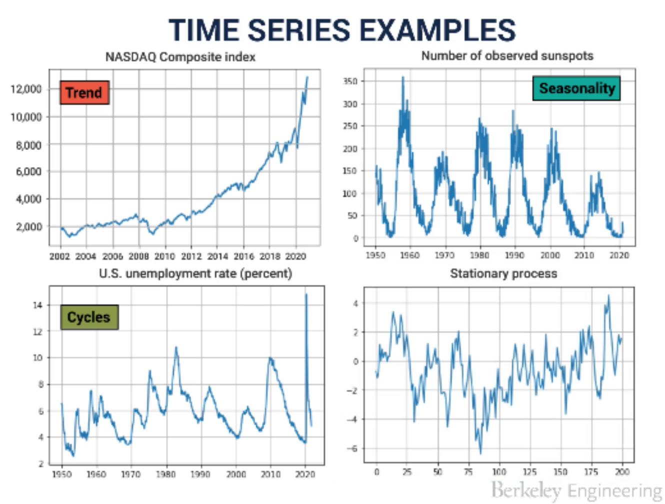


Using the ACF, one can pick out the time-dependent correlations e.g. the above with a noticeable lag-spike at 10, corresponding to the period of the wave

1.3 Autocorrelations using statsmodel

statsmodel: Shares some functionality with sklearn, includes time-series analysis.

```
from statsmodel.ts.arima_process as arima_process
process = arima_process.ArmaProcess( ar = [1, -.8], ma = [1] )
z = process.generate_sample( n_sample = 100 )
acf = process.acf( lags = 20 ) # Autocorrelation Function

import statsmodel.graphics.tsaplots as tsaplots
fig, ax = plt.subplots()
tsaplots.plot_acf( z, lags = 20, ax = ax )
```



There may be a general trend, a periodicity and a cyclicality. We encapsulate this by using a model:

$$y_{t-h:t} = t + c + s + r$$

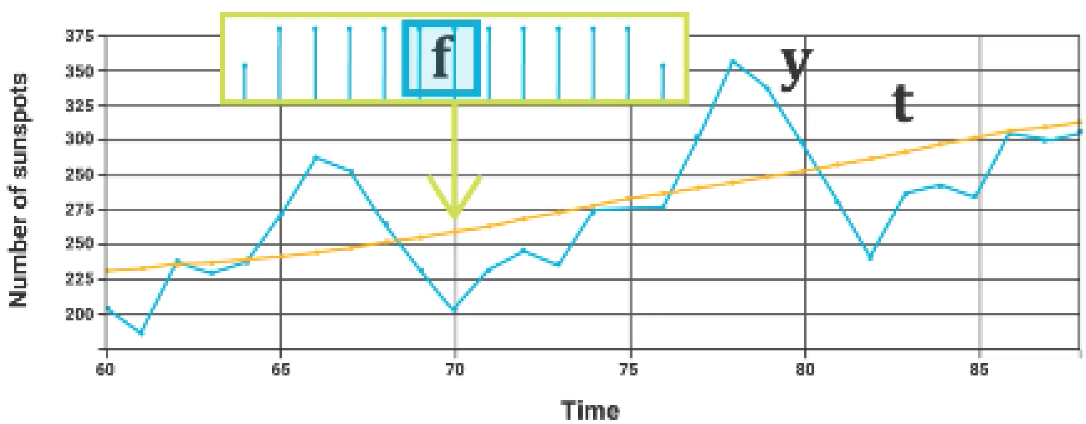
- Trend (t): long-term behavior
- Cycles (c): random low-frequency variations
- Seasonality (s): known periodicity
- Residue (r): everything else

If y characterizes the trend, cycles and seasonality well, then we would expect the residue to behave like a stationary process. NOTE: Sometimes you may need to multiply terms e.g. trend by seasonality, depending on if the behaviors scale this way.

Time Series decomposition

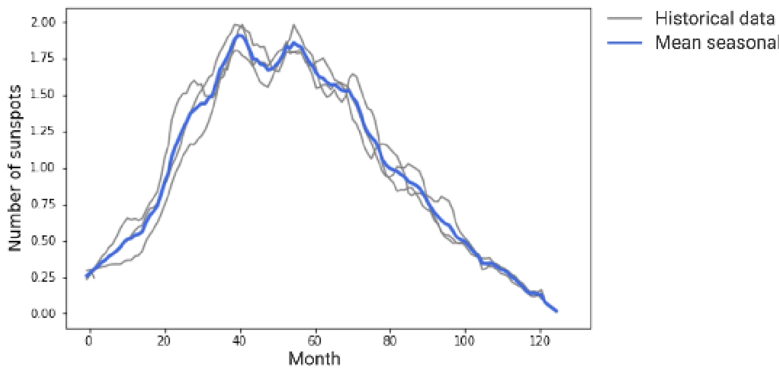
Trend

$$t = \text{conv}(y, f) \to t_t = f_0 y_{t-6} + f_1 y_{t-5} + \ldots + f_{12} y_{t+6}$$



Seasonality

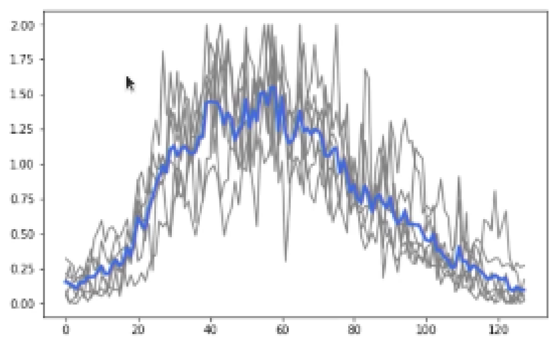
Chop up the seasonality by period and overlay the data. Filter the high-frequency noise & calculate the mean of the data.



1.4 Programming the Time Series Decomposition

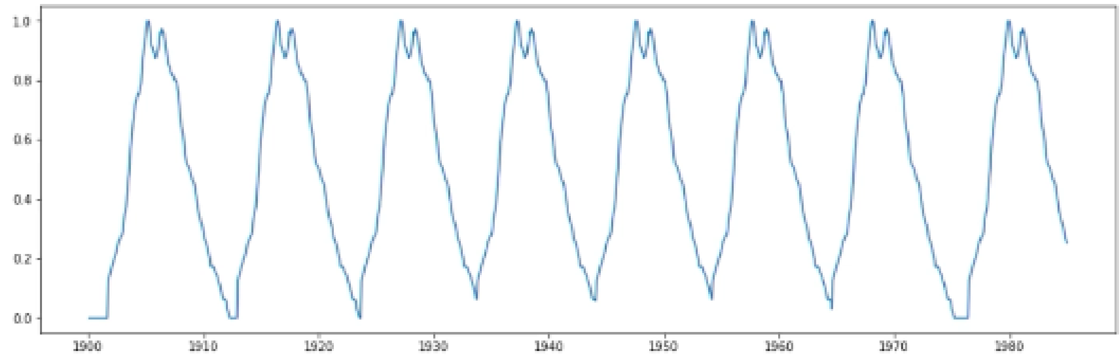
```
from statsmodels.tsa.seasonal import _extrapolate_trend
from statsmodels.tsa.filters.filtertools import convolutional_filter

# Split up data into historical time-series and future
y_historical, y_future

# We need first to extract the trend.

# Smooth the historical data with a filter:
# e.g. with sunspot data w/ known period 128 days
period = 128
filt = np.ones(period + 1)
filt[0] = .5
filt[1] = .5
# To ensure filter does not affect mean of historical data
filt /= period
# sum(filt) == 1

trend = convolution_filter(y_historical, filt)
# Ensures data will reach the bounds of the historical data
trend = extrapolate_trend(trend, period + 1)

# Having determined trend, now we detrend the data (remove the trend's effect)
detrended = y_historical - trend

# Next, we focus on seasonality.

# Identify first the indices of the minima of the detrended data
# This may need to be done manually ala
lows_index = [,]
lows = y_historical.index[lows_index]
```

```
# Can plot using plt.axvline

# Still need to set the period to a fixed value
period = np.round(np.mean(np.diff(low_index)))
n_seasons = len(lows)-1 # Number of seasons

# Instantiate for the 2D array
seasons = np.empty((period, n_seasons))
# Assemble stack of seasonal data
for p in range(num_seasons):
    s = detrended[lows_index[p]:lows_index[p]*period]
    s = 2*(s - np.min(s))/(np.max(s) - np.min(s))
    seasons[:,p] = s
mean_seasons = seasons.mean(axis=1)
```

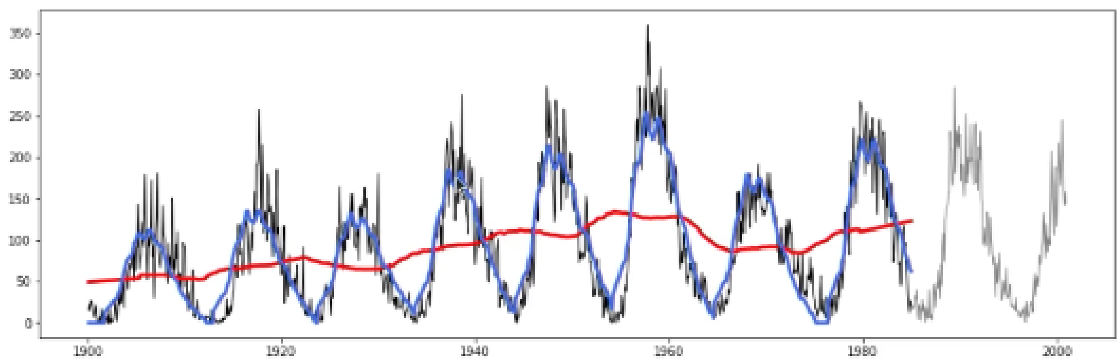


```
# Smooth seasonal data by creating a filter \& applying
filt_size = 9
filt = np.repeat(1. / filt_size, filt_size)

for p in range(n_seasons):
    s = seasons[:,p]
    s = convolution_filter(s, filt)
    s = extrapolate_trend(s, filt_size)
    s = 2*(s - np.min(s))/(np.max(s) - np.min(s))
    seasonals[:,p] = s
# Make sure to remove outliers after this step.
# Recalculate mean
mean_seasons = seasons.mean(axis=1)

# Build the seasonality template
# Instantiate frame with correct indices (set data to all zeros)
seasonality = pd.Series(index = y_historical.index, data = 0)
for low in lows_index:
    if low_period<len(seasonality):
```

```
        seasonality[low:low+period] = mean_seasons
    else:
        seasonality[low:] = mean_seasons[:len(seasonality) - (low+period)]
# Renormalize to 1
seasonality = seasonality/np.max(seasonality)
```




```
# Final model
model = 2 * trend * seasonality
```




```
# Compute residue to view whether remaining data is stationary
residue = y_historical - model
# View the ACF
tsaplots.plot_acf(residue, lags=20, ax=ax)

# Forecasting.
# Create trend
yhat_trend = pd.Series(index = y_future.index, data = trend[-1])
# Create seasonality
yhat_seasonality = pd.Series(index = y_future.index)
for i in range(len(yhat_seasonality)):
```

```
    yhat_seasonality[i] = seasonality[-(2*len(mean_seasonality) + 1)]

# Compute forecast
yhat = 2*yhat_trend*yhat_seasonality

# Compute prediction error
pred_error = y_future - yhat
```

Statsmodels does have a fcn. which allows you to naively do this all-in-one:

```
from statsmode.tsa.seasonal import seasonal_decompose

seasonal_decompose(
    x = y_historical,
    model = 'additive' or 'multiplicative',
    period = int
).plot()
```

1.5 The ARMA Framework

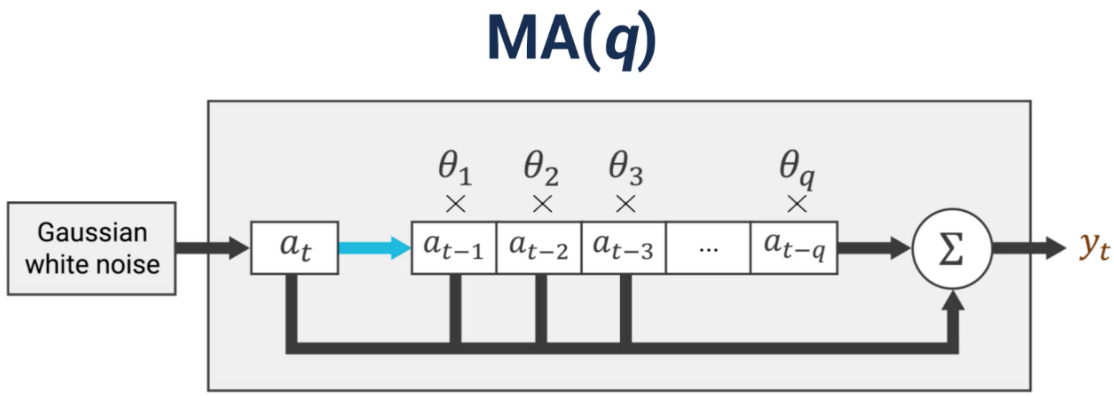
Designed to capture the time-invariant structure of stationary time series.

- Autoregression AR(p): model based on observations that are correlated with lagged observations
- Integrated: term indicating that raw observations have been differentiated to make the time series stationary
- Moving Average MA(q): model based on the dependence between observation & residual error after applying a moving average model to lagged observations

where p and q are the orders of the AR and MA processes.

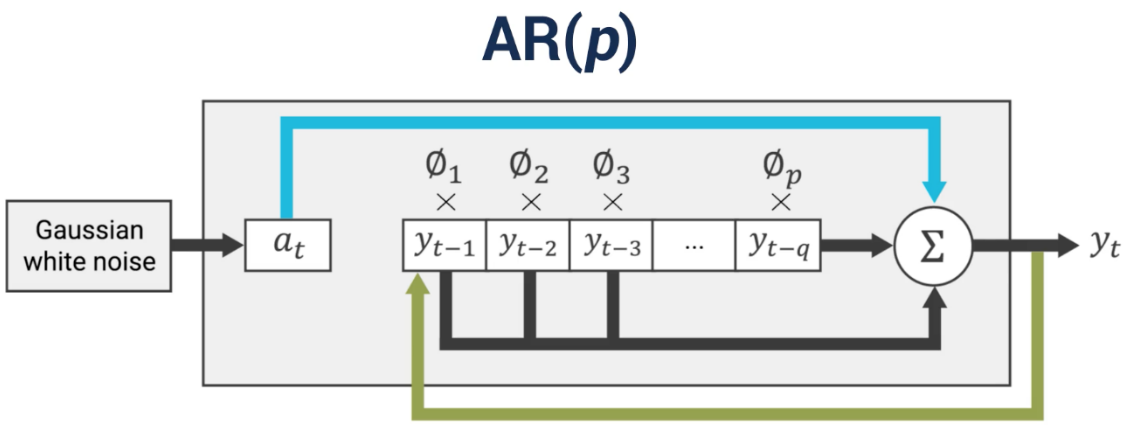
For moving average, the process assumes that an output y_t is fed by gaussian white noise (a_t).

$$\mathbf{MA}(q)\text{: } y_t = a_t + \sum_{j=1}^{q} \theta_j a_{t-j}$$



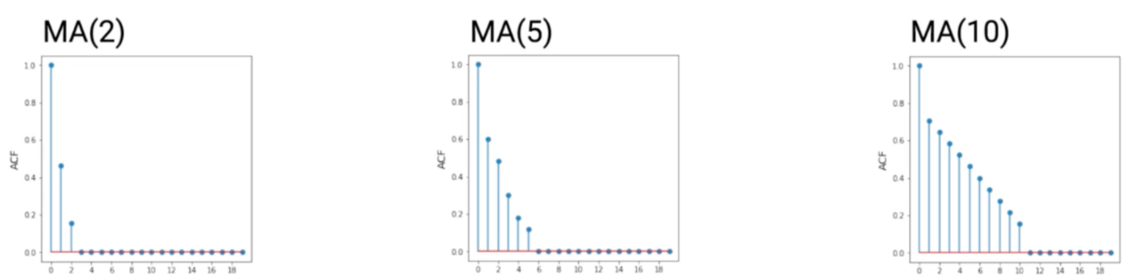
With AR, we feed the output y_t and feed it back on itself.

$$\textbf{AR}(p)\text{: } y_t = a_t + \sum_{j=1}^{p} \phi_j y_{t-j}$$

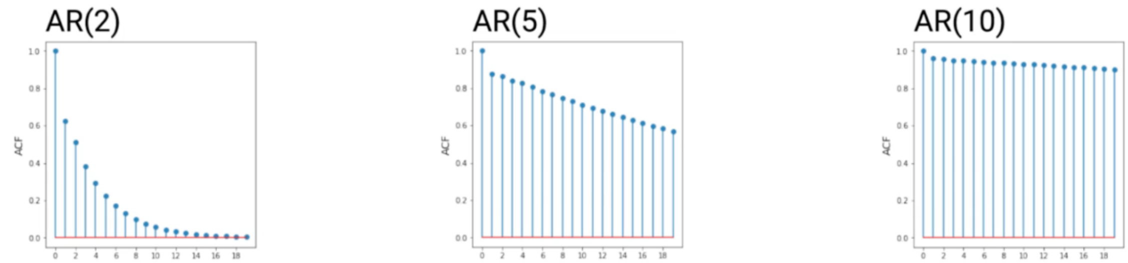


Determining the order of MA or AR

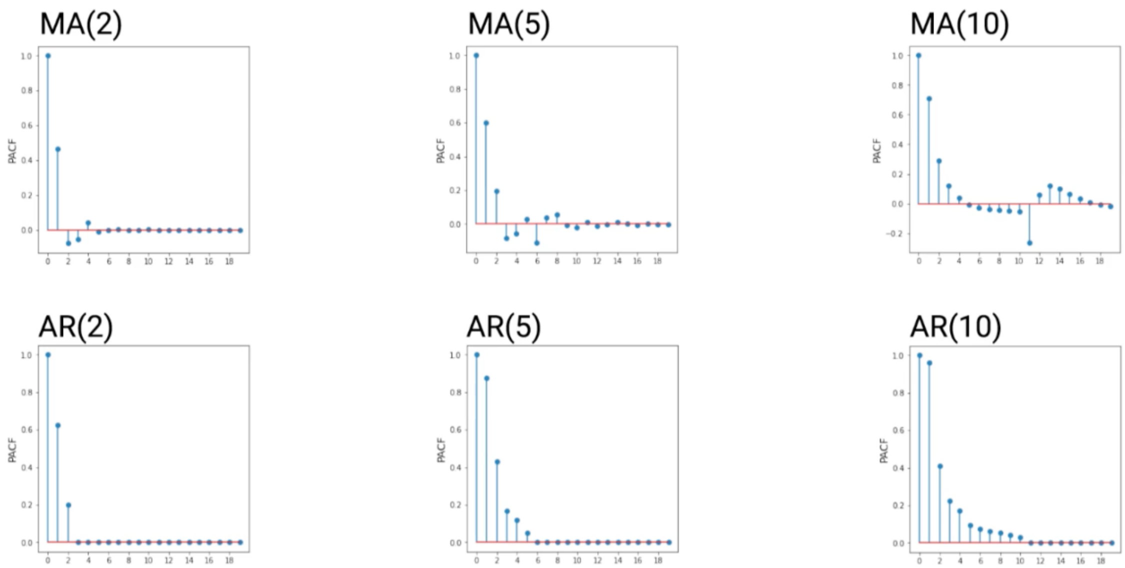
With MA, the ACF will have as many non-zero entries (aside from the first) as the order.



This is not true for AR.



However, with the Partial Autocorrelation Function (PACF), the opposite will be true. The AR order will have a corresponding number of nonzero entries.



Combine moving average and autoregression to create ARMA.

$$\textbf{ARMA(p,q)}\text{: } y_t - \sum_{j=1}^{p} \phi_j y_{t-j} = a_t + \sum_{j=1}^{q} \theta_j a_{t-j} \tag{1}$$

Using ARMA:

- Check that the signal is stationary
- Use sample autocorrelation fcn. (SAFC) and sample partial autocorrelation fcn. (SPACF) to select p and q
- Compute θ and ϕ coefficients of MA(q) and AR(p)

- Compute the residuals (check that is is == white noise)
- Make the forecast