Time series analysis: AR-MA

ΕΠΛ 428: IOT PROGRAMMING

Dr. Panayiotis Kolios
Assistant Professor, Dept. Computer Science,
KIOS CoE for Intelligent Systems and Networks
Office: FST 01, 116
Telephone: +357 22893450 / 22892695
Web: https://www.kios.ucy.ac.cy/pkolios/

Πανεπιστήμιο Κύπρου

- Correlation functions measure the degree of association between two random variables

- In time series data it measures the association between two different points in time

- Correlation is essential for understanding linear relationships and dependencies in the data

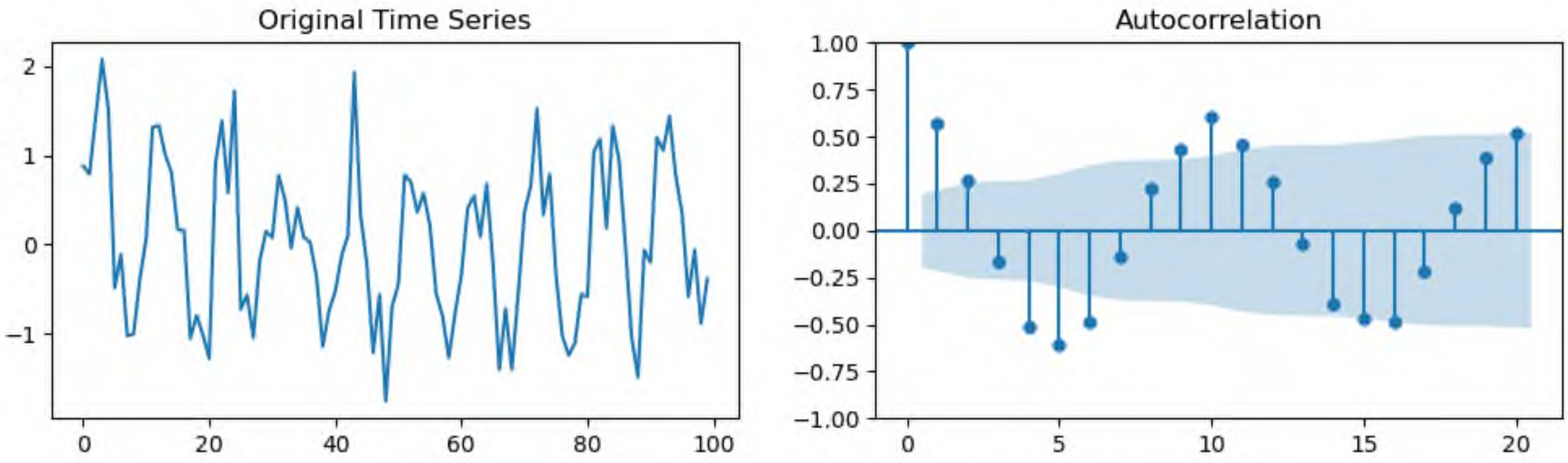
- Two types of correlation functions
    - Autocorrelation function (ACF) and partial ACF (PACF)
    - Cross-correlation

- ACF measures the correlation of a time series with its own lagged values

$$ACF(k) = \frac{E[(X(t) - \mu)(X(t-k) - \mu)]}{\sigma^2}$$

- where
  - k is the lag
  - X(t) is the value at time t
  - μ is the mean of the time series
  - σ^2 is the variance of the series
- Values close to 1 or -1 indicate strong correlation, while values near 0 indicate weak correlation

```
n = 100
time_series_1 = np.random.normal(0, 0.5, n) + np.sin(2*np.pi/10*np.arange(n))
fig, axes = plt.subplots(1, 2, figsize=(12, 3))
axes[0].plot(time_series_1)
axes[0].set_title('Original Time Series')
plot_acf(time_series_1, lags=20, alpha=0.05, ax=axes[1]); # We plot the first 20 lags. We could plot more by changing the `lags` argument.
```



- Peaks indicate lagged correlation values
- The specific time series correlates well with itself shifted by 1 (lag 1)
- Blue region represents a confidence interval
- Correlations outside of the confidence interval (95% used here), are statistically significant whereas the others are not

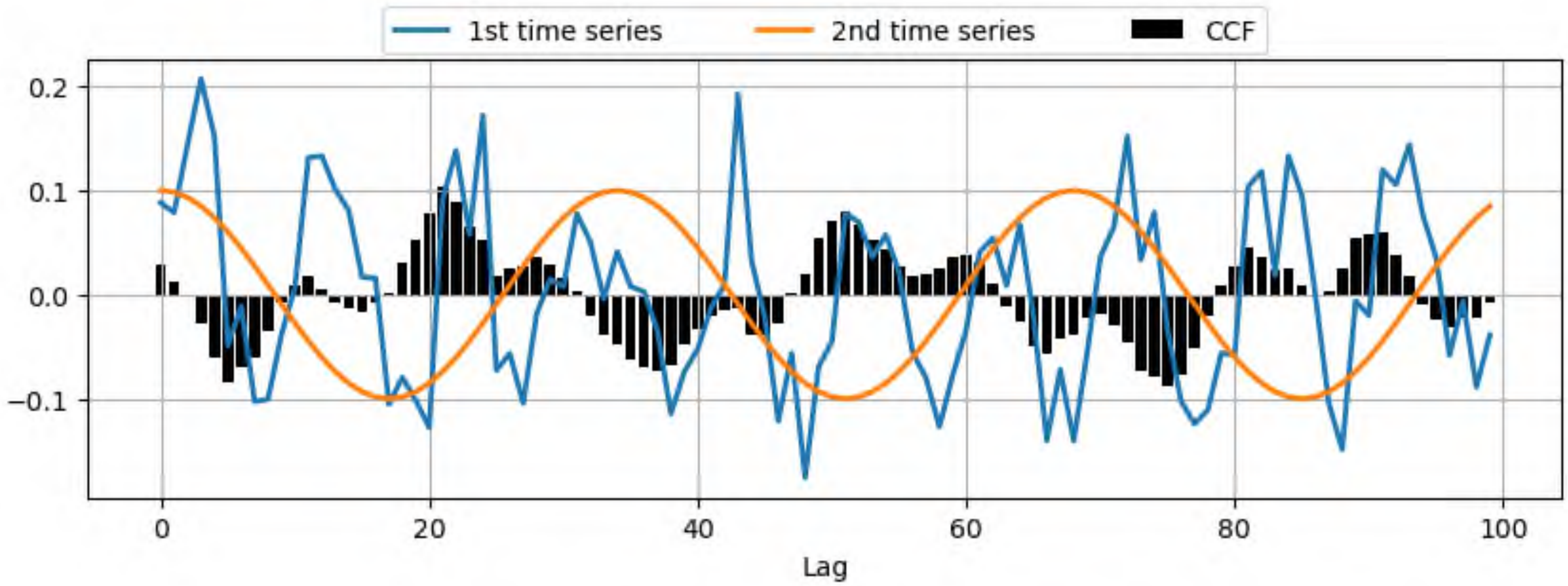
- CCF measures the correlation between two time series at different lags

$$CCF(k) = \frac{E[(X(t) - \mu_x)\left(Y(t-k) - \mu_y\right)]}{\sigma_x \sigma_y}$$

- where
  - k is the lag
  - X and Y are different time series
  - μ σ are their means and standard deviations

```
time_series_2 = np.cos(np.pi/17*np.arange(n))

# Calculate CCF between the two time series
ccf_vals = ccf(time_series_1, time_series_2, adjusted=False)

# Plot CCF
_, ax = plt.subplots(1,1,figsize=(10,3))
ax.bar(range(len(ccf_vals)), ccf_vals, color='k', label='CCF')
ax.plot(time_series_1*0.1, linewidth=2, label='1st time series')
ax.plot(time_series_2*0.1, linewidth=2, label='2nd time series')
```

CCF EXAMPLE



- CCF commonly used for searching a shorter, known feature, within a long signal

- Applications
  - Identifying the nature of the data (e.g., whether it is random, has a trend or a seasonality)
  - Identifying outliers in time series data

- Limitations
  - Only measure linear relationships
  - Time series should be stationary for meaningful results
  - Higher correlation does not imply causation
  - ACF measures both direct and indirect correlations between lags
    - Strong correlation at higher lags could be a result of accumulated correlations of shorter lags

CORRELATION IS NOT CAUSATION

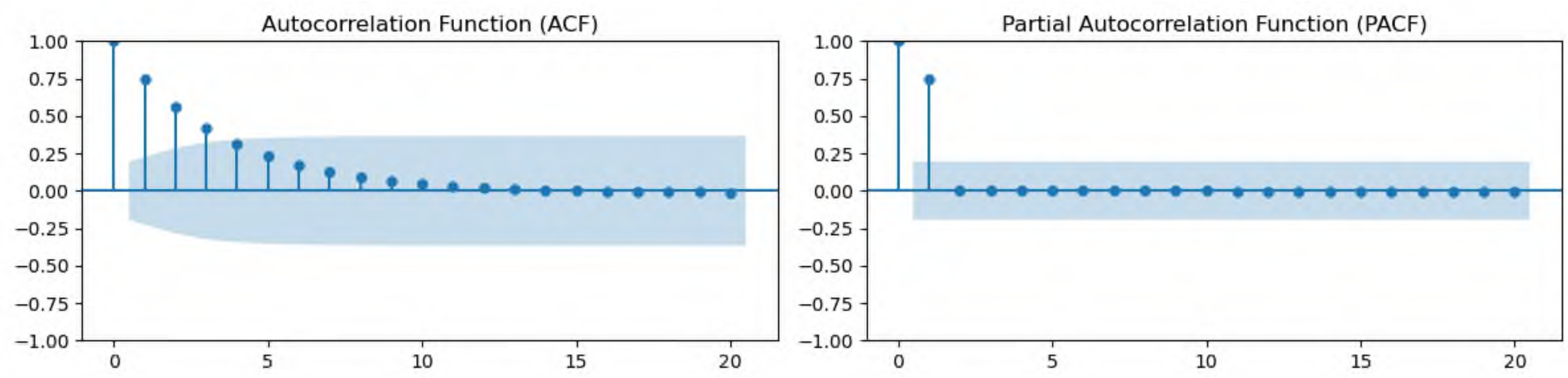
- PACF avoids indirect correlations between lags
- It does so by isolating the direct correlation and the lagged version

$$\varphi_{kk} = Corr(X(t) - \hat{X}(t), X(t-k) - \hat{X}(t-k))$$

- where
  - $\hat{X}(t)$ is the predicted value of $X(t)$ based on all values up to t-1
  - $\hat{X}(t-k)$ is the predicted value of $X(t-k)$ based on all values up to t-k-1
- Using ACF and PACF together provides a more comprehensive understanding of the time series data
- ACF helps in identifying the overall correlation structure and potential seasonality
- PACF pinpoints the specific lags that have a significant direct impact on the current value

Πανεπιστήμιο Κύπρου

- Consider a time series where ACF shows significant correlation at lags 1, 2, and 3
- Without PACF, it is unclear whether the correlation at lag 3 is direct or merely a reflection of the strong correlations at lags 1 and 2
- PACF can resolve by calculating direct correlation at lag 3



- Lag estimation with ACF and PACF is helpful in modelling time-series with autoregressive and moving-average models

AUTOREGRESSIVE MODELS

Πανεπιστήμιο Κύπρου

- An Autoregressive (AR) model is a type of time series model that uses observations from previous time steps as input to a regression equation to predict the value at the next time step
- The AR model is dependent solely on its own past values
- The general form of an AR model of order p is:

$$X(t) = c + \varphi_1 X(t-1) + \varphi_2 X(t-2) + \ \ldots + \varphi_p X(t-p) + \varepsilon_t$$

- where
  - X(t) value at time t
  - c is constant term
  - $\varphi_1, \varphi_2, \ldots, \varphi_p$ are coefficients of the model
  - ε_t is the error magnitude at time t

Πανεπιστήμιο Κύπρου

- AR(1) is the first-order autoregressive model:

$$X(t) = c + \varphi_1 X(t-1) + \varepsilon_t$$

  - where the current value depends on the immediately preceding value

- AR(2) is the second-order autoregressive model:

$$X(t) = c + \varphi_1 X(t-1) + \varphi_2 X(t-2) + \varepsilon_t$$

  - where the current value depends on two past values

- Coefficients of AR models can be estimated using various methods like the Maximum Likelihood Estimation, or Least Squares Estimation.

- The estimated coefficients provide insights into the influence of past values on the current value in the time series
    - Hence AR models are interpretable!!

Limitations of AR Models

- AR models require the time series to be stationary
- Higher order AR models could overfit on training data
    - Hence perform bad in prediction

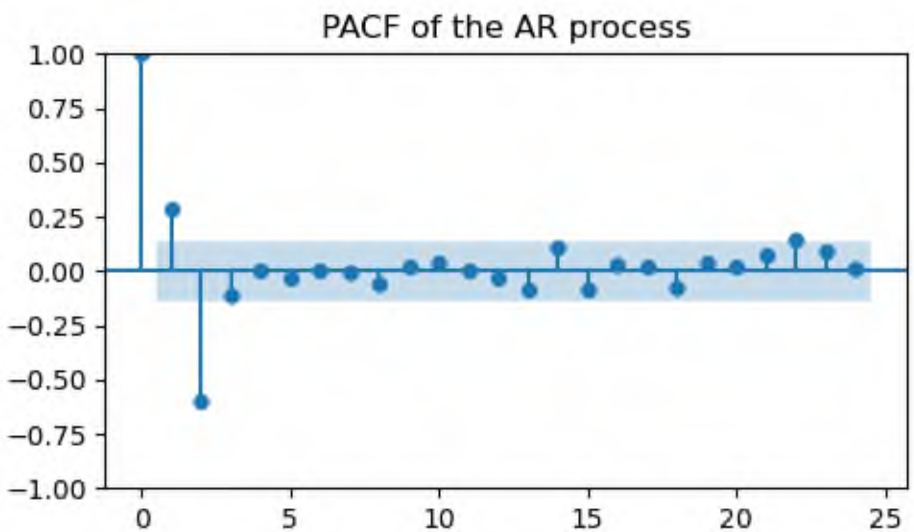
- Cannot model non-linear relationships in the data

- AR models do not account for exogenous factors
    - Additional data relevant for the prediction
    - e.g., outside temperatures when predicting the electricity demand
    - Extension models (such as ARMAX) accommodate exogenous variables

Πανεπιστήμιο Κύπρου

- How do we determine the best order p of the AR model?
- Look at the first lags of the PACF
- Example:
  - $\varphi = [1.0, -0.5, 0.7]$;
  - note: zero lag is always, i.e., $\varphi_0 = 1.0$

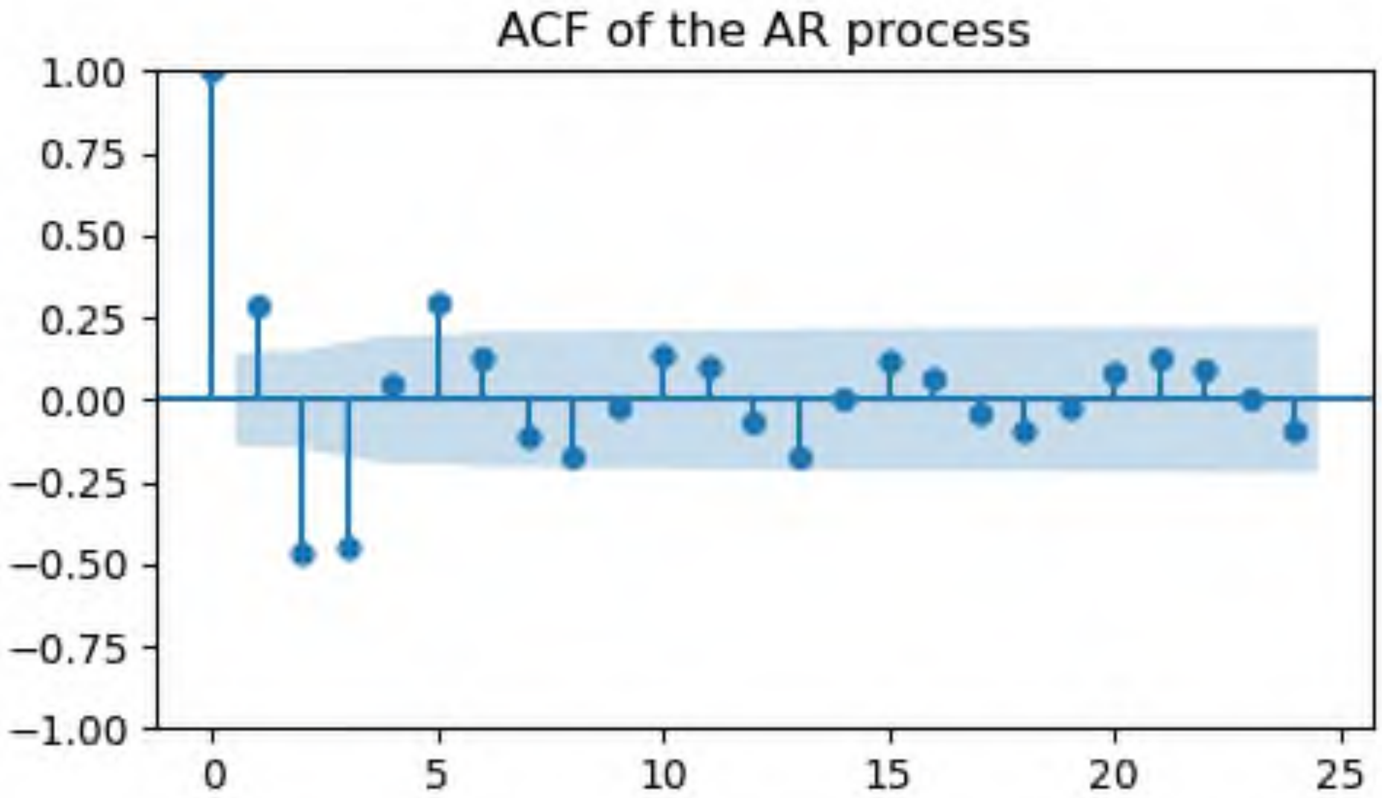
```
ar_data = arma_generate_sample(ar=np.array([1.0, -0.5, 0.7]), ma=np.array([1]),
nsample=200, scale=1, burnin=1000)

# Compute PACF
_, ax = plt.subplots(1, 1, figsize=(5, 3))
plot_pacf(ar_data, ax=ax, title="PACF of the AR process")
plt.tight_layout();
```



Πανεπιστήμιο Κύπρου

- Besides the spike at lag 0, which is always there, we see two significant lags:
    - positive spike at lag 1, introduced by the negative coefficient 0.5
    - negative spike at lag 2, introduced by the positive coefficient 0.7
- Indicating a process with a memory of length 2
- i.e, most of the correlations in the data are explained by the previous 2 time steps
- Hence to fit an AR model to our data choose $p = 2$, AR(2) model

- An AR process is characterized by correlations that decay **slowly** in time
- This can be seen by looking at the ACF plot, where we see significant spikes over many lags



Πανεπιστήμιο Κύπρου

How to use AR models for forecasts?

- In general, time series has a trend and a seasonality
- However, an AR model can only be used on *stationary* data
- Therefore, we need to proceed as follows

Step 1: Remove trend and seasonality

- Apply standard and seasonal differencing

$R'(t) = X(t+1) - X(t)$ removes trend
$R(t) = R'(t+L) - R'(t)$ removes seasonality

- Or estimate trend T and seasonality S (e.g.,by using seasonal decomposition or smoothing techniques) and subtract them:

$$R(t) = X(t) - T(t) - S(t)$$

Step 2: Apply the AR model

- Identify the order of the AR model
- Fit an AR model to the detrended and deseasonalized time series $R(t)$
- Use the model to forecast the next values

$$\hat{R}(t+\tau), \tau = 1, \ldots, H \text{ where } H \text{ is the forecast horizon}$$

Step 3: Reconstruct the forecast

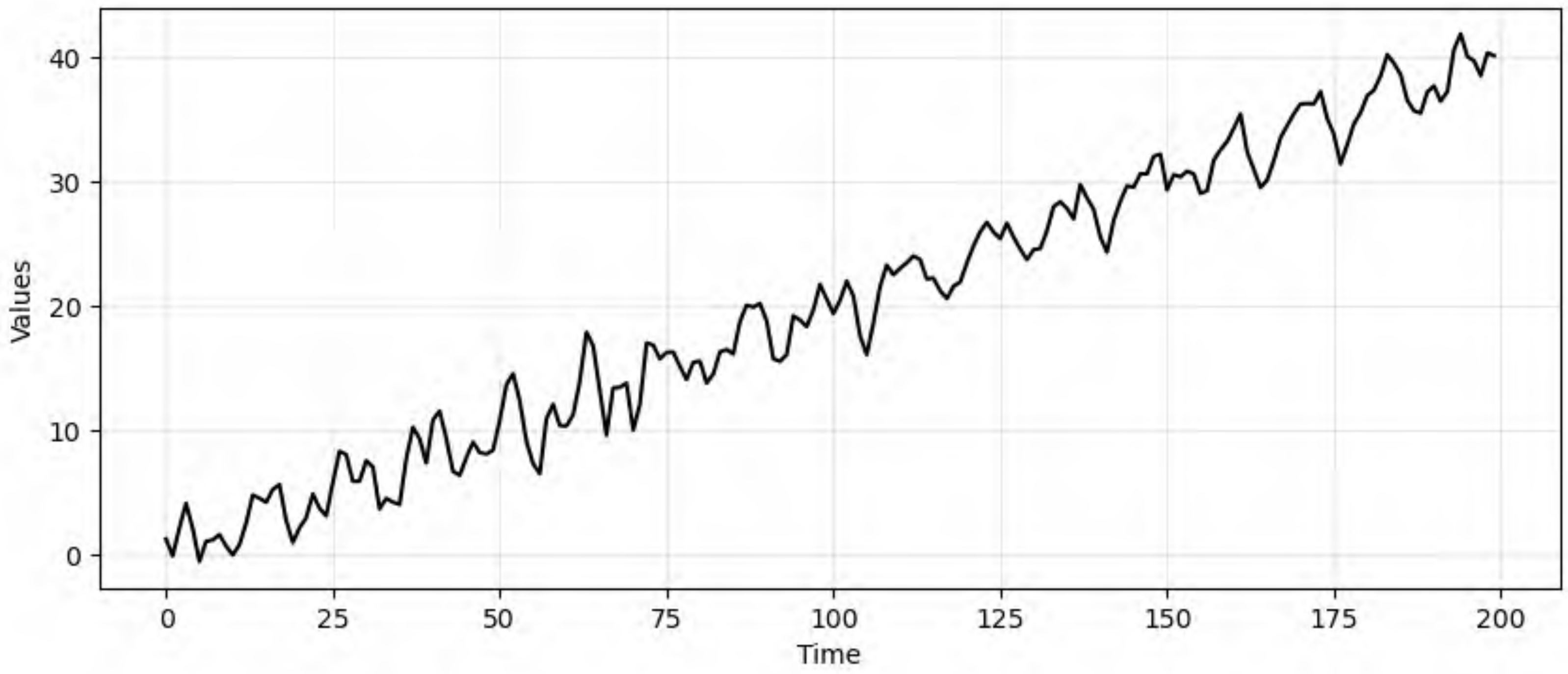
- If differencing used in step 1:
  - take cumulative sums of the residuals
- If modeled and removed trend and seasonality in step 1:
  - Predict trend $\hat{T}(t+\tau)$ and seasonality comp. $\hat{S}(t+\tau)$
  - Add back estimates:

$$\hat{X}(t+\tau) = \hat{R}(t+\tau) + \hat{T}(t+\tau) + \hat{S}(t+\tau)$$

Πανεπιστήμιο Κύπρου

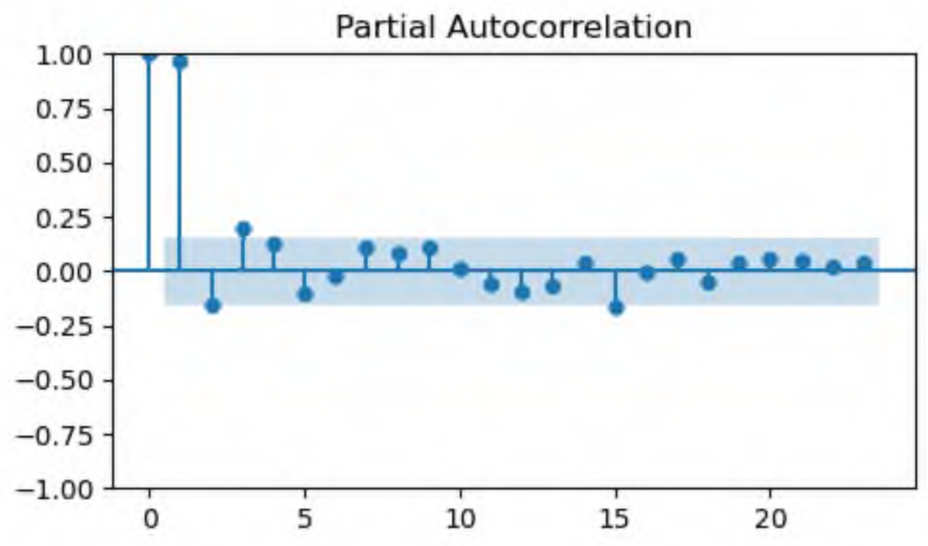
EXAMPE AR FORECAST

```python
# Generate data with trend and seasonality
time = np.arange(200)
trend = time * 0.2
seasonality = 2*np.sin(2*np.pi*time/12) # Seasonality 12
time_series_ar = trend + seasonality + ar_data

_, ax = plt.subplots(1, 1, figsize=(10, 4))
run_sequence_plot(time, time_series_ar, "", ax=ax);
```



Πανεπιστήμιο Κύπρου

```
# Split data to train and test dataset
train_data_ar = time_series_ar[:164]
test_data_ar = time_series_ar[164:]

# Determine order p of AR model by finding the least nonzero lag in PACF
_, ax = plt.subplots(1, 1, figsize=(5, 3))
plot_pacf(train_data_ar, ax=ax)
plt.tight_layout();
```



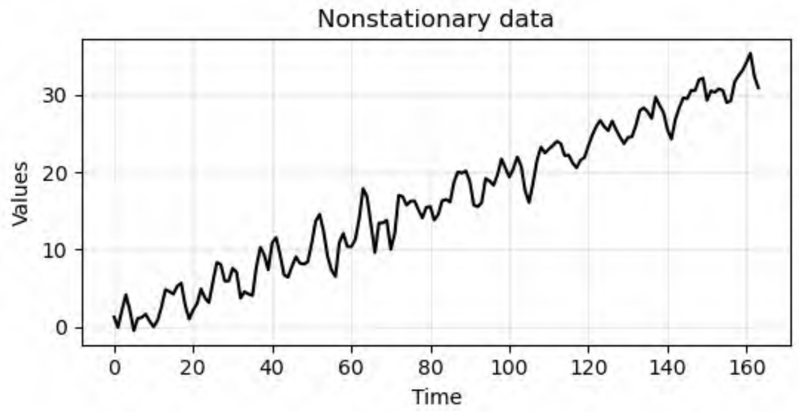
- PACF suggests $p = 1$, however correlation does not drop quickly and this is an indication on nonstationarity

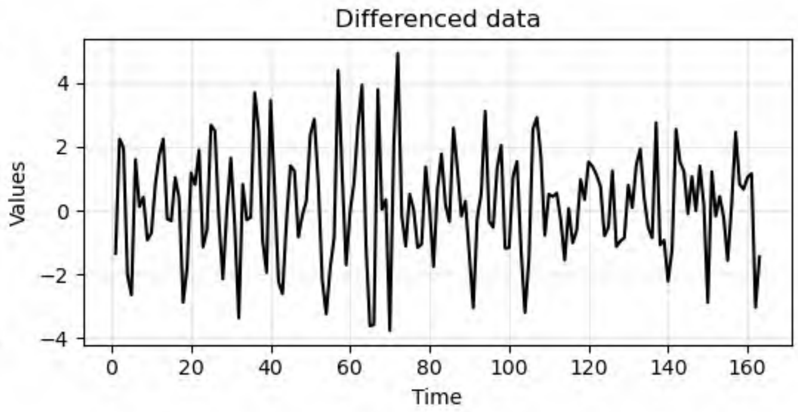
```
# Test data for stationarity
_, pvalue, _, _, _, _ = adfuller(train_data_ar)
print(f"p-value: {pvalue:.3f}")
```

- $\mathrm{p} - value = 0.985$, which is high!! and fails to reject the null hypothesis H_0: $the\ data\ is\ nonstationary$
- Hence to achieve stationarity:
  - Apply differencing

```
# Test data for stationarity
diff_ar = train_data_ar[1:] - train_data_ar[:-1]
_, pvalue, _, _, _, _ = adfuller(diff_ar)
print(f"p-value: {pvalue:.3f}")
```

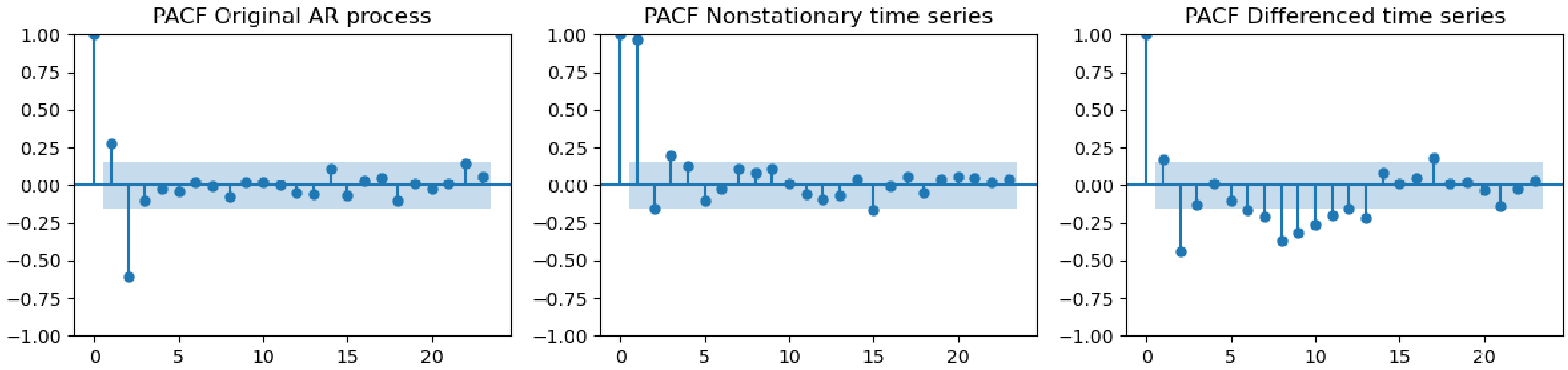
  - $\mathrm{p} - value = 0.000$





- Recompute PACF on the differenced data

```
# Test data for stationarity on differenced data
_, axes = plt.subplots(1, 3, figsize=(12, 3))
plot_pacf(ar_data[:len(train_data_ar)], ax=axes[0], title="PACF Original AR
process")
plot_pacf(train_data_ar, ax=axes[1], title="PACF Nonstationary time series")
plot_pacf(diff_ar, ax=axes[2], title="PACF Differenced time series")
plt.tight_layout();
```

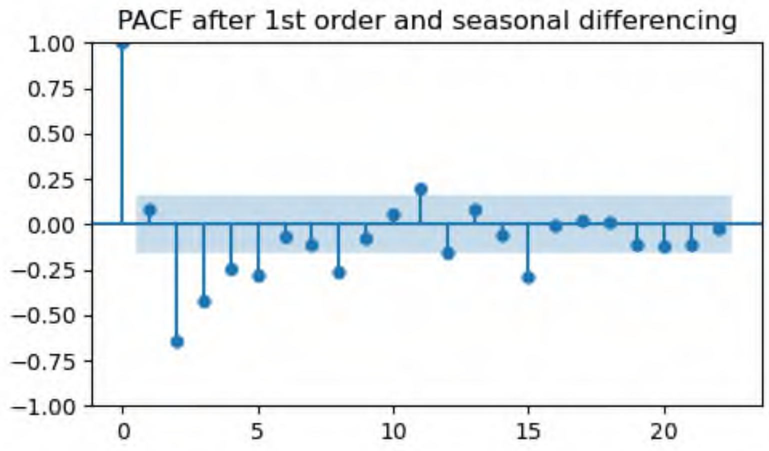


- Prominent spike now at $p = 2$, and suggest use of AR(2)
- There are additional spikes at higher lags that are however not present in the original data PACF plot

Πανεπιστήμιο Κύπρου

- This is because of the seasonal component in the data
- Removing also seasonality with differencing

```
# Removing seasonality with differencing and plot PACF
diff_diff_ar = diff_ar[12:] - diff_ar[:-12]

_, ax = plt.subplots(1, 1, figsize=(5, 3))
plot_pacf(diff_diff_ar, ax=ax, title="PACF after 1st order and seasonal
differencing")
plt.tight_layout();
```



- This PACF looks even more different that the original PACF
- Difficult in practice to select model order with PACF using differencing, especially for first order models

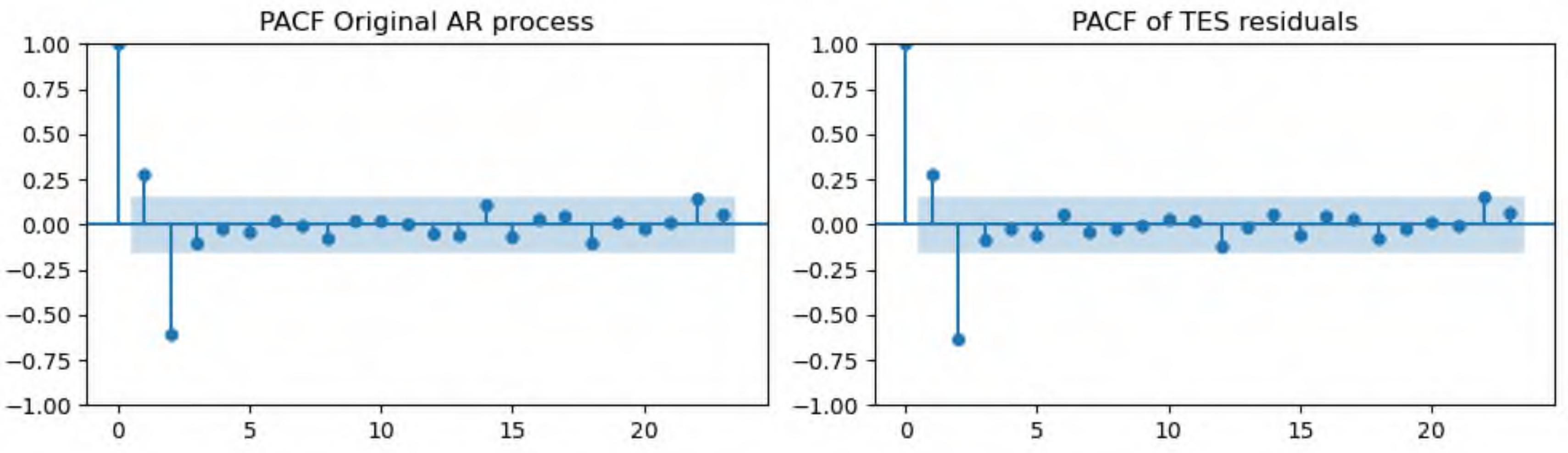
Πανεπιστήμιο Κύπρου

- Overdifferencing
  - Differencing too many times might compromise the data structure
  - Challenge to determine the optimal order of differentiation
- In addition when differencing used to achieve stationarity
  - Data is lost at the beginning and end of the time series
  - Easier to make mistakes when reverting differencing to make predictions
  - Complicates the process when the forecast horizon H goes beyond the seasonality L

Stationarity by subtracting estimated trend and seasonality

- Use for example Triple exponential smoothing (TES)

```
# Triple Exponential Smoothing
tes = ExponentialSmoothing(train_data_ar, trend='add',
                           seasonal='add', seasonal_periods=12).fit()
trend_and_seasonality = tes.fittedvalues # Estimated trend and seasonality
tes_resid = train_data_ar - trend_and_seasonality

_, axes = plt.subplots(1, 2, figsize=(10, 3))
plot_pacf(ar_data[:len(train_data_ar)], ax=axes[0], title="PACF Original AR
process")
plot_pacf(tes_resid, ax=axes[1], title="PACF of TES residuals")
plt.tight_layout();
```



Πανεπιστήμιο Κύπρου

Stationarity by subtracting estimated trend and seasonality

- Need to estimate the period of seasonality

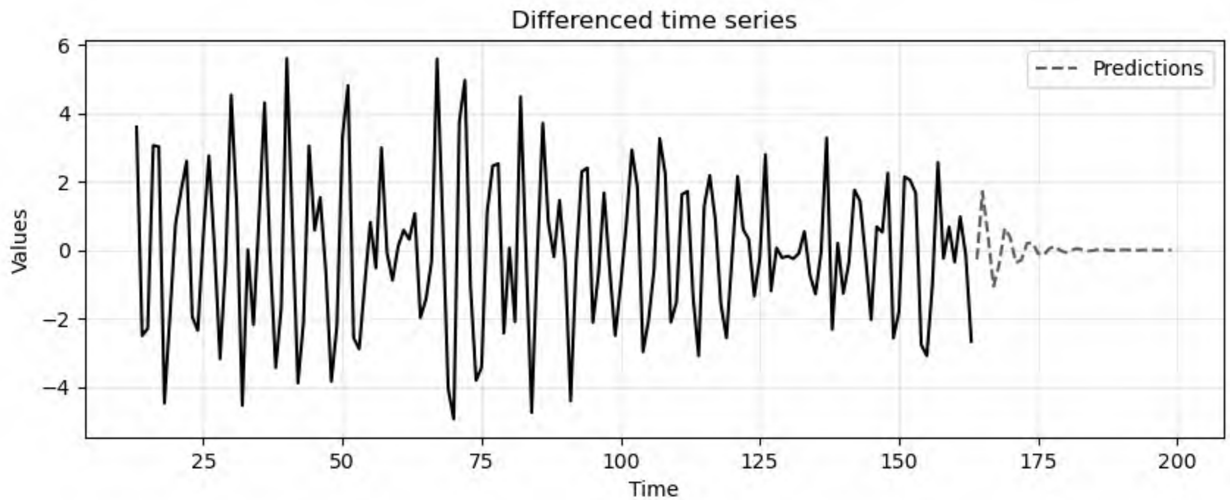
```
# Use FFT to compute period of seasonality in data
from tsa_course.lecture1 import fft_analysis

period, _, _ =fft_analysis(time_series_ar)
print(f"Period: {np.round(period)}")
```

- Smoothing works well because toy data has additive components, linear trend, constant variance!
- In reality things are not so straightforward!

Forecasting with AR(2) model

- Stationarity obtained with differencing
  - Forecast as long as the test data
  - However, forecasts tend to zero due to very high uncertainty

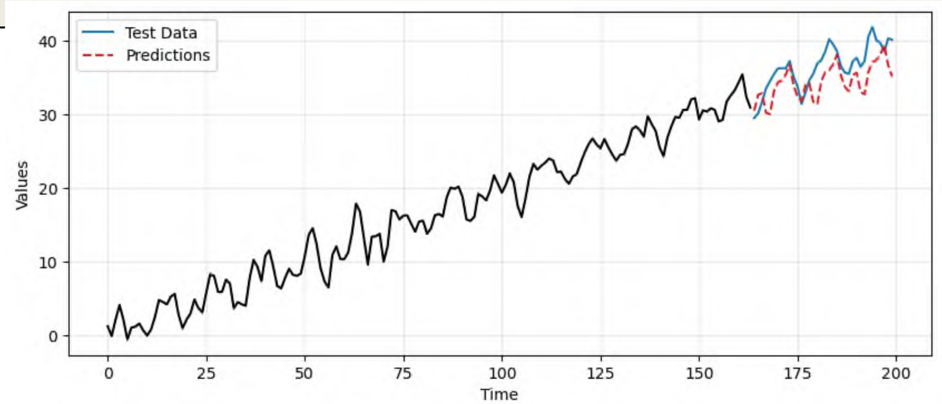
```
# Fit the model
model = ARIMA(diff_diff_ar, order=(2,0,0))
model_fit = model.fit()

# Compute predictions
diff_preds = model_fit.forecast(steps=len(test_data_ar))
ax = run_sequence_plot(time[13:len(train_data_ar)], diff_diff_ar, "")
ax.plot(time[len(train_data_ar):], diff_preds, label='Predictions', linestyle='--
', color='tab:red')
plt.title('Differenced time series')
plt.legend();
```



Forecasting with AR(2) model

- Stationarity obtained with differencing
  - Actual predictions obtained by reverting seasonal difference and 1st-order differencing

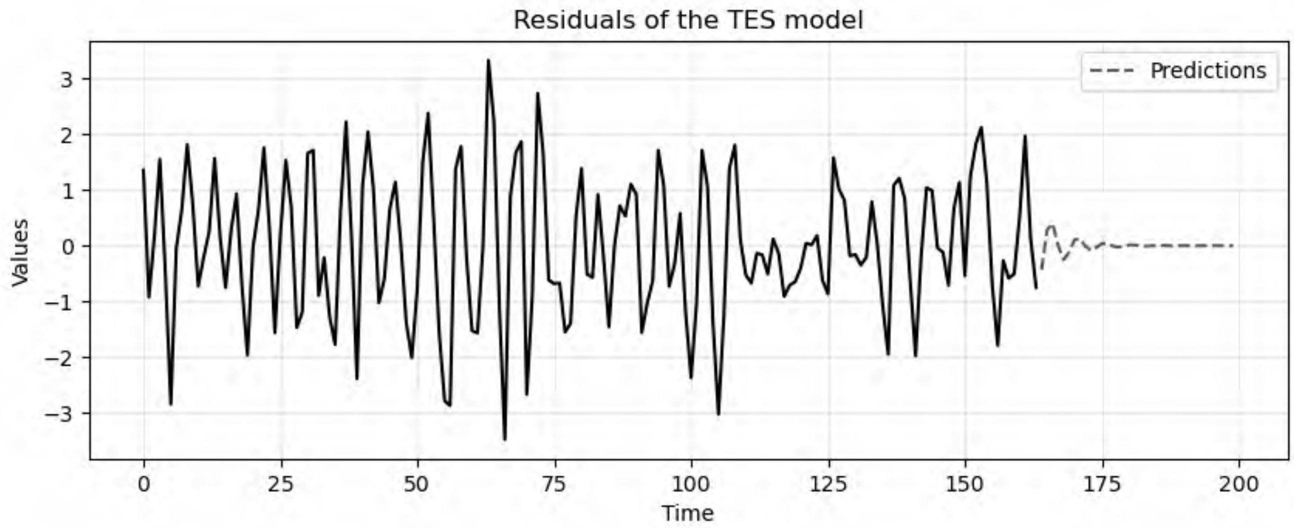
```
# Reintegrating the seasonal differencing
reintegrated_seasonal = np.zeros(len(test_data_ar))
reintegrated_seasonal[:12] = diff_ar[-12:] + diff_preds[:12]
for i in range(12, len(test_data_ar)):
    reintegrated_seasonal[i] = reintegrated_seasonal[i-12] + diff_preds[i]

# Reintegrating 1st order differencing
reintegrated = reintegrated_seasonal.cumsum() + train_data_ar[-1]
_, ax = plt.subplots(1, 1, figsize=(10, 4))
run_sequence_plot(time[:len(train_data_ar)], train_data_ar, "", ax=ax)
ax.plot(time[len(train_data_ar):], test_data_ar, label='Test Data',
color='tab:blue')
ax.plot(time[len(train_data_ar):], reintegrated, label='Predictions',
linestyle='--', color='tab:red')
plt.legend();
```



Forecasting with AR(2) model

- Stationarity obtained with TES
  - Originally subtracted trend and seasonality estimated with TES

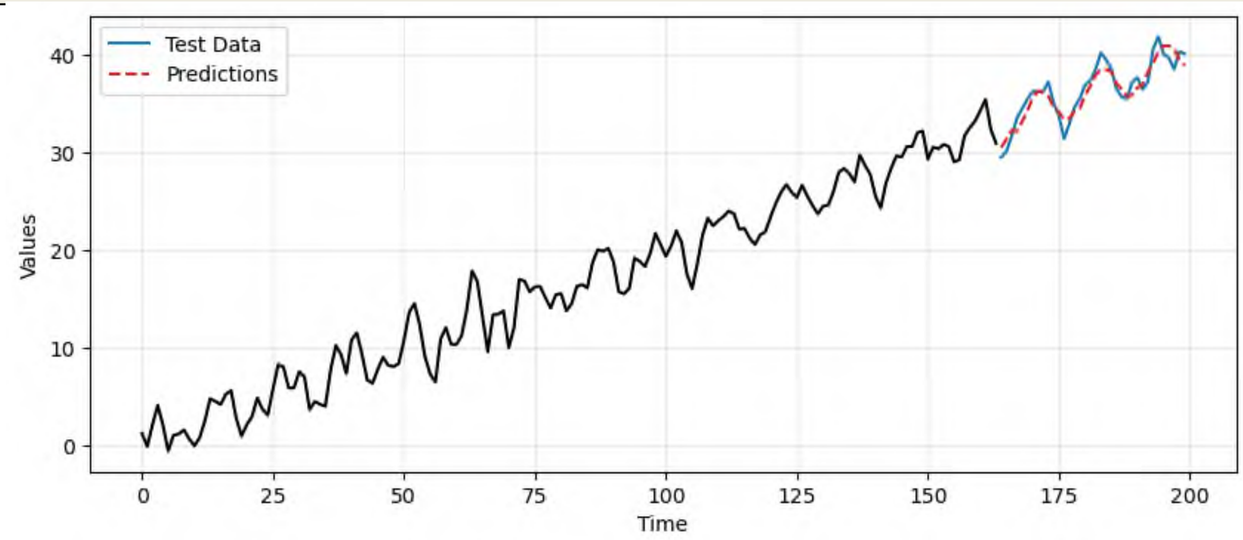
```
# Use tes_resid
model = ARIMA(tes_resid, order=(2,0,0))
model_fit = model.fit()

resid_preds = model_fit.forecast(steps=len(test_data_ar))

ax = run_sequence_plot(time[:len(train_data_ar)], tes_resid, "")
ax.plot(time[len(train_data_ar):], resid_preds, label='Predictions', linestyle='-
-', color='tab:red')
plt.title('Residuals of the TES model')
plt.legend();
```



Forecasting with AR(2) model

- Stationarity obtained with TES
  - Add trend and seasonality to the predictions
  - Sum all predictions to obtain the actual values

```
# Add back trend and seasonality to the predictions
tes_pred = tes.forecast(len(test_data_ar))
final_preds = tes_pred + resid_preds

_, ax = plt.subplots(1, 1, figsize=(10, 4))
run_sequence_plot(time[:len(train_data_ar)], train_data_ar, "", ax=ax)
ax.plot(time[len(train_data_ar):], test_data_ar, label='Test Data',
color='tab:blue')
ax.plot(time[len(train_data_ar):], final_preds, label='Predictions', linestyle='-
-', color='tab:red')
plt.legend();
```



Quantifying the performance difference of the two approaches

- Compute MSE

```
# Compute MSE for differencing and TES
mse_differencing = mean_squared_error(test_data_ar, reintegrated)
mse_tes = mean_squared_error(test_data_ar, final_preds)

print(f"MSE of differencing: {mse_differencing:.2f}")
print(f"MSE of TES: {mse_tes:.2f}")
```

- MSE of differencing: 8.03
- MSE of TES: 1.12

MOVING AVERAGE MODELS

Πανεπιστήμιο Κύπρου

- Another approach to modeling univariate time series is the moving average (MA) model
- The MA model is a linear regression of the current value of the series against the white noise of one or more of the previous values of the series
- The noise at each point is assumed to come from a normal distribution with mean 0 and constant variance
- The MA model is defined as

$$X(t) = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ \dots + \theta_q \varepsilon_{t-q}$$

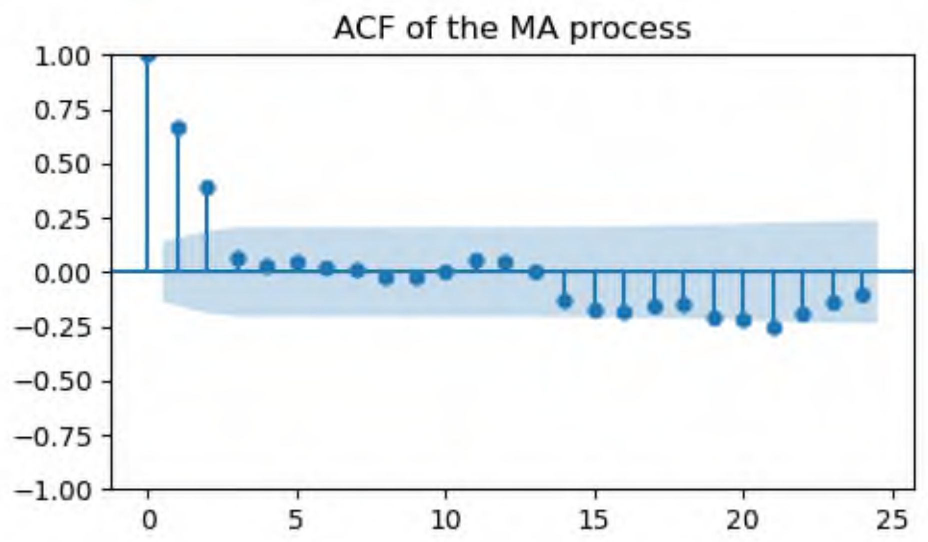
- where
  - μ is the mean of the time series
  - $\theta_1, \theta_2, \theta_q$ are model coefficients
  - q is the model order (lagged error terms)
  - ε_t is the error term at time t

Πανεπιστήμιο Κύπρου

- MA models capture the dependency between an observation and a residual error through a moving average applied to lagged observations
- Fitting MA estimates is more complicated than AR models because the error terms are not observable
- Therefore, iterative nonlinear fitting procedures need to be used
- MA models are less interpretable than AR models
- MA also requires data to be stationary
- Important
    - Moving average smoothing **is not the same as** Moving Average Models
    - Serve different purposes

- Identify the order q as the lag which spikes in ACF become nonsignificant

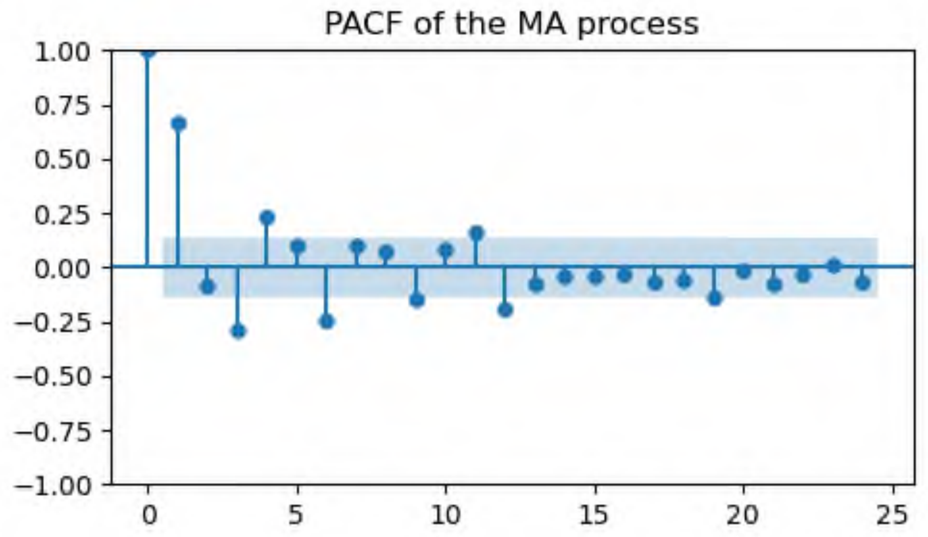
```
# Generate data from an MA(2) process
ma = np.array([1.0, 0.7, 0.8])  # MA parameters
ma_data = arma_generate_sample(np.array([1]), ma, nsample=len(time), scale=1,
burnin=1000) # MA process

_, ax = plt.subplots(1, 1, figsize=(5, 3))
plot_acf(ma_data, ax=ax, title="ACF of the MA process")
plt.tight_layout();
```



- Cutoff after the second lag indicating MA model with $q = 2$

- Characteristic of an MA process are the slowly decaying, alternative spikes in the PACF plot

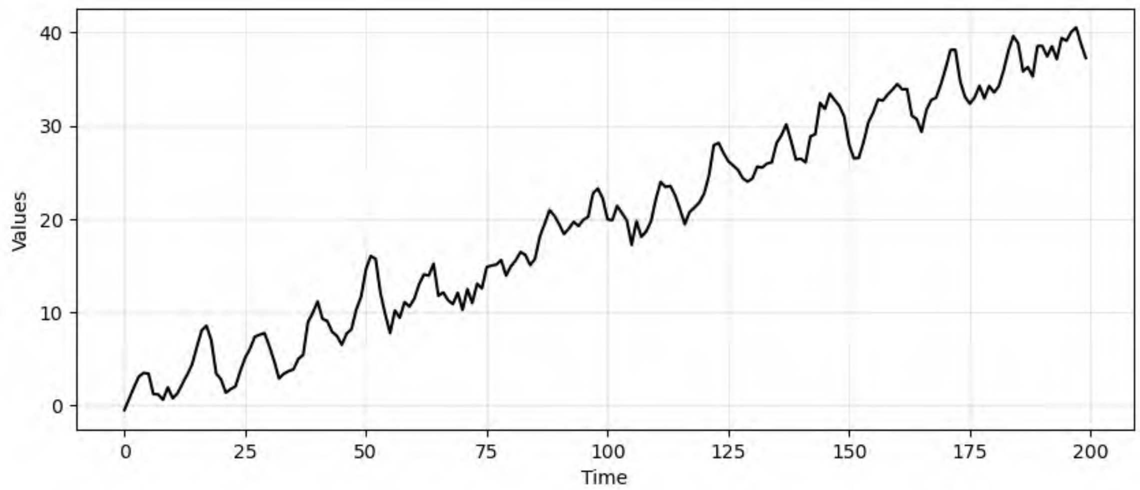
```
# Plot PACF of MA(2) process
_, ax = plt.subplots(1, 1, figsize=(5, 3))
plot_pacf(ma_data, ax=ax, title="PACF of the MA process")
plt.tight_layout();
```



- Complementary to what was observed for AR

Example: Forecasting with MA Model

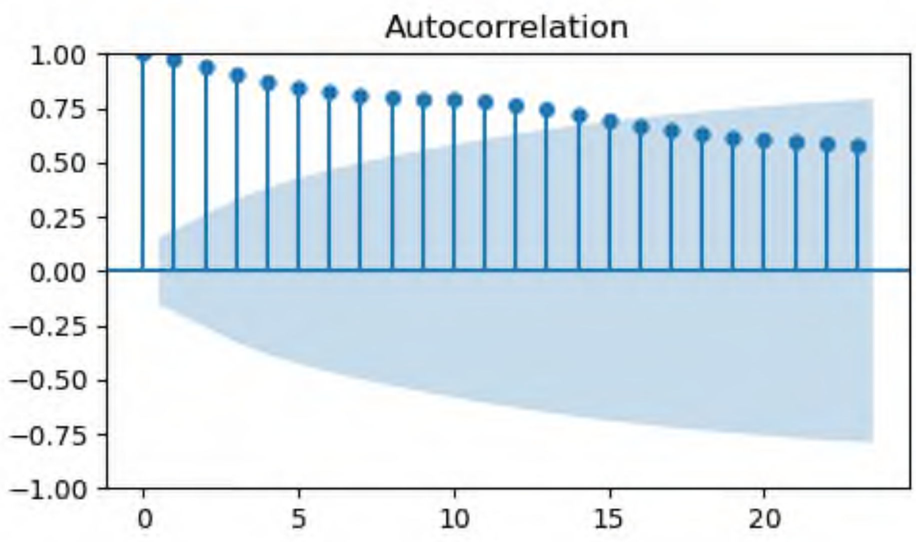
```
# Create MA dataset
time_series_ma = trend + seasonality + ma_data

_, ax = plt.subplots(1, 1, figsize=(10, 4))
run_sequence_plot(time, time_series_ma, "", ax=ax);
```




```
# Split to train/test data set
train_data_ma = time_series_ma[:164]
test_data_ma = time_series_ma[164:]
```

- Identify order q using the ACF plot

```
# Compute ACF of train_data_ma
_, ax = plt.subplots(1, 1, figsize=(5, 3))
plot_acf(train_data_ma, ax=ax)
plt.tight_layout();
```



- As before, stationarity is needed to correctly estimate the MA coefficients and meaningful ACF plot

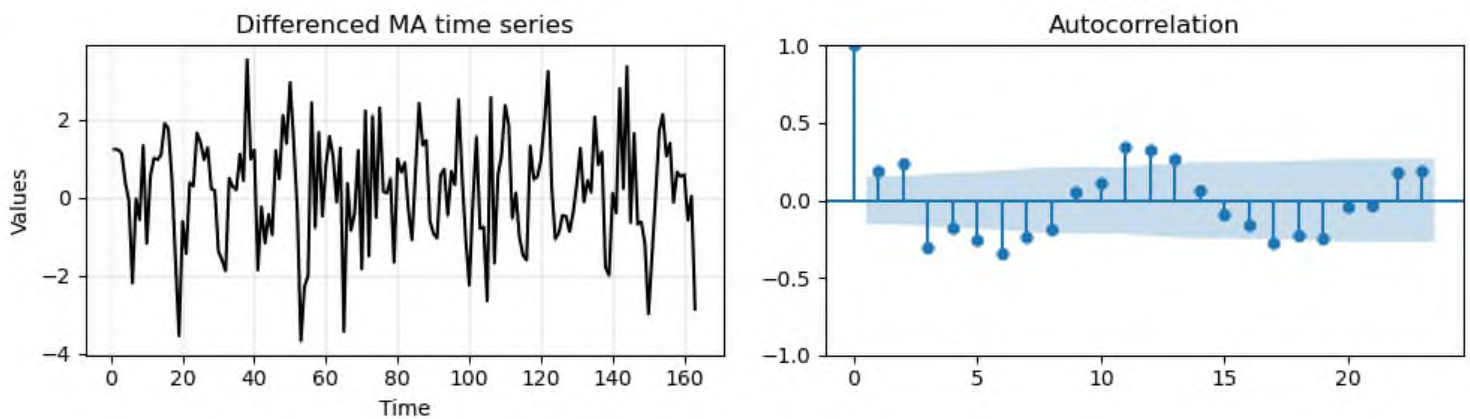
Πανεπιστήμιο Κύπρου

- Stationarity using differencing

```
# Stationarity for MA dataset
diff_ma = train_data_ma[1:] - train_data_ma[:-1]
# Use ADF test to verify data stationarity (show before and after value)
_, pvalue_ts, _, _, _, _ = adfuller(train_data_ma)
_, pvalue_diff, _, _, _, _ = adfuller(diff_ma)
print(f"p-value (original ts): {pvalue_ts:.3f}")
print(f"p-value (differenced ts): {pvalue_diff:.3f}")
```

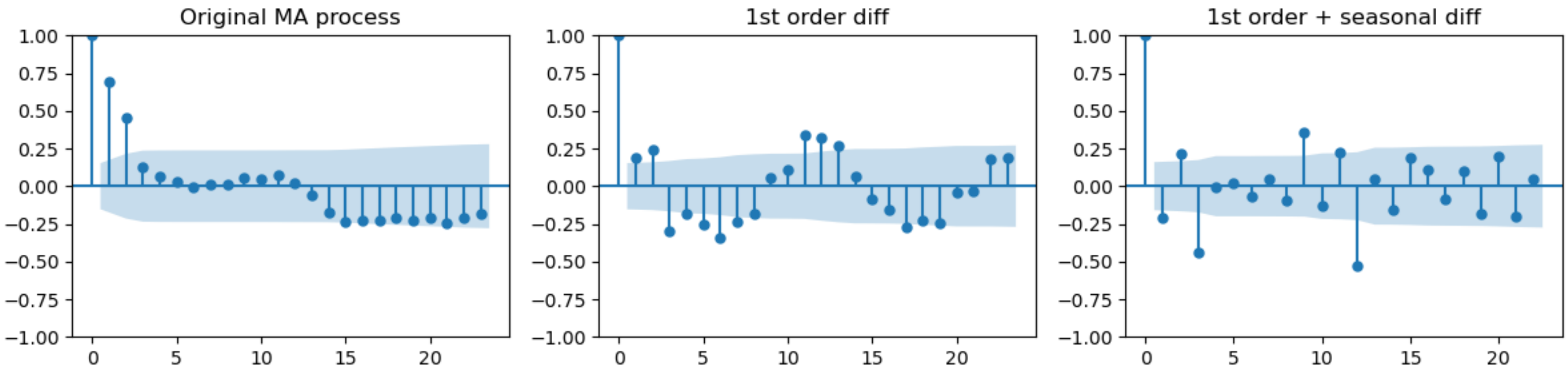
- $\mathrm{p} - value(original\ ts)$: 0.959
- $\mathrm{p} - value(differenced\ ts)$: 0.000

```
# Plot differenced data and compute ACF
_, axes = plt.subplots(1,2, figsize=(10, 3))
run_sequence_plot(time[1:len(train_data_ma)], diff_ma, "Differenced MA time
series", ax=axes[0])
plot_acf(diff_ma, ax=axes[1])
plt.tight_layout();
```



- Removing seasonal component

```
# Seasonal component
diff_diff_ma = diff_ma[12:] - diff_ma[:-12]

_, axes = plt.subplots(1,3, figsize=(12, 3))
plot_acf(ma_data[:len(train_data_ma)], ax=axes[0], title="Original MA process")
plot_acf(diff_ma, ax=axes[1], title="1st order diff")
plot_acf(diff_diff_ma, ax=axes[2], title="1st order + seasonal diff")
plt.tight_layout();
```



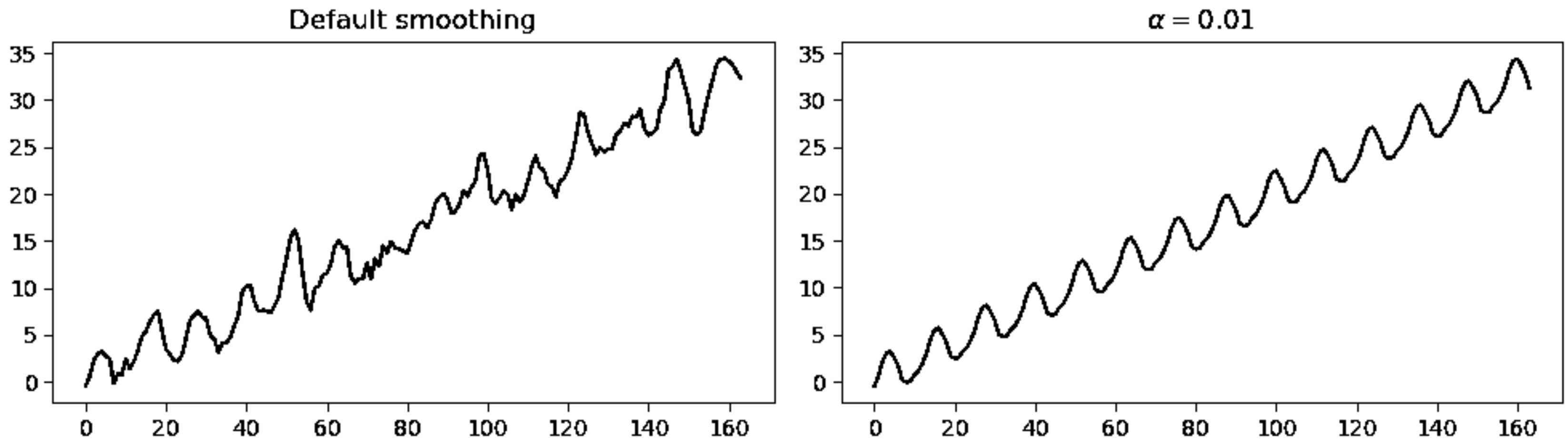
- However this results to overdifferencing
  - Hence not sure which MA model to use by analysing ACF plots

Πανεπιστήμιο Κύπρου

- Stationarity by subtracting estimated trend and seasonality
  - Use TES
  - MA process is more noisy than AR and thus increase the smoothing level by setting $\alpha = 0.01$

```
# Seasonal component
period, _, _ =fft_analysis(time_series_ma)
period = np.round(period).astype(int)

tes_ma_default = ExponentialSmoothing(train_data_ma, trend='add',
                            seasonal='add',
seasonal_periods=period).fit(smoothing_level=None)
tes_ma = ExponentialSmoothing(train_data_ma, trend='add',
                            seasonal='add',
seasonal_periods=period).fit(smoothing_level=0.01)
trend_and_seasonality_default = tes_ma_default.fittedvalues
trend_and_seasonality = tes_ma.fittedvalues

_, axes = plt.subplots(1, 2, figsize=(10, 3))
axes[0].plot(trend_and_seasonality_default, 'k')
axes[0].set_title('Default smoothing')
axes[1].plot(trend_and_seasonality, 'k')
axes[1].set_title('$\\alpha=0.01$')
plt.tight_layout();
```

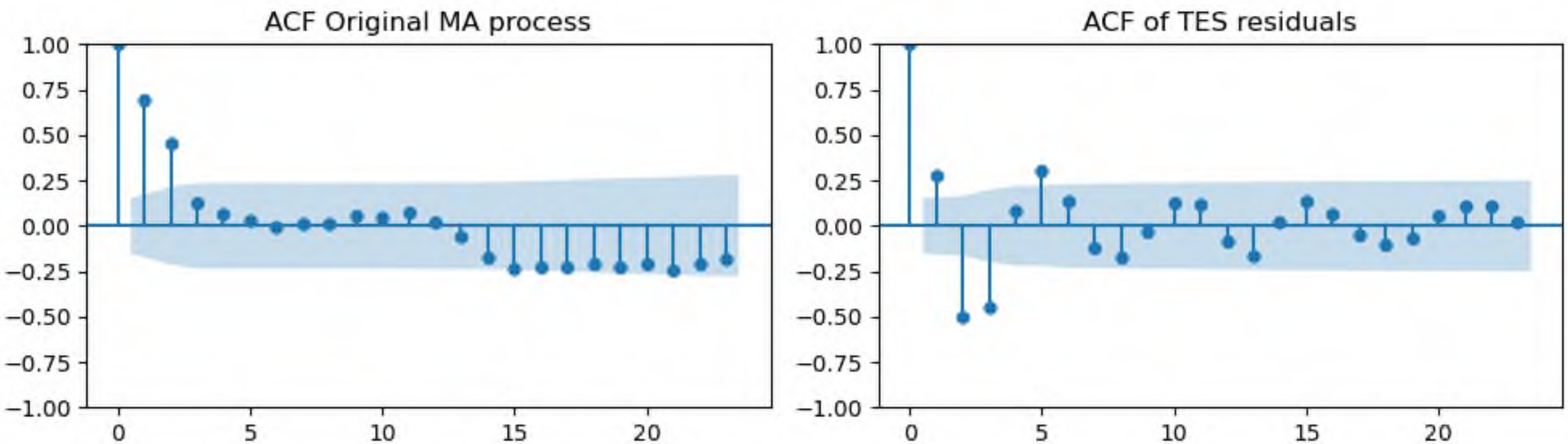
- Stationarity by subtracting estimated trend and seasonality
  - Use TES
  - MA process is more noisy than AR and thus increase the smoothing level by setting $\alpha = 0.01$



  - Leads to smoother estimate of trend and seasonality

- Stationarity by subtracting estimated trend and seasonality
    - Compute residuals

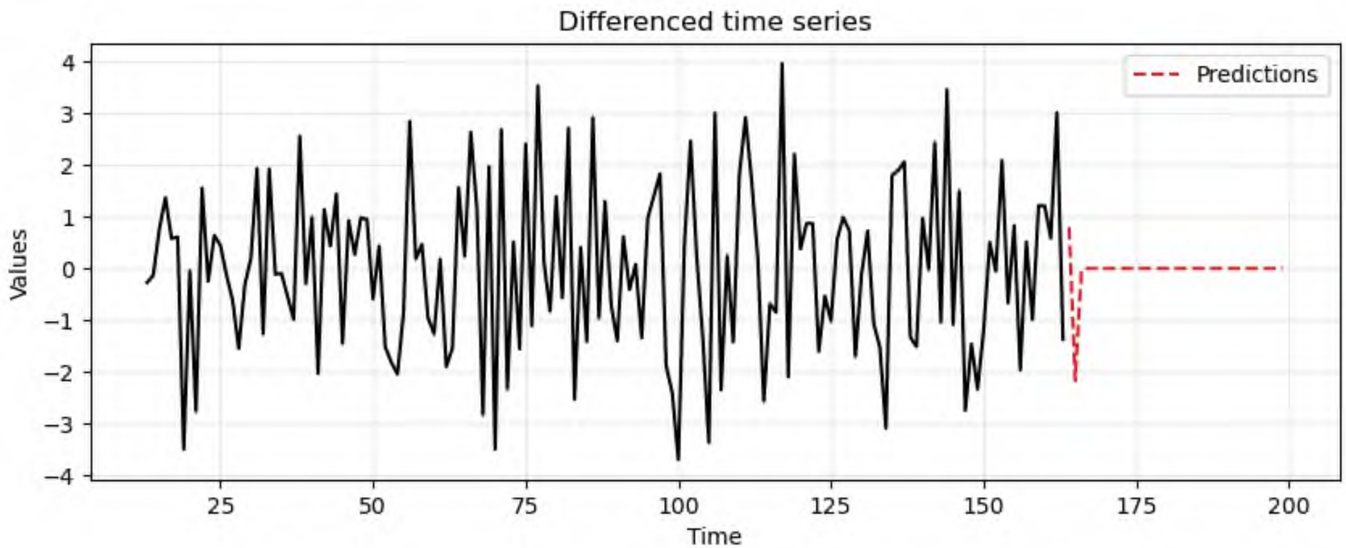
```
# Compute residuals and ACF plot of original and residual
tes_resid_ma = train_data_ma - trend_and_seasonality

_, axes = plt.subplots(1, 2, figsize=(10, 3))
plot_acf(ma_data[:len(train_data_ma)], ax=axes[0], title="ACF Original MA
process")
plot_acf(tes_resid, ax=axes[1], title="ACF of TES residuals")
plt.tight_layout();
```



- Significant lag at 2 so suggest MA model order $q = 2$

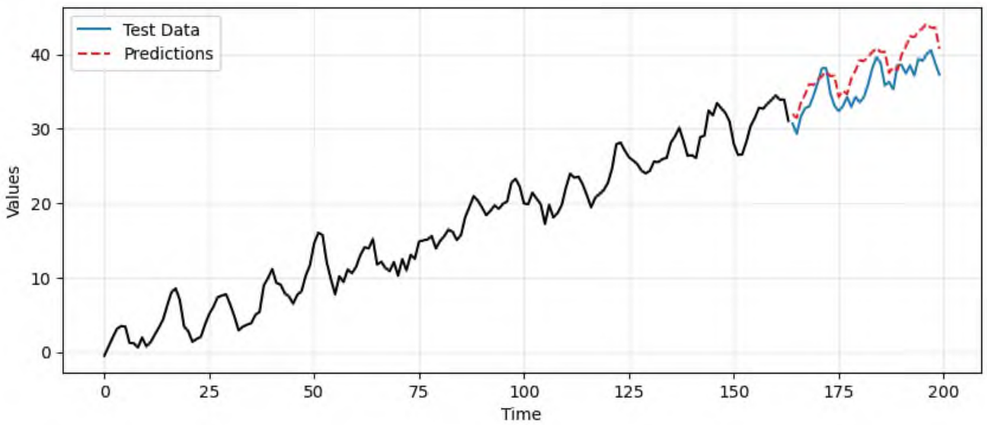
- Make predictions with differencing approach
    - Fit an MA model and compute predictions

```
# Fit the model
model = ARIMA(diff_diff_ma, order=(0,0,3))
model_fit = model.fit()

# Compute predictions
diff_preds = model_fit.forecast(steps=len(test_data_ma))

ax = run_sequence_plot(time[13:len(train_data_ma)], diff_diff_ma, "")
ax.plot(time[len(train_data_ma):], diff_preds, label='Predictions', linestyle='--
', color='tab:red')
plt.title('Differenced time series')
plt.legend();
```



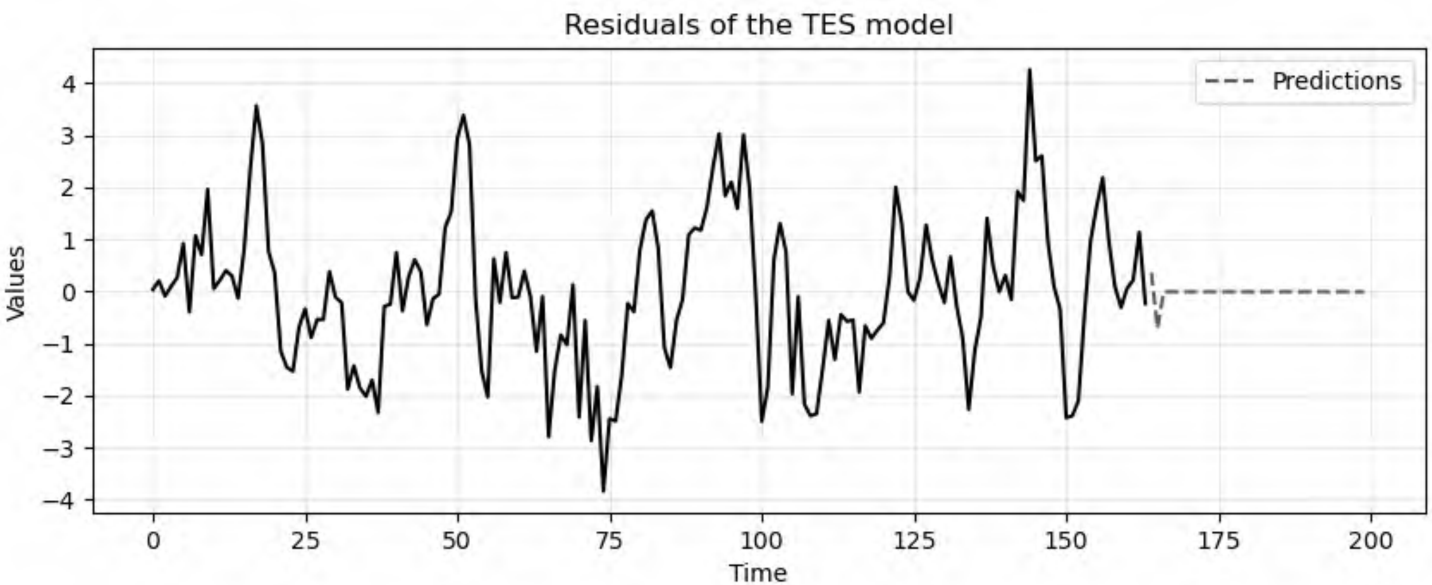
- Make predictions with either differencing/ TES approach
    - Revert the two differencing operations for final pred.

```
# Reintegrating the seasonal differencing
reintegrated_seasonal = np.zeros(len(test_data_ma))
reintegrated_seasonal[:12] = diff_ma[-12:] + diff_preds[:12]
for i in range(12, len(test_data_ma)):
    reintegrated_seasonal[i] = reintegrated_seasonal[i-12] + diff_preds[i]

# Reintegrating 1st order differencing
reintegrated = reintegrated_seasonal.cumsum() + train_data_ma[-1]

_, ax = plt.subplots(1, 1, figsize=(10, 4))
run_sequence_plot(time[:len(train_data_ma)], train_data_ma, "", ax=ax)
ax.plot(time[len(train_data_ma):], test_data_ma, label='Test Data',
color='tab:blue')
ax.plot(time[len(train_data_ma):], reintegrated, label='Predictions',
linestyle='--', color='tab:red')
plt.legend();
```



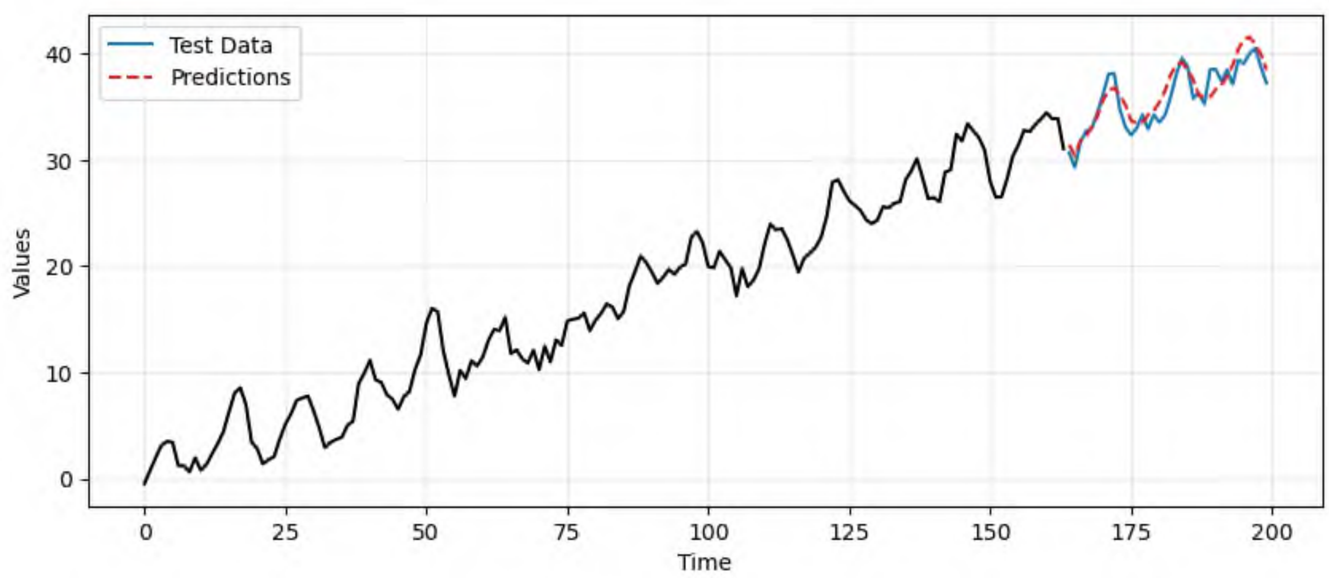
- Make predictions with TES approach
    - Fit MA model with tes_resid_ma

```
# Fit the model
model = ARIMA(tes_resid_ma, order=(0,0,2))
model_fit = model.fit() # Fit the model

resid_preds = model_fit.forecast(steps=len(test_data_ma)) # Compute predictions

ax = run_sequence_plot(time[:len(train_data_ma)], tes_resid_ma, "")
ax.plot(time[len(train_data_ma):], resid_preds, label='Predictions', linestyle='-
-', color='tab:red')
plt.title('Residuals of the TES model')
plt.legend();
```



- Make predictions with TES approach
  - Combine predictions of residuals with predictions of trend and seasonality for final predictions

```
# Add back trend and seasonality to the predictions
tes_pred = tes_ma.forecast(len(test_data_ma))
final_preds = tes_pred + resid_preds

_, ax = plt.subplots(1, 1, figsize=(10, 4))
run_sequence_plot(time[:len(train_data_ma)], train_data_ma, "", ax=ax)
ax.plot(time[len(train_data_ma):], test_data_ma, label='Test Data',
color='tab:blue')
ax.plot(time[len(train_data_ma):], final_preds, label='Predictions', linestyle='-
-', color='tab:red')
plt.legend();
```



- Compare performance of differencing and TES for MA

```
# Compute MSE for differencing and TES for MA
mse_differencing = mean_squared_error(test_data_ma, reintegrated)
mse_tes = mean_squared_error(test_data_ma, final_preds)

print(f"MSE of differencing: {mse_differencing:.2f}")
print(f"MSE of TES: {mse_tes:.2f}")
```

- MSE of differencing: 9.49
- MSE of TES: 1.69

AR Models	**MA Models**
Depend on past values of the series	Depend on past forecast errors
Suitable when past values have a direct influence on future values and for slowly changing time series	Useful when the series is better explained by shocks or random disturbances, i.e., time series with sudden changes
If the **PACF** drops sharply at a given lag p or the first lag autocorrelation is **positive**, then use an AR model with order p	If the **ACF** drops sharply at a given lag q or the first lag autocorrelation is **negative**, the use MA model with order q