Basics of Time-Series Analysis

Charalampos E. Tsourakakis
ctsourak@bu.edu

CS365 Foundations of Data Science

April 2024

Time series – ubiquitous!

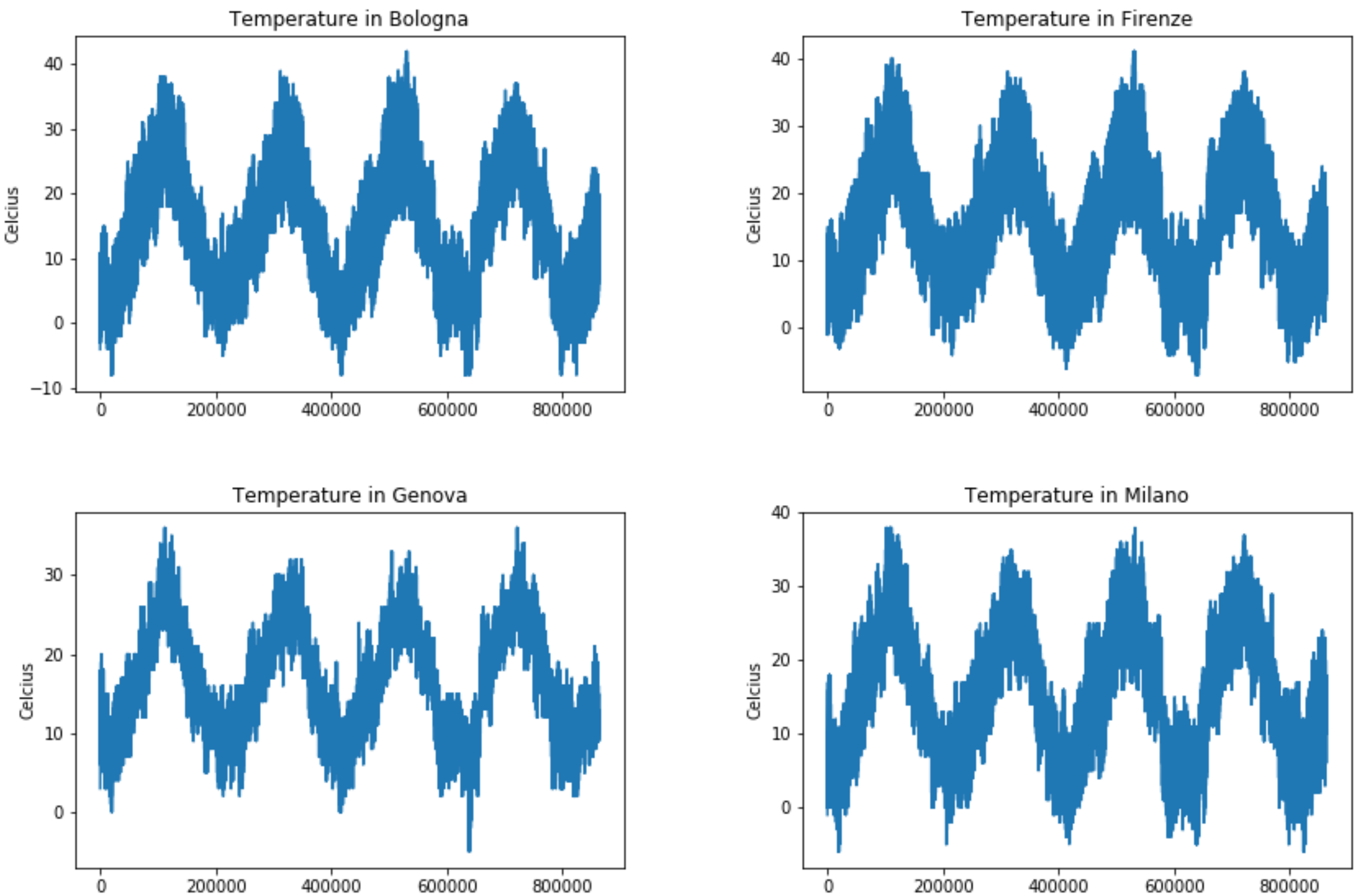
Time-series appear in any domain that involves temporal measurements.

Examples of such domains include:

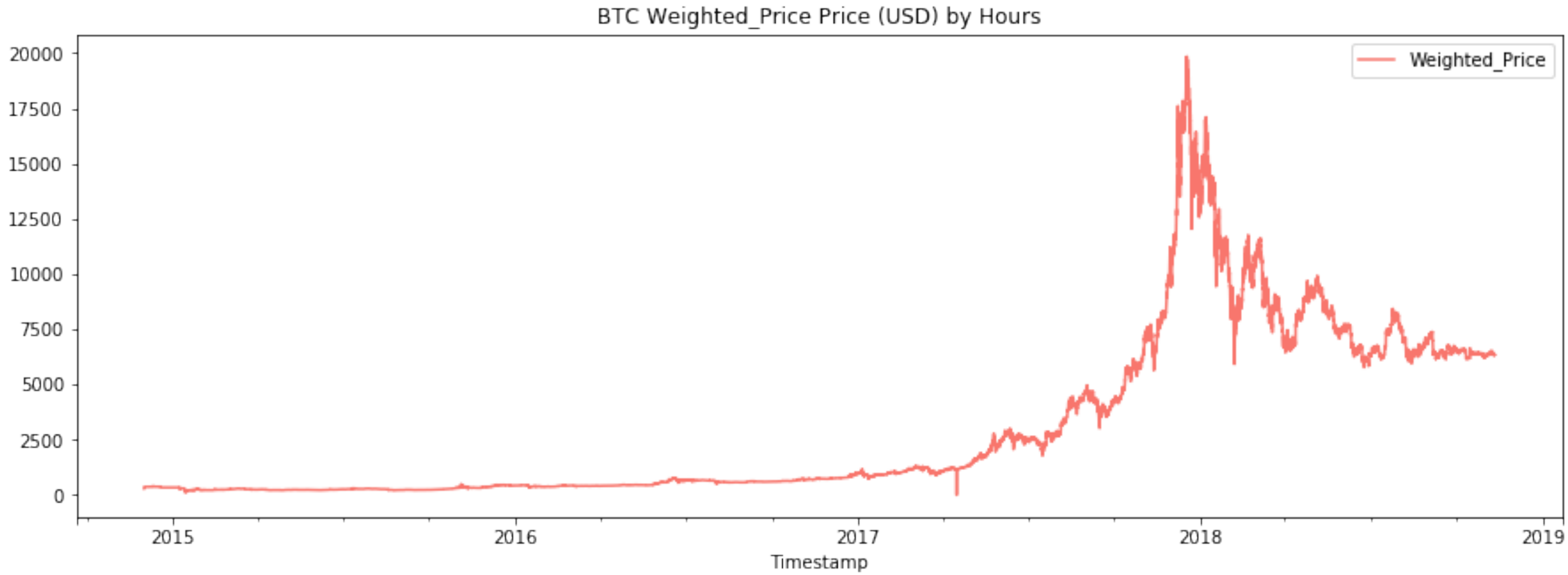
- signal processing
- econometrics
- mathematical finance
- weather forecasting
- electroencephalography
- control engineering
- astronomy
- communications engineering
- ...

Time series – examples

Temperature time series across 4 Italian cities



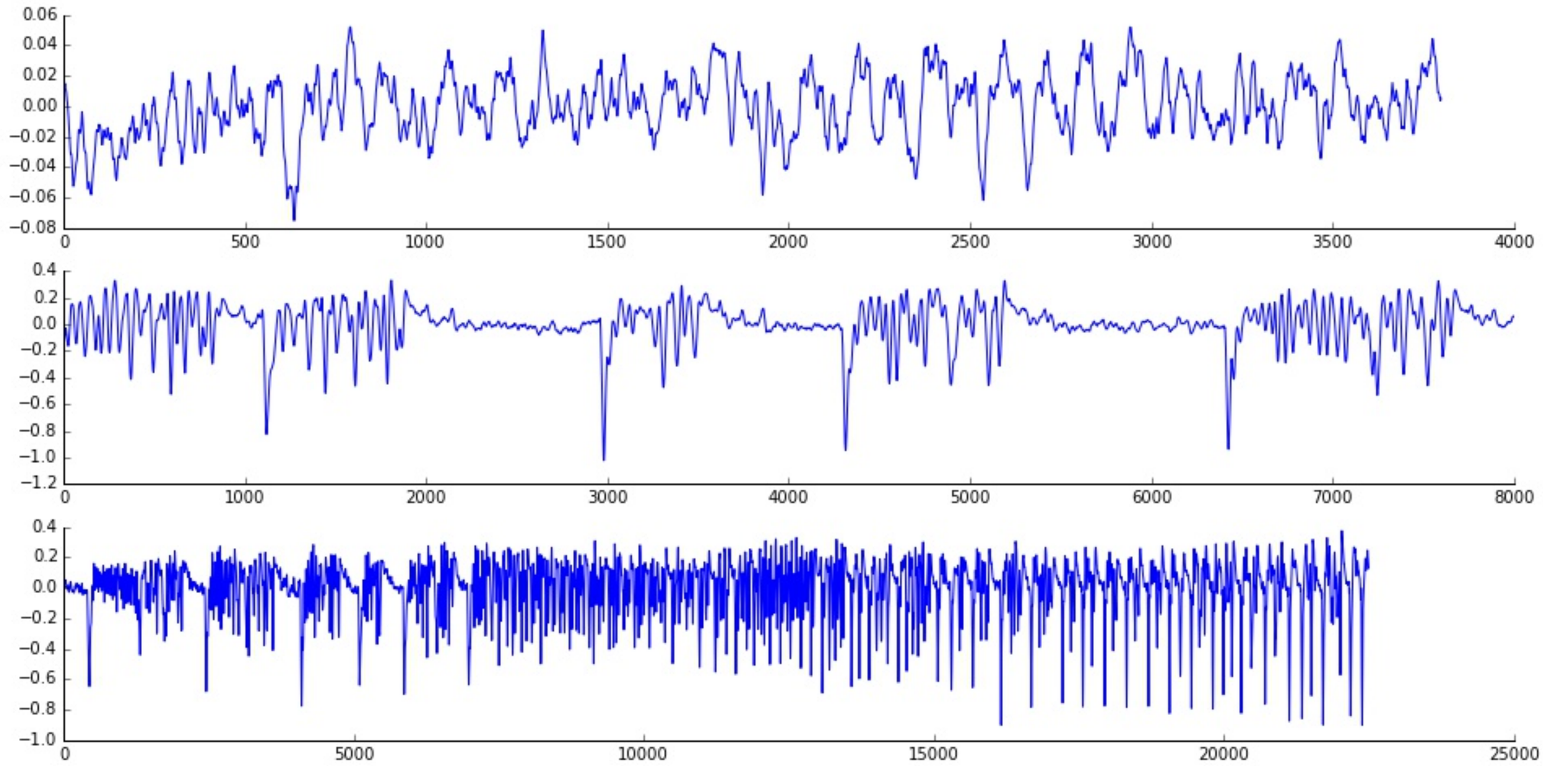
Time series – examples



Bitcoin price

- When we have a *single time-series* we wish to forecast using its past values, we call this *univariate* analysis.

Time series – examples



EEG time series from different sections of the brain

- When we have a *target time-series* we wish to forecast using its past values and other co-evolving time-series, we call this *multivariate* analysis.

Time series analysis

Why do we analyze time-series?

1. Interpretation

   E.g., seasonal adjustment

2. Forecasting

   E.g., predict stock prices

3. Control

   E.g., how does increasing VAT will affect (un)employment?

Time series analysis

4. Hypothesis testing

   E.g., should we be believers in global warming?

5. Understanding catastrophic events

   E.g., will an impactful earthquake take place tomorrow (here)?

Today's focus – Forecasting

A trend is a trend is a trend. But the question is, will it bend? Will it alter its course through some unforeseen force and come to a premature end? (Alex Cairncross)

Today's outline

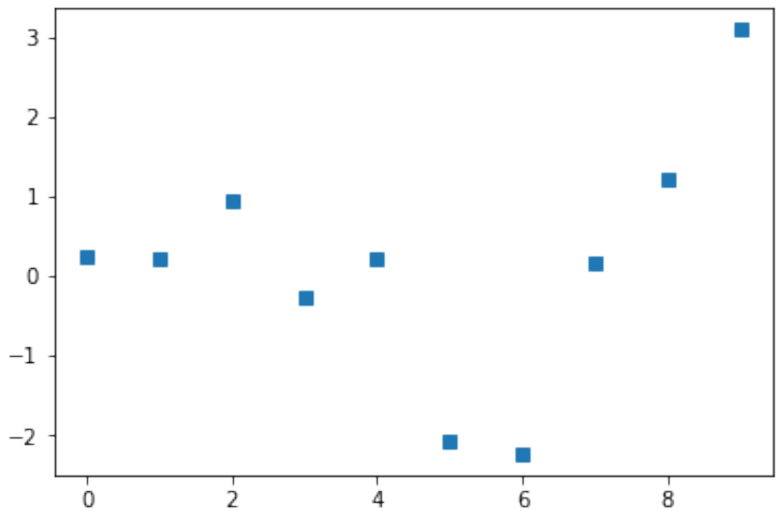
1. Part I: Fundamentals
   - Stochastic processes, strict vs weak stationarity, correlograms, partial correlograms, periodograms, autoregressive models (AR), moving average models (MA), transformations, compact time-series description.
2. Part II: Forecasting methods
   - Evaluation protocols, basic metrics, null models, spectral methods, ARMA, ARIMA, SARIMA, VAR, similarity search, deep learning (deep feedforward neural nets, recursive neural nets)

Stochastic processes – informal description

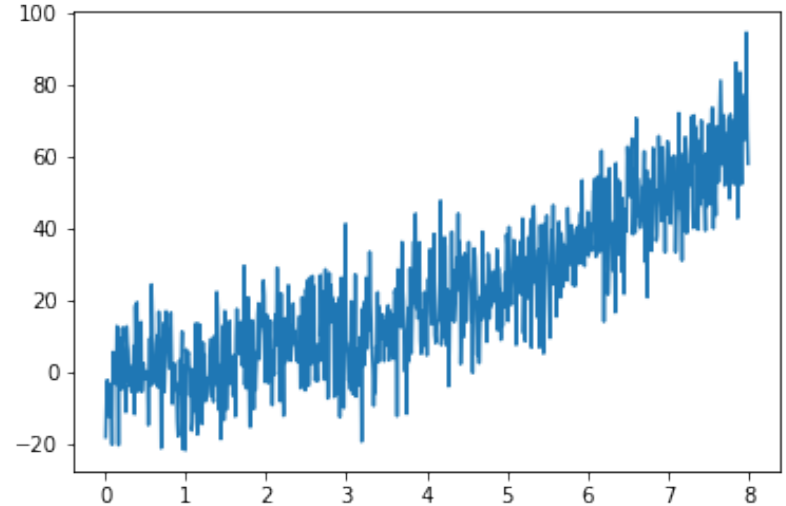
Definition. A stochastic process is a *collection of random variables indexed by time.*

(a) Discrete time stochastic processes (e.g., $X_0, X_1, X_2, \ldots$)

(b) Continuous time stochastic processes ($\{X_t\}_{t \geq 0}$)



(a)



(b)

Time-series – formal definition

- A **time series** is a stochastic process consisting of random variables indexed by time t.

- The stochastic behavior of the *time series* is determined by specifying the probability density/mass functions (pdf's)

$$p(x_{t_1}, x_{t_2}, \ldots, x_{t_m}),$$

for all finite collections of time indexes

$$\{(t_1, \ldots, t_m), m < +\infty\},$$

i.e, all finite-dimensional distributions of $\{X_t\}$.

Time-series – Stationarity

Strictly stationary time-series. A time series $\{X_t\}$ is strictly stationary if $\forall t, m, (t_1, \ldots, t_m)$

$$p(t_1 + \tau, \ldots, t_m + \tau) = p(t_1, \ldots, t_m).$$

- In other words, a time series is strictly stationary if the probability distribution is invariant under time translation.

<u>Examples</u>

(a) *iid* processes are strictly stationary.

(b) $X_t = Z_1 \cos(t) + Z_2 \sin(t)$ is strictly stationary if Z_1, Z_2 are independent normal variables.

(c) Random walks in certain types of graphs (stationary Markov chains)

Remark: Stationary time series are typically non-stationary.

Time-series – Covariance stationarity

Covariance stationary time-series. A time series $\{X_t\}$ is covariance stationary if

$$\mathbb{E}\left[X_t\right] = \mu$$

$$\mathbb{V}ar\left[X_t\right] = \sigma^2$$

$$\mathrm{Cov}[X_t, X_{t+h}] = \gamma(h)$$

Reminder: $\mathrm{Cov}[X_t, X_{t+h}] = \mathbb{E}\left[(X_t - \mu_t)(X_{t+h} - \mu_{t+h})\right].$

Basic exploration tools – auto-correlation function

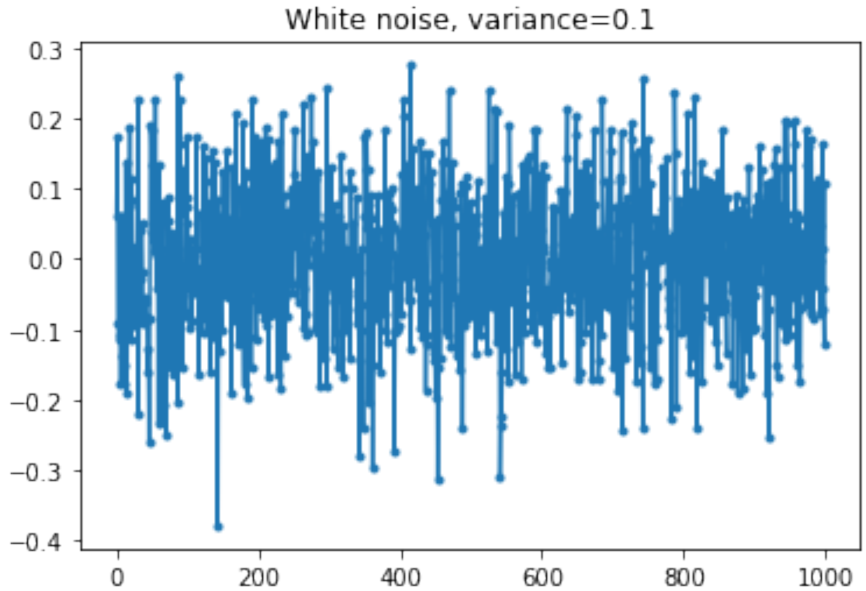
Auto-correlation function of $\{X_t\}$ is defined as

$$\begin{aligned} \rho_X(h) &= \frac{\gamma_X(h)}{\gamma_X(0)} \\ &= \frac{\mathrm{Cov}(X_{t+h}, X_t)}{\mathrm{Cov}(X_t, X_t)} \\ &= \frac{\mathrm{Cov}(X_{t+h}, X_t)}{\mathbb{V}ar\left[X_t\right]} \\ &= \mathrm{Corr}(X_{t+h}, X_t) \end{aligned}$$

Basic exploration tools – auto-correlation function

White noise: $X_t \sim WN(0, \sigma^2)$.

- $\mathbb{E}\left[X_t\right] = 0$
- $\mathbb{V}ar\left[X_t\right] = 0$
- $\mathbf{Pr}\left[X_t \leq x_t\right] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x_t} e^{-x^2/2} dx.$



Question: What is the auto-correlation function of white noise? Is it stationary?

Basic exploration tools – auto-correlation function

We have:

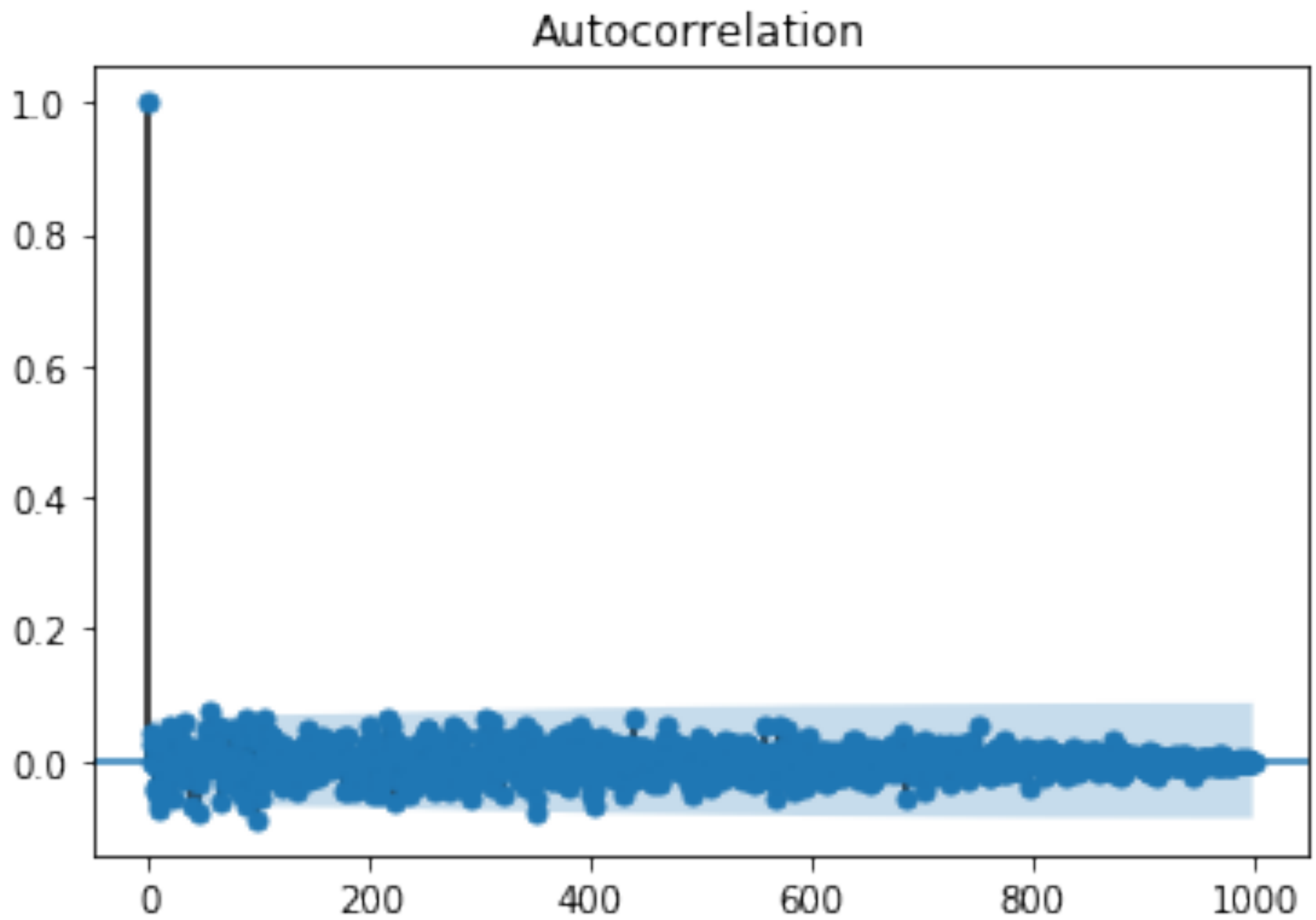
$$\gamma_X(t+h, t) = \begin{cases} \sigma^2 & h = 0 \\ 0 & h > 0 \end{cases}$$

Thus,

- $\mu_t = 0$ for all t
- $\gamma_X(t+h, ht = \gamma_X(h, 0)$ for all $t \geq 0$
- Thus $\rho_X(h) = 1$ if $h = 0$, 0 otherwise.
- question: is X_t stationary?

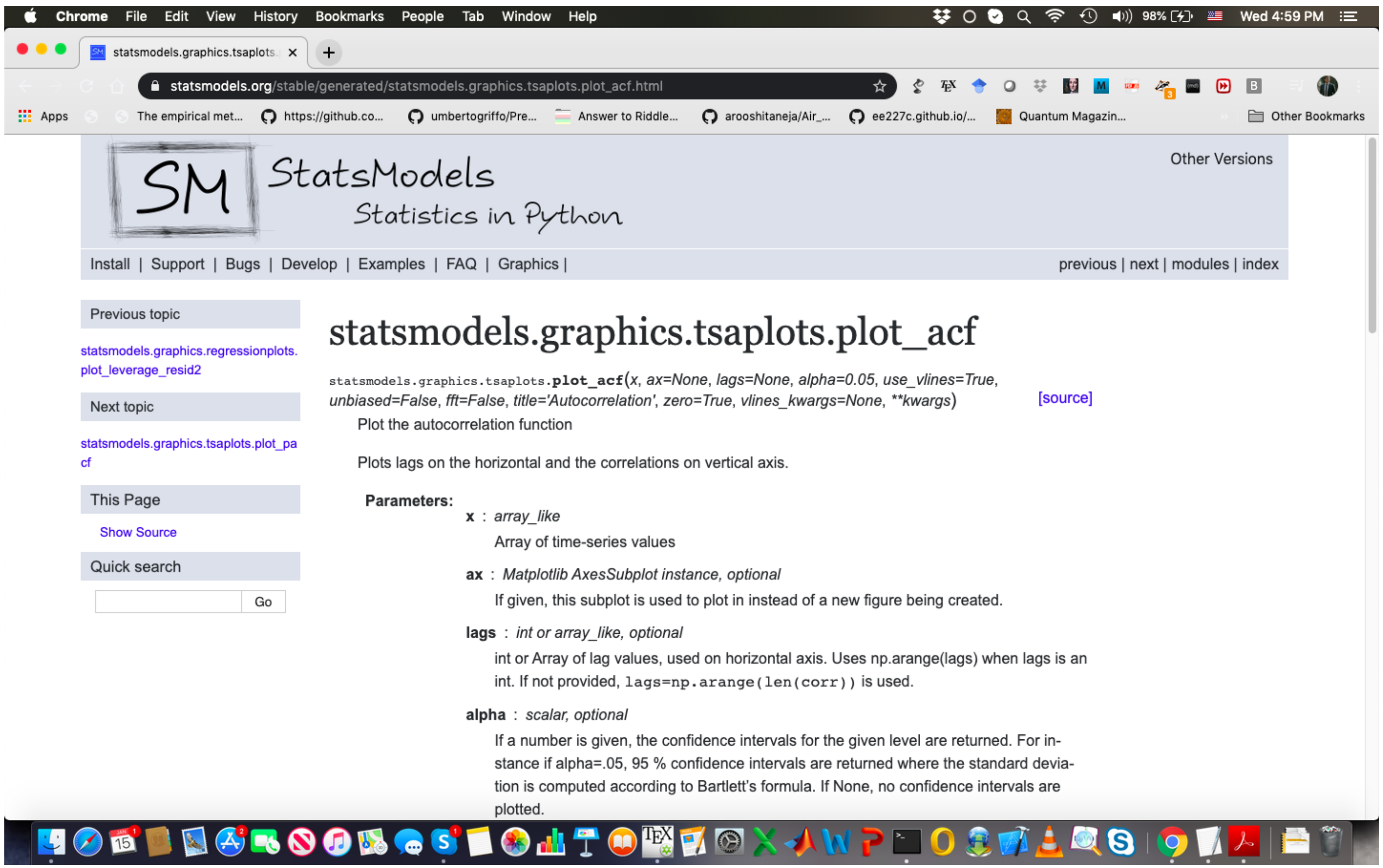
Basic exploration tools – correlogram

- The correlogram plots the auto-correlation function versus the lag.



Basic exploration tools – correlogram

Python:



StatsModels
Statistics in Python

Other Versions

Install | Support | Bugs | Develop | Examples | FAQ | Graphics |

previous | next | modules | index

Previous topic

statsmodels.graphics.regressionplots.plot_leverage_resid2

Next topic

statsmodels.graphics.tsaplots.plot_pacf

This Page

Show Source

Quick search

Go

statsmodels.graphics.tsaplots.plot_acf

`statsmodels.graphics.tsaplots.plot_acf`(*x, ax=None, lags=None, alpha=0.05, use_vlines=True, unbiased=False, fft=False, title='Autocorrelation', zero=True, vlines_kwargs=None, **kwargs*) [source]

Plot the autocorrelation function

Plots lags on the horizontal and the correlations on vertical axis.

Parameters:

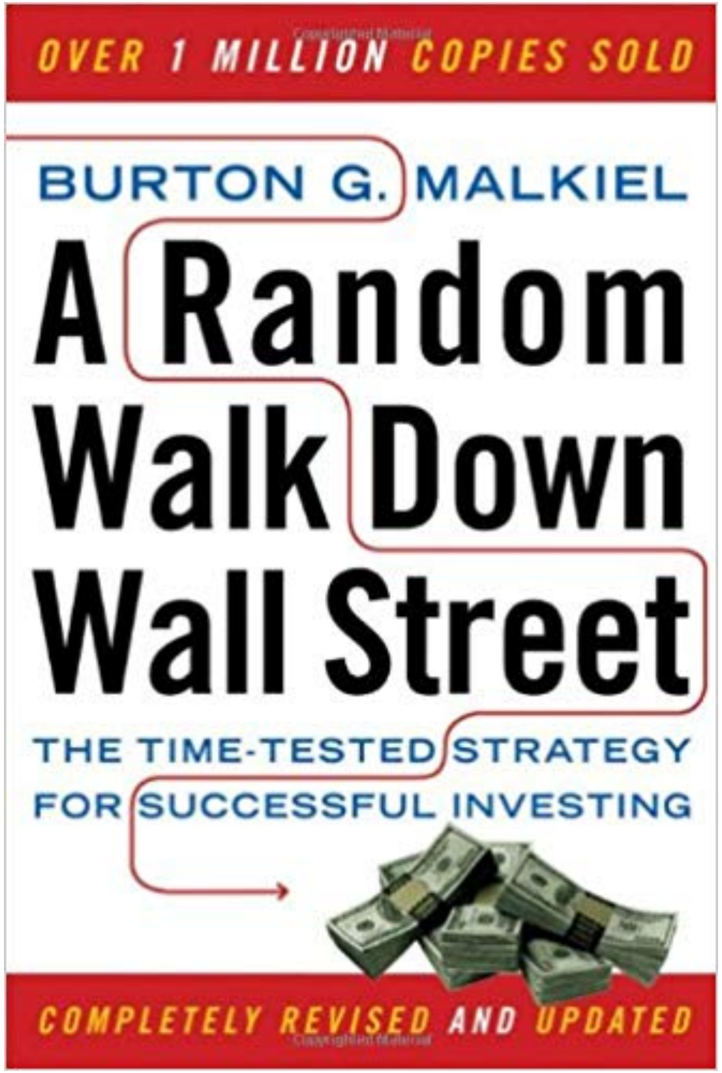
x : *array_like*
Array of time-series values

ax : *Matplotlib AxesSubplot instance, optional*
If given, this subplot is used to plot in instead of a new figure being created.

lags : *int or array_like, optional*
int or Array of lag values, used on horizontal axis. Uses np.arange(lags) when lags is an int. If not provided, `lags=np.arange(len(corr))` is used.

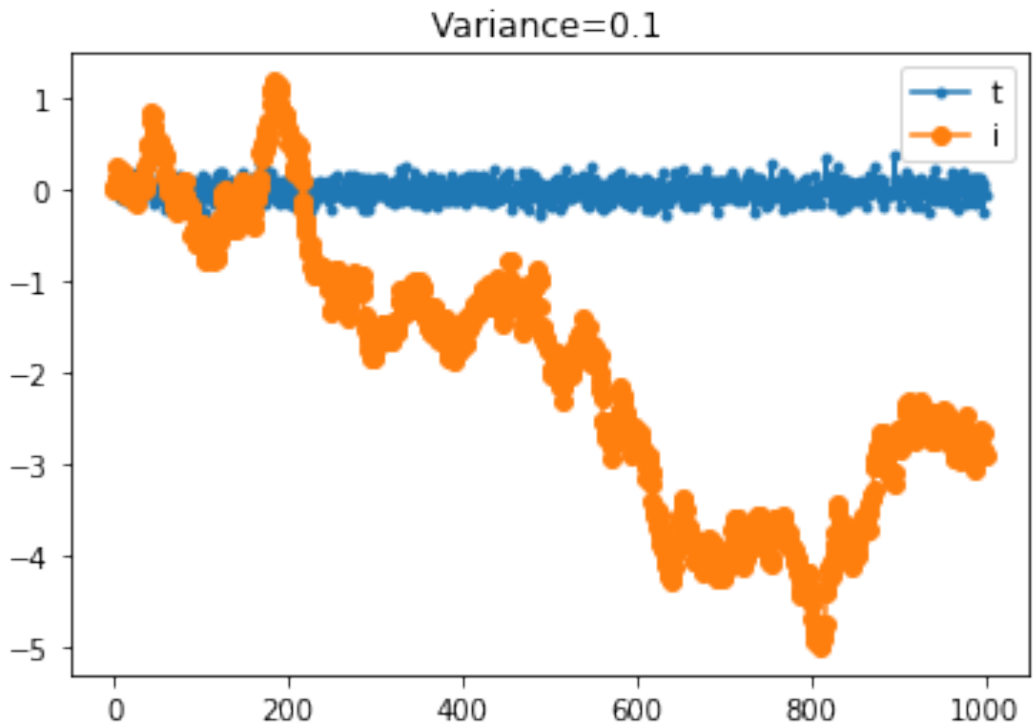
alpha : *scalar, optional*
If a number is given, the confidence intervals for the given level are returned. For instance if alpha=.05, 95 % confidence intervals are returned where the standard deviation is computed according to Bartlett's formula. If None, no confidence intervals are plotted.

What is a random walk time series?



Random walk

Random walk: $S_t = \sum_{i=1}^{t} X_i$ for $X_i \sim WN(0, \sigma^2)$.



question: is a random walk stationary or not?

Random walk is not stationary

- $\mathbb{E}[S_t] = 0$
- However, the covariance function is a function of t:

$$
\begin{aligned}
\gamma_S(t+h, t) &= \mathrm{Cov}(S_{t+h}, S_t) \\
&= \mathrm{Cov}(S_t + \sum_{s=1}^{h} X_{t+s}, S_t) \\
&= \mathrm{Cov}(S_t, S_t) = \mathbb{V}ar[S_t] = t\sigma^2.
\end{aligned}
$$

Therefore, S_t is **not stationary**.

Moving average process $MA(1)$

We define the moving average process of order 1 $MA(1)$ as

$$X_t = Z_t + \theta Z_{t-1}, \{Z(t)\} \sim WN(0, \sigma^2).$$

- We have $\mathbb{E}[X_t] = 0$ and

$$\begin{aligned}\gamma_X(t+h, t) &= \mathbb{E}[X_{t+h} X_t] \\ &= \mathbb{E}[(Z_{t+h} + \theta Z_{t+h-1})(Z_t + \theta Z_{t-1})] \rightarrow\end{aligned}$$

$$\gamma_X(t+h, t) = \begin{cases} \sigma^2(1+\theta^2) & \text{if } h = 0 \\ \sigma^2\theta & \text{if } h = 1 \\ 0 & \text{otherwise} \end{cases}$$

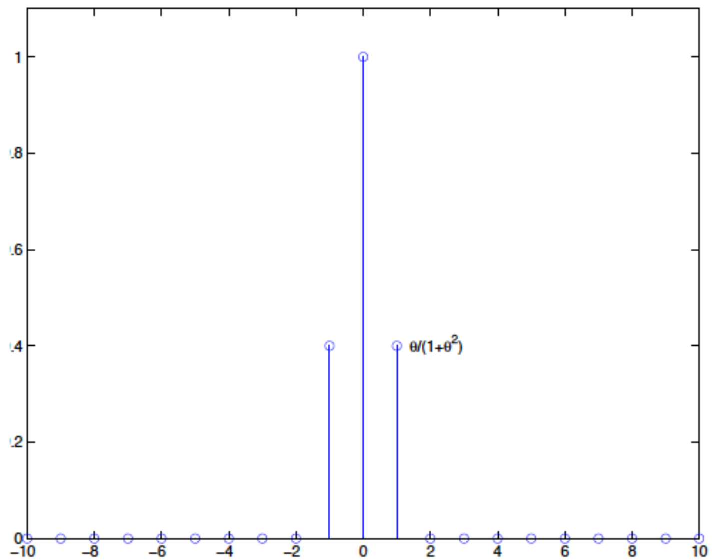
Moving average process $MA(1)$

- As we observe $MA(1)$ is stationary.
  - $\mathbb{E}[X_t] = 0$
  - $\gamma_X(t+h,t) = \gamma_X(h,0)$ is
- How *can we use it*?
  - $\Delta\text{Icecream}_t = \Delta\text{temperature}_t - 0.9\Delta\text{temperature}_{t-1}$,
- We assume that the changes in the temperature are iid normal random variables.

- **Important observation:** We model differences, not the actual time series values. This is a common time-series transformation that we use frequently in practice.

Moving average process $MA(1)$

The correlogram of ρ_X of an $MA(1)$ process:



- Moving average processes generalize to order q $MA(q)$ processes

$$X_t = \mu + Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q}.$$

- The autocorrelation function of an $MA(q)$ process is zero at lag $q+1$ and greater.

Autoregressive process $AR(1)$

Mathematically an $AR(1)$ process is defined as

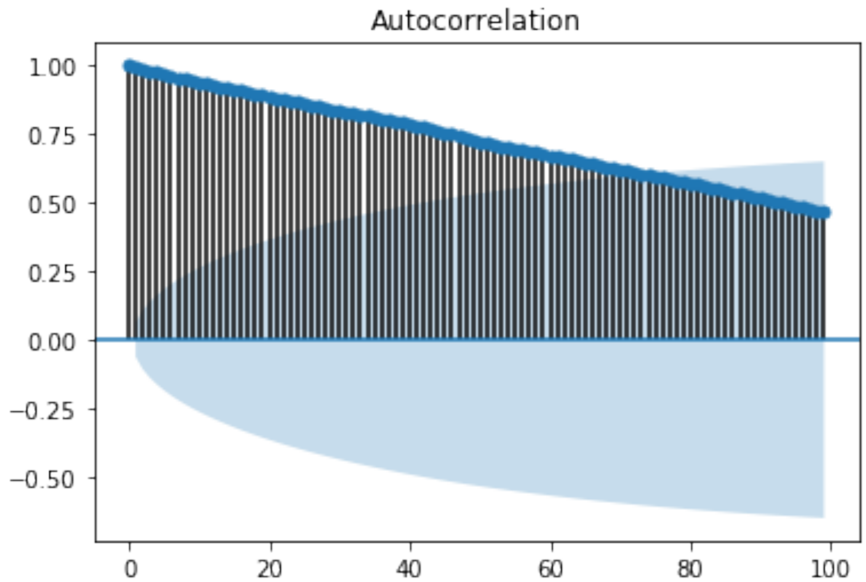
$$X_t = \rho X_{t-1} + Z_t, Z_t \sim WN(0, \sigma^2).$$

- In general, the order p autoregressive process is defined similarly as:

$$X_t = c + \rho_1 X_{t-1} + \ldots + \rho_p X_{t-p} + Z_t, Z_t \sim WN(0, \sigma^2).$$

- Exercise
  1. $\mathbb{E}[X_t] = \frac{c}{1-\phi}$
  2. $\gamma_X(t+h, t) = \frac{\sigma^2}{1-\phi^2}\phi^h$.

Autoregressive process $AR(1)$



- In contrast to moving average processes, the ACF plot cannot tell us (at least by inspection) the order p (here, $p = 1$).
- Still though, we will show another tool, the partial correlogram plot that allows us to get an idea of the order of the AR process.

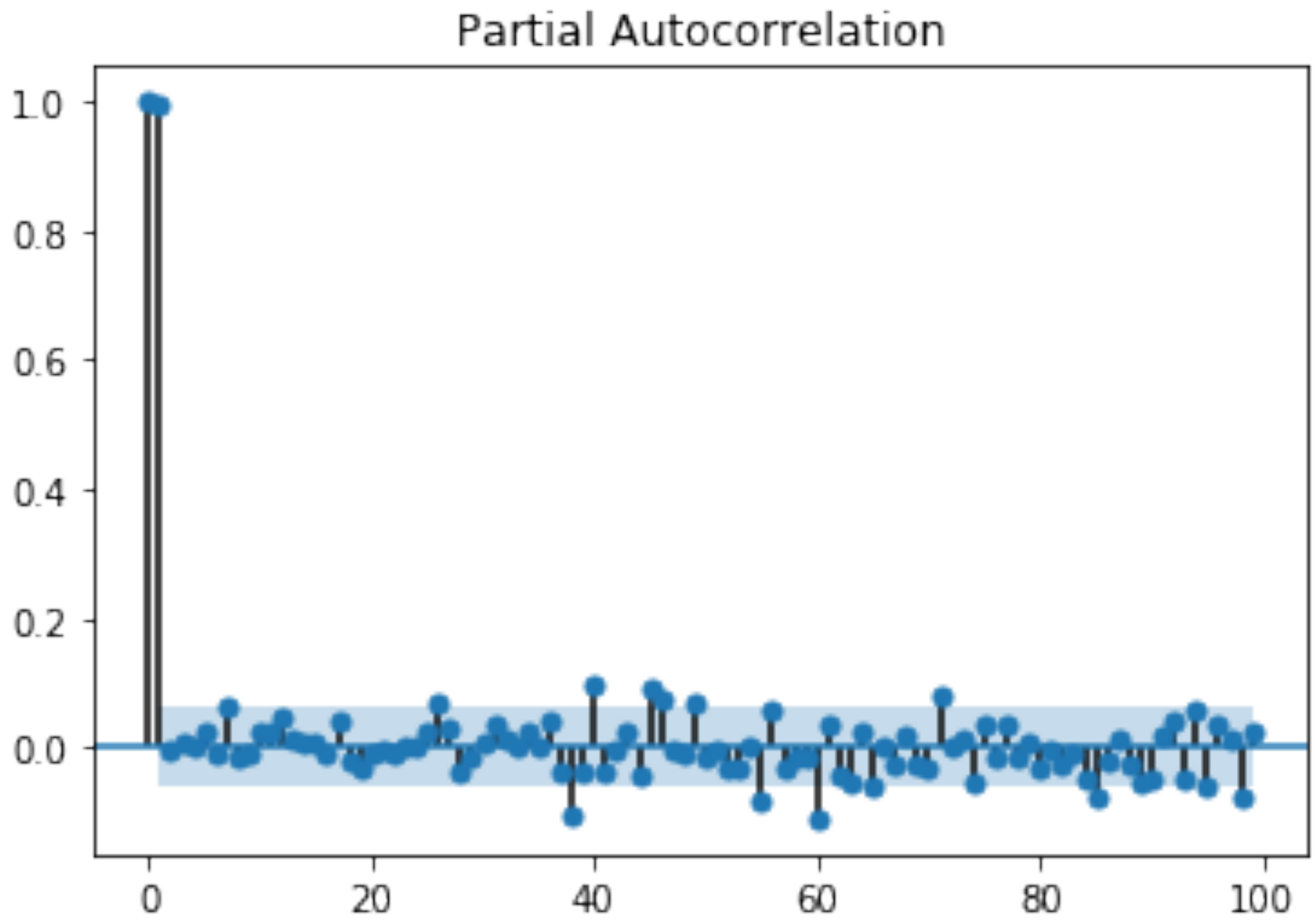
Basic exploration tools - partial correlogram

- To understand the necessity of another tool that is used *jointly* with the correlogram we will give an example.

$$x(t+2) = x(t+1) + \epsilon(t+1) = x(t) + \epsilon(t) + \epsilon(t+1).$$

- Thus $x(t+2)$ and $x(t)$ are only related because of $x(t+1)$ in between.
- The idea of the partial correlogram is to measure the correlation of $x(t+h)$ and $x(t)$ after removing linear relationship due to values in between.

Partial correlogram $AR(1)$



Estimating the ACF

- In reality, we have only access to the data.
- Estimating the mean:

$$\bar{x} = \frac{1}{n} \sum_{t=1}^{n} x_t.$$

- Estimating autocovariance function:

$$\hat{\gamma}(h) = \frac{1}{n} \sum_{t=1}^{n-h} (x_{t+h} - \bar{x}) \cdot (x_t - \bar{x}).$$

- Estimating autocorrelation function:

$$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}.$$

Stationarity statistical tests

- Given the data, we can test whether the time series (or the differences, or other transformations) is stationary or not.
    - Dickey-Fuller test (unit root test)
    - Kwiatkowski-Phillips-Schmidt-Shin test
- The KPSS test can test *trend-stationarity*
- Dickey-Fuller and its variations are popular.
    - The Dickey–Fuller test:
    - formally, it tests the null hypothesis that a unit root is present in an autoregressive model.
    - Alternative hypothesis: time-series is stationary

Dickey-Fuller test in Python

```python
from statsmodels.tsa.stattools import adfuller


def Dickey_Fuller_test(timeseries):
    '''
    Test stationarity
    '''
    #Perform Dickey-Fuller test:
    print( 'Results of Dickey-Fuller Test:')
    dftest = sm.tsa.stattools.adfuller(timeseries)
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

How to interpret the Dickey-Fuller test?

- Null Hypothesis: time series has a unit root, meaning it is non-stationary.
- Alternative Hypothesis : time-series is stationary
- Code returns:

```
Results of Dickey-Fuller Test:
Test Statistic                        -1.102293e+01
p-value                                5.925157e-20
Number of Observations Used            9.900000e+01
Critical Value (5%)                   -2.891208e+00
Critical Value (1%)                   -3.498198e+00
Critical Value (10%)                  -2.582596e+00
```

- Check the value of the statistic and compare it the critical values.
- If it is less than the critical value, then we can reject the null hypothesis with that level of confidence.
- Also p-values are informative: Rule of thumb:
  - If p-value$>$ 0.05, then time-series is non-stationary.

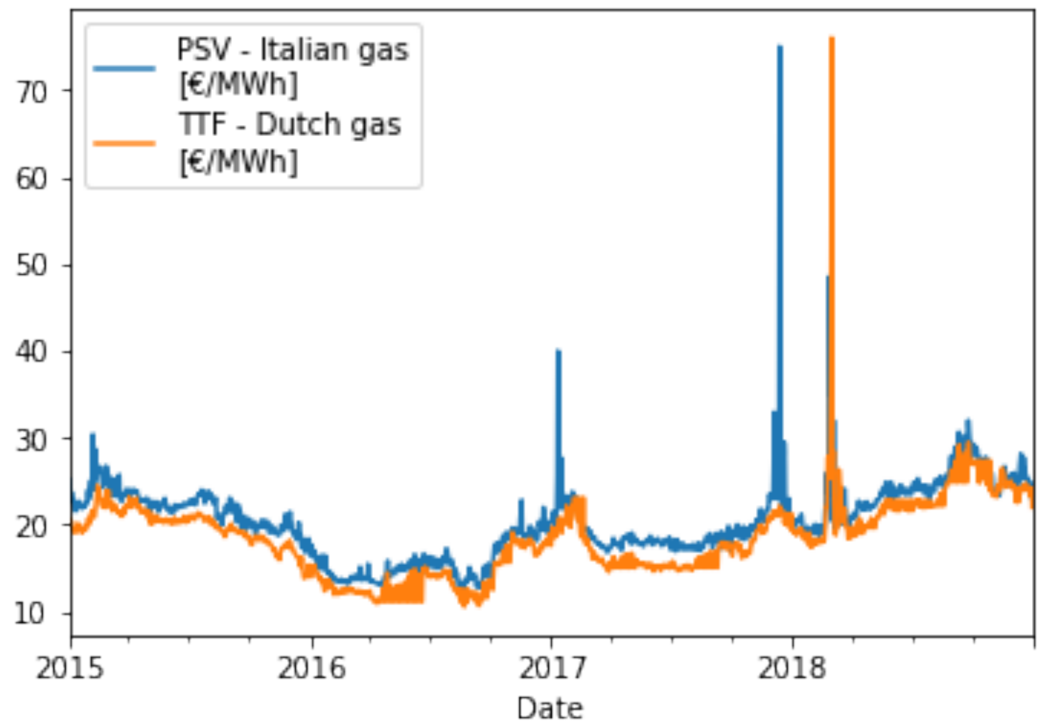
Kwiatkowski-Phillips-Schmidt-Shin test in Python

```
def kpss_test(timeseries):
  print ('Results of KPSS Test:')
  kpsstest = tsa.stattools.kpss(timeseries)
  kpss_output = pd.Series(kpsstest[0:2], index
=['Test Statistic','p-value'])
  for key,value in kpsstest[2].items():
      kpss_output['Critical Value (%s)'%key] =
  value
  print (kpss_output)
```

Cointegration – Stationarity in multivariate time series

- When we have multiple co-evolving time-series, there is the notion of cointegration.
- For example:
  - D_t: position of a dog at time t
  - P_t: position of the person who takes the dog for a walk at time t
- While D_t, P_t may look individually random, their difference is not.
- Question: does there exist a constant c such that $P_t - cD_t$ is stationary?

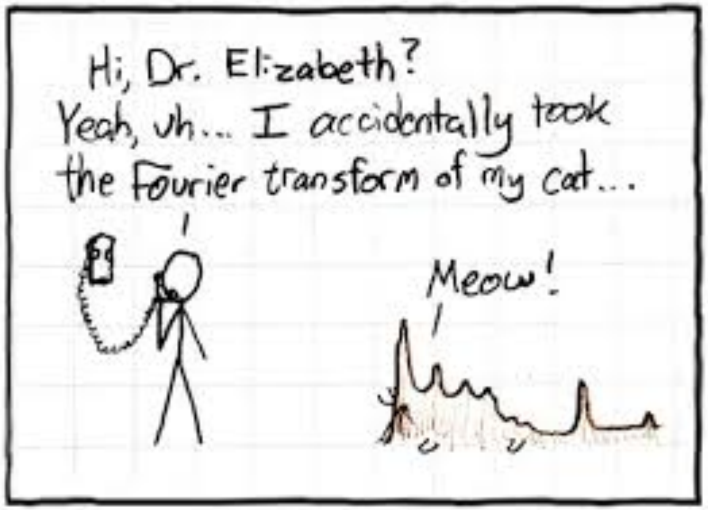
Cointegration – Stationarity in multivariate time series




```python
from statsmodels.tsa.stattools import coint
coint(P,Q)
```

Fourier analysis - basics

Joseph Fourier showed how to represent a periodic function as a sum of trigonometric functions (oscillations).



Fourier analysis - basics

$$x(t) - x\bar{(t)} = \sum_{k=1}^{n} \gamma_k \Big(a_k \cos(2\pi t \omega_k) + b_k \sin(2\pi t \omega_k) \Big).$$

where $\omega_k = \frac{k}{n}$. If we define:

- $C_k = \sqrt{a_k^2 + b_k^2}$
- $\phi_k = \arctan \frac{b_k}{a_k}$

then we can also write the formula also as (trigonometric identity):

$$x(t) - x\bar{(t)} = \sum_{k} \gamma_k C_k \cos(2\pi t \omega_k - \phi_k).$$

Fourier analysis - sinusoidal decomposition

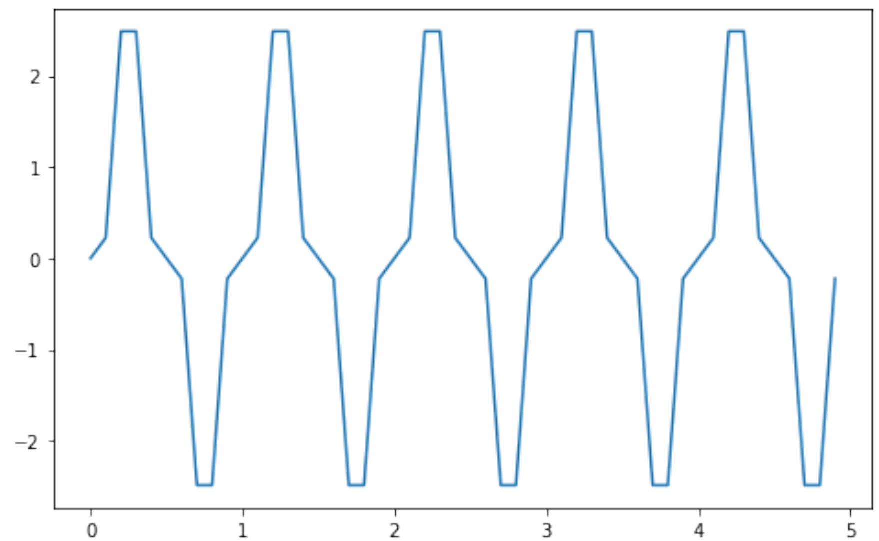
- We prefer to express things in *complex numbers*.
- Trigonometric identities are trivial. E.g.,

$$\begin{aligned}
\sin(x)\cos(x) &= \frac{1}{2i}(e^{ix} - e^{-ix})\frac{1}{2}(e^{ix} + e^{-ix}) = \\
&= \frac{1}{4i}(e^{i2x} - 1 + 1 - e^{-2ix}) \\
&= \frac{1}{2}\frac{1}{2i}e^{i2x} - e^{-2ix}) = \frac{1}{2}\sin(2x).
\end{aligned}$$

Periodogram

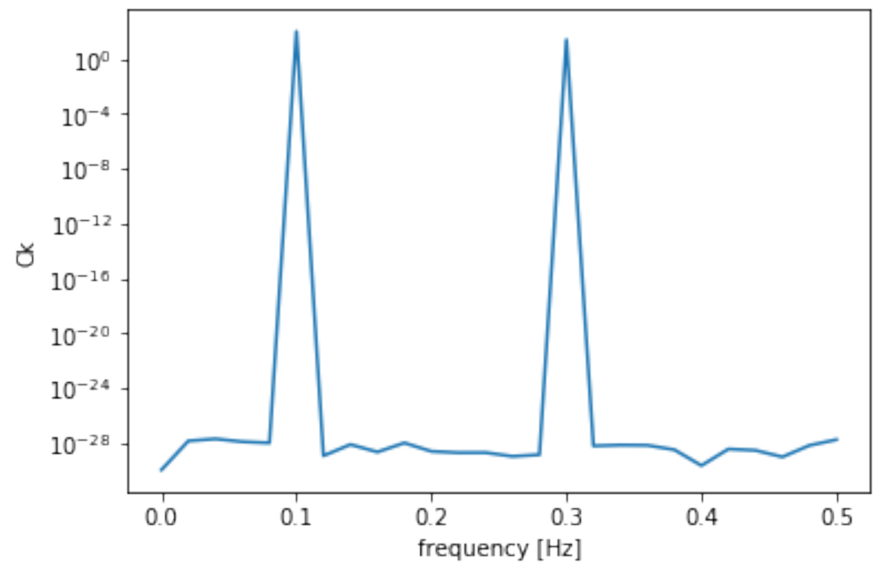
- A plot of nC_k^2 versus the frequency ω_k for $k = 1, 2, \ldots$ is called the periodogram of the dataset.
- The periodogram is a very useful tool.
- Random-signals that lack structure have all sinusoids with equal importance.
- If a time series has a strong sinusoidal signal for some frequency, then there will be a peak in the periodogram at that frequency.
- If a large C_k value appears at ω_k and other large values $C_{\ell k}$ appear at multiples $\ell\omega_k$ where $\ell = 2, ..$ it suggests that there is a non-sinuisodal signal with that specific frequency.

Periodogram – example




```
sig =  np.sin(2 * np.pi * f1*time_vec )  + 2* np.sin(2 * np.pi * f2*time_vec )
f, Pxx_den = signal.periodogram(sig)
plt.semilogy(f, Pxx_den)
plt.show()
```

Periodogram



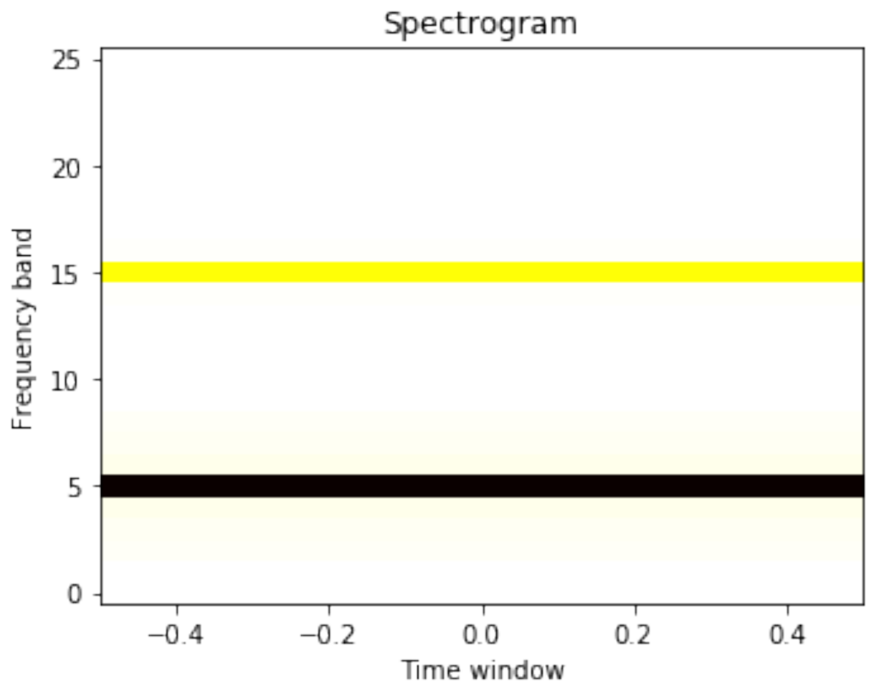
- Python plots

$$\hat{f}(\omega) = \frac{1}{n} \left| \sum_{t=1}^{n} x(t) e^{2\pi i t \omega} \right|, \omega \in [0, 0.5].$$

- Symmetry $\hat{f}(\omega) = \hat{f}(1-\omega)$

Spectrogram

- **Which frequencies appear across time?**
- Spectrogram!

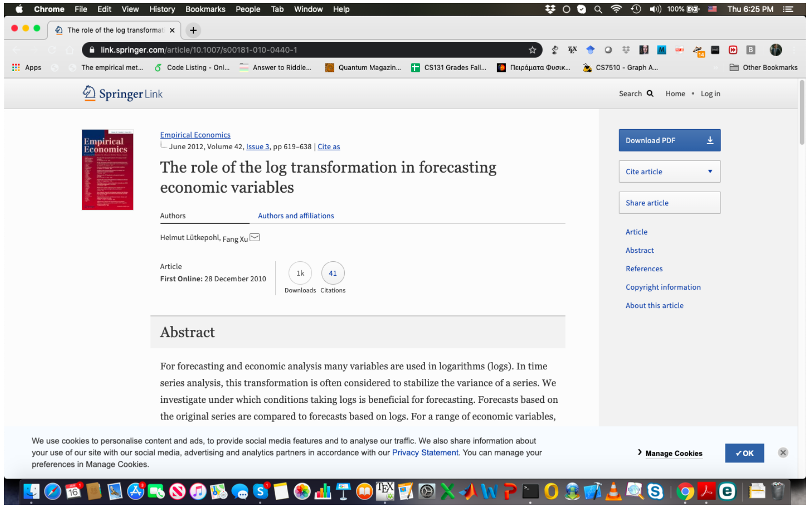


Spectrogram – Python snippet

```
from scipy import signal
freqs, times, spectrogram = signal.spectrogram(
    sig)

plt.figure(figsize=(5, 4))
plt.imshow(spectrogram, aspect='auto', cmap='
    hot_r', origin='lower')
plt.title('Spectrogram')
plt.ylabel('Frequency band')
plt.xlabel('Time window')
plt.tight_layout()
```

Time-series transformations

- Earlier we saw that a random walk time series is not stationary. Nonetheless, the increments form a stationary time series (white noise).
- Therefore, *transforming* the time-series **can** give a different structured signal
- Goal: Transform the signal so that hopefully it is easier to model mathematically (and hence better forecasts).



Time-series transformations - Box-Cox

- A Box-Cox transformation is a way to stabilize the variance of a time series with non-negative values.
- If negative values are present, we use the Yeo-Johnson power transform.
- Box-Cox function is is invertible.
- Working with the Box-Cox transformation of financial time series frequently helps (even slightly).
- There is a parameter λ that defines the Box-Cox transformation of a measurement y as follows:

$$f(y, \lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log y & \text{if } \lambda = 0 \end{cases}$$

Remark

- An example of how much a transformation can help the prediction on the gas prices in Italy.

```
Box-Cox transformation
---------------------------
Absolute error:12.727407888799966
---------------------------

No preprocessing
---------------------------
Absolute error:23.151990039645284
---------------------------
```

Time-series transformations - Box-Cox

```
def box_cox(y):
    if not isinstance(y, pd.Series):
        y = pd.Series(y)
        y.astype(float)
    y_boxcox, lmbda = stats.boxcox(y)
    return y_boxcox, lmbda


def invboxcox(y, lmbda):
    if not isinstance(y, pd.Series):
        y = pd.Series(y)
        y.astype(float)

    if lmbda == 0:
        return(np.exp(y))
    else:
        return(np.exp(np.log(lmbda*y+1)/lmbda))
```

Time-series transformations - Differencing

- First order differencing: convert time series $x(1), x(2), \ldots, x(n)$ to $x(2) - x(1), \ldots, x(n) - x(n-1)$.

```python
def first_order_diff(y):
    if not isinstance(y, pd.Series):
        y = pd.Series(y)
        y.astype(float)
    z = y.diff(1).dropna()
    return z



def first_order_diff_numpy(data):
  return [data[i] - data[i - 1] for i in range
    (1, len(data))]
```

Time-series transformations - Differencing (variations)

- First order differencing: sometimes when we assume increments have seasonality we may do the following:

  E.g., weekly seasonality on daily data, then we convert time series $x(1), x(2), \ldots, x(n)$ to $x(7) - x(1), x(8) - x(2), \ldots, x(n) - x(n-7)$

```
def difference(data, gap):
  return [data[i] - data[i - gap] for i in range(gap, len(data))]
```

- Another variation: second order differencing, i.e., differences on the differences (third order etc.)

Time-series transformations - Logarithm

- The log-transformation of a time-series under certain mathematical modeling assumptions provably improves the accuracy of predictors.
- In practice, where the distribution that generates the time-series is not known, we use it, and evaluate it empirically.

```
def log_(y):
    ''' The logarithm of the time series (always apply on non-negative values!)'''
    if not isinstance(y, pd.Series):
        y = pd.Series(y)
        y.astype(float)
    log_y = np.log(y)
    return log_y

```

Time-series transformations - Log-returns

- Differences on the log results in what is known as log-returns.
- Transforms series $(x_1, ..., x_n)$ to $\log(x_2/x_1), \ldots, \log(x_n/x_{n-1})$

```python
def log_returns(y):
    if not isinstance(y, pd.Series):
        y = pd.Series(y)
        y.astype(float)
    log_returns = np.log(y/y.shift(1)).dropna()
    return log_returns

```

Time-series transformations - Standarization

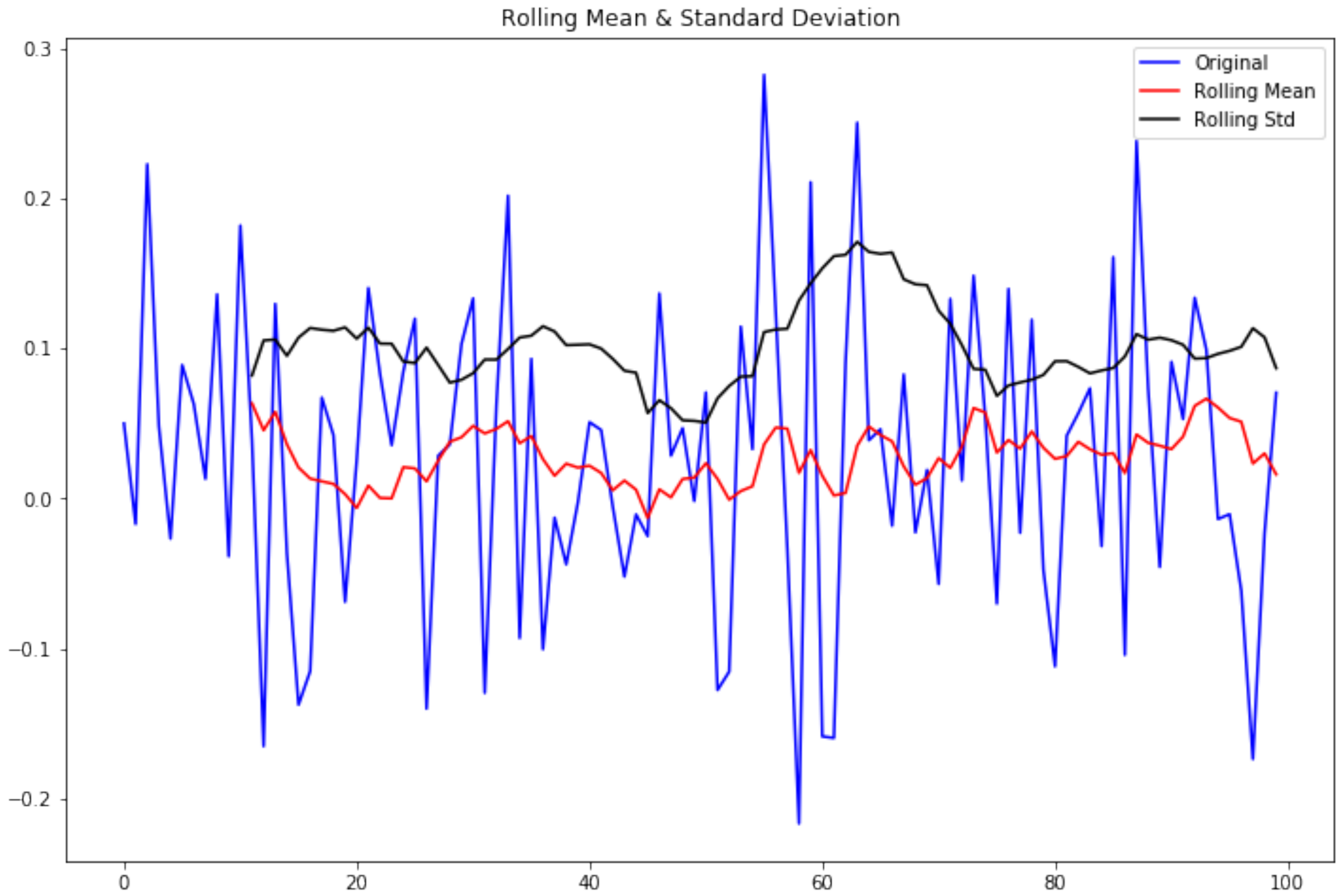
- Standarization is especially important when we have multiple time series, with different types of measurements.
- There exist two popular standarization methods:

$$x_i' = \frac{x_i - \min_{1 \leq j \leq n}(x_j)}{\max_{1 \leq j \leq n}(x) - \min_{1 \leq j \leq n}(x)},$$

and

$$x_i' = \frac{x_i - \bar{x}}{\sigma(x)},$$

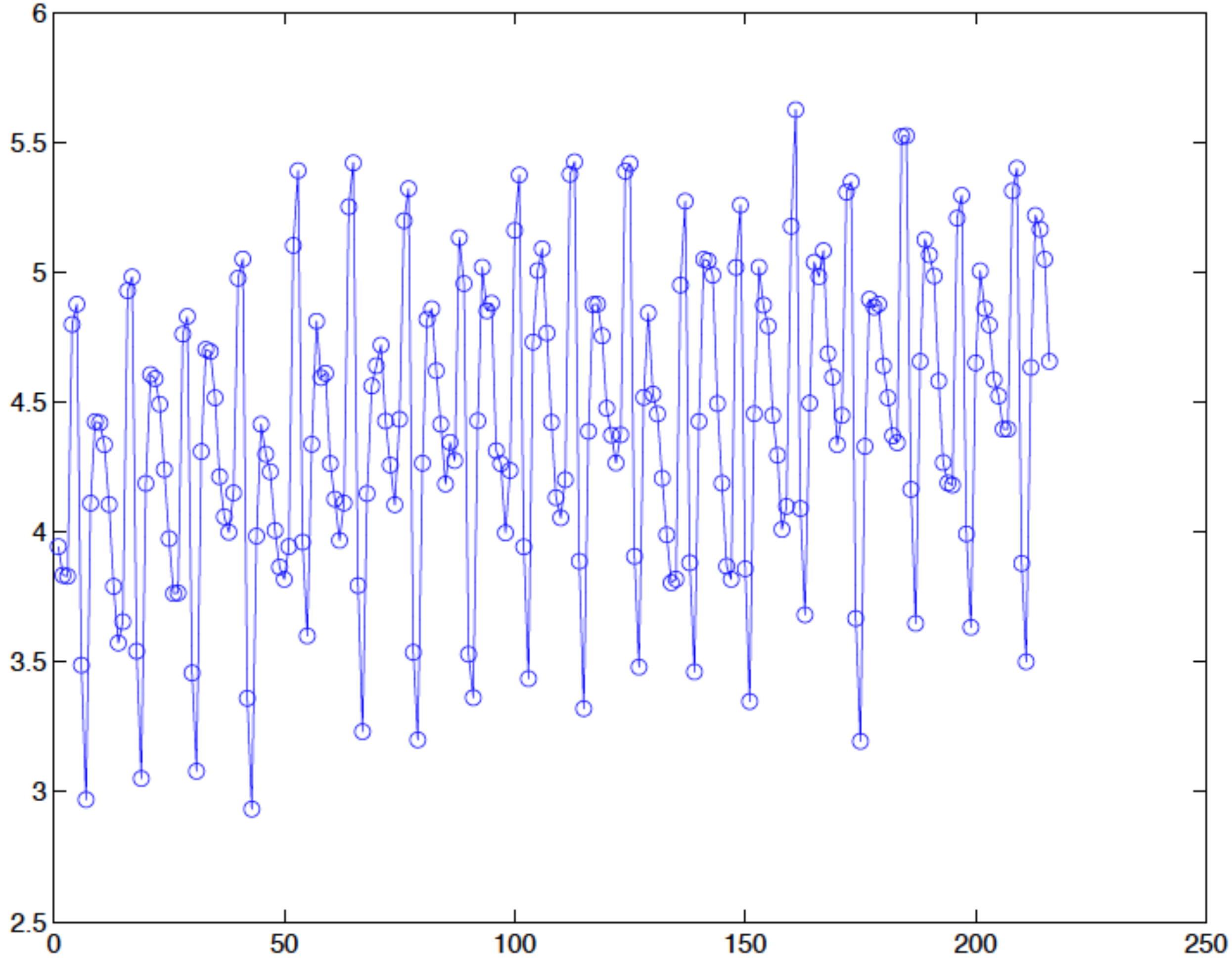
Time-series transformations - Rolling mean



Compact time-series description

$$\boxed{X(t) = \text{trend}(t) + \text{seasonal}(t) + \text{residual}(t)}$$

- Example: Suppose we have the following time-series:

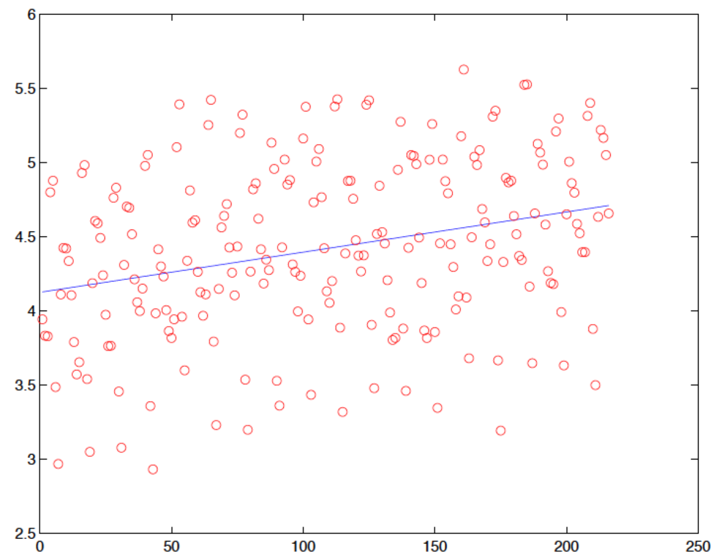


Compact time-series description

- In terms of equations, (roughly) we perform two types of regression, and obtain residuals

$$\boxed{X(t) = \text{polynomial}(t) + \sum_i \left(\beta_i \cos(\lambda_i t) + \gamma_i \sin(\lambda_i t)\right) + R_t}$$

- First, we perform regression: $X(t) = \alpha_0 + \alpha_i t + E(t)$:

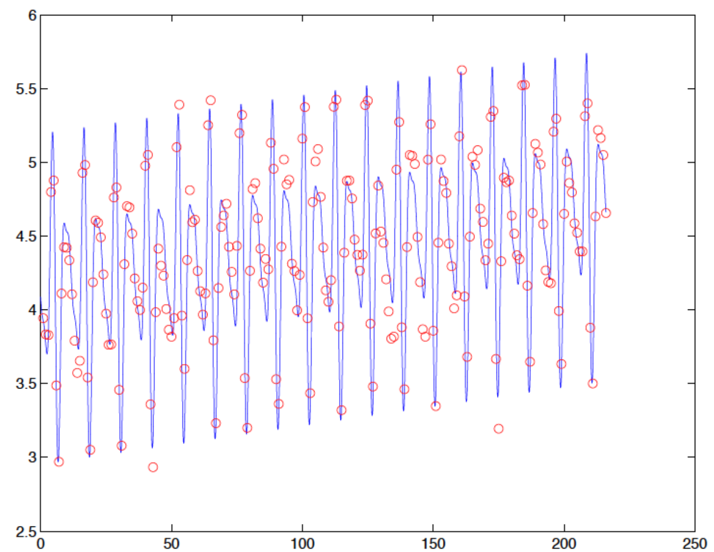


Compact time-series description

- In terms of equations, (roughly) we perform two types of regression, and obtain residuals

$$\boxed{X(t) = \text{polynomial}(t) + \sum_i \left(\beta_i \cos(\lambda_i t) + \gamma_i \sin(\lambda_i t)\right) + R_t}$$

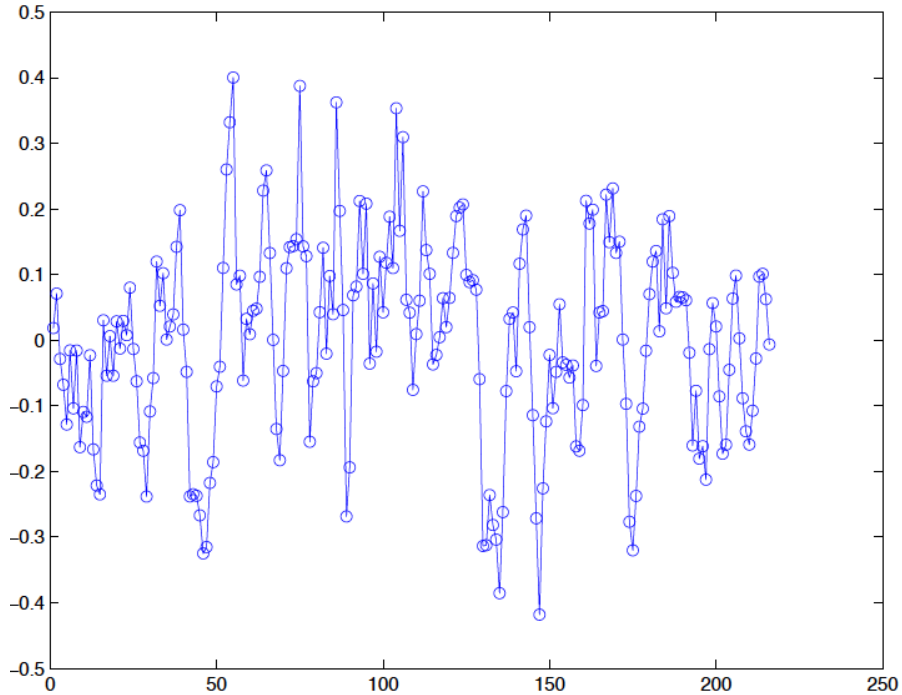
- Second regression
$X(t) - (\alpha_0 + \alpha_i t) = \sum_i \left(\beta_i \cos(\lambda_i t) + \gamma_i \sin(\lambda_i t)\right) + E(t)$:



Compact time-series description

- Left with the residuals:

$$R(t) = X(t) - (\alpha_0 + \alpha_i t) - \sum_i \left(\beta_i \cos(\lambda_i t) + \gamma_i \sin(\lambda_i t)\right)$$



- **Not done yet!** Study the residuals. E.g., can we forecast them?

Compact time-series description

- In Python things have been made easy for us.

```
def _decom(dataset):
    decomposition = statsmodels.tsa.seasonal.seasonal_decompose(dataset)
    plt.plot(decomposition.trend, label='Trend', color = 'b')
    plt.plot(decomposition.seasonal,label='Seasonality', color = 'b')
    plt.plot(decomposition.resid, label='Residuals', color = 'b')
```

Today's outline

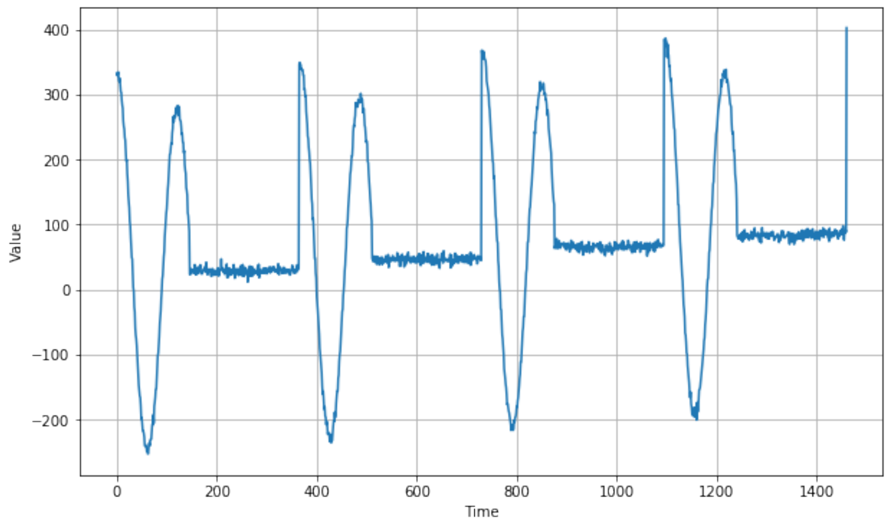
1. Part I: Fundamentals
   - Stochastic processes, strict vs weak stationarity, correlograms, partial correlograms, periodograms, autoregressive models (AR), moving average models (MA), transformations, compact time-series description.

2. Part II: Forecasting methods
   - Evaluation protocols, basic metrics, null models, spectral methods, ARMA, ARIMA, SARIMA, VAR, similarity search, deep learning (DNNs, RNNs)

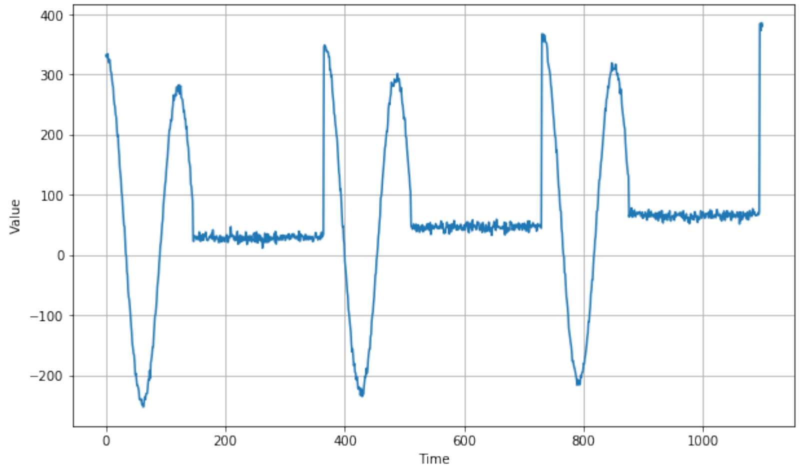
Train and test – Respect order!


```
split_time = 1000
x_train = series[:split_time]
x_test = series[split_time:]
```

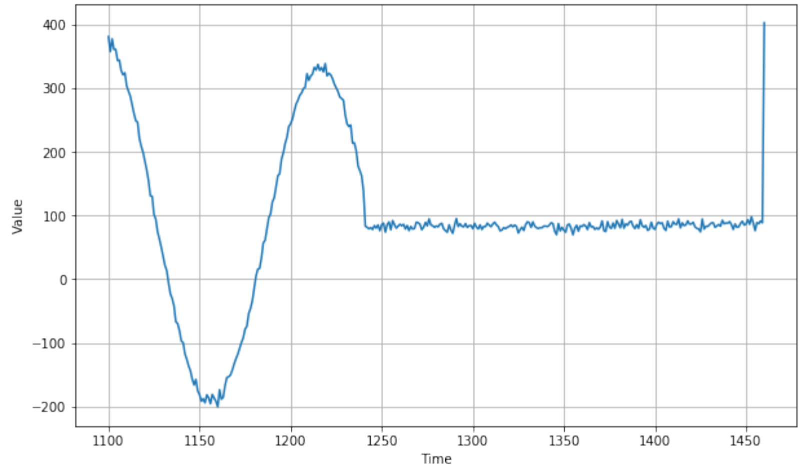
Train and test – Respect order!



Dataset



Training set



Test set

Evaluation protocols

There exist **two main evaluation protocols** for a forecasting method.

1. k-fold cross validation

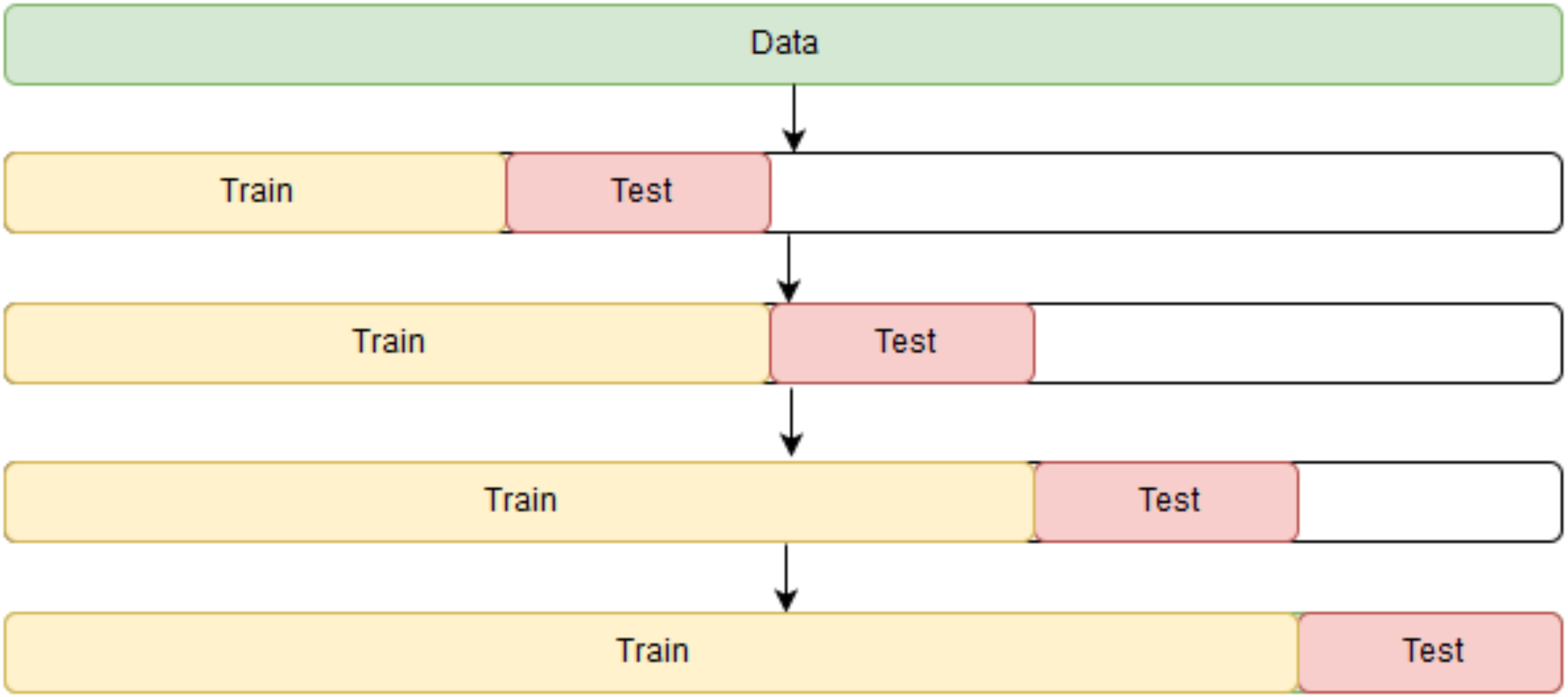
   Remarks

   - Cannot just randomly partition the time-series into folds as in other ML tasks (respect the order!)

2. Walk forward evaluation protocol

- Both protocols can be easily explained visually.

k-fold cross validation



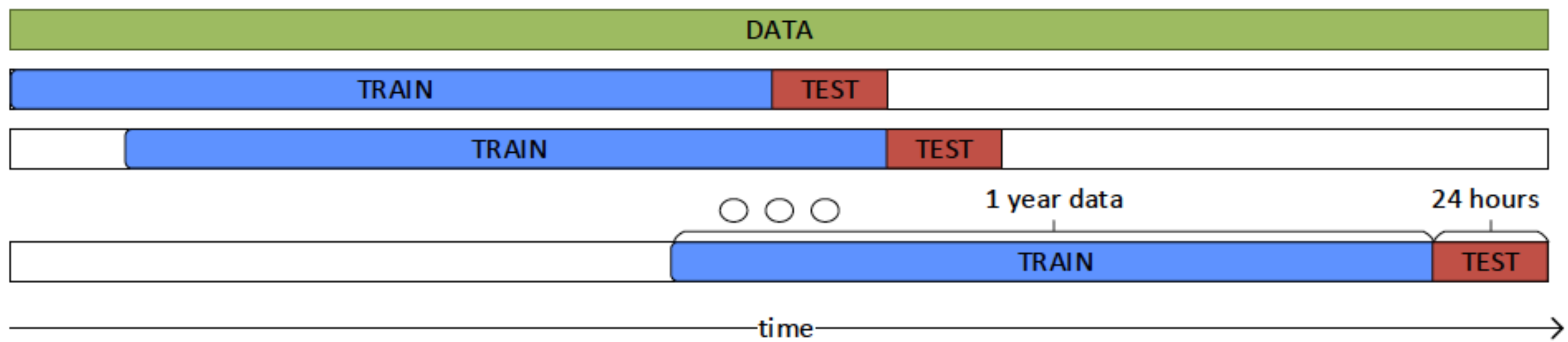
- Test set size remains same, but training size increases.

```
from sklearn.model_selection import
    TimeSeriesSplit
```

k-fold cross validation – sklearn

```
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4],
    [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4, 5, 6])
tscv = TimeSeriesSplit(max_train_size=None,
    n_splits=5)
for train_index, test_index in tscv.split(X):
...     print("TRAIN:", train_index, "TEST:",
    test_index)
...     X_train, X_test = X[train_index], X[
    test_index]
...     y_train, y_test = y[train_index], y[
    test_index]
>> TRAIN: [0] TEST: [1]
>>  TRAIN: [0] TEST: [1]
>> TRAIN: [0 1] TEST: [2]
>> TRAIN: [0 1 2] TEST: [3]
>>  TRAIN: [0 1 2 3] TEST: [4]
>>  TRAIN: [0 1 2 3 4] TEST: [5]
```

Walk-forward evaluation protocol



- Sliding window of same size.
- Straight-forward implementation.
- In reality, when training is expensive we may sample as many as possible such windows, with a bias towards more recent windows.

Basic metrics – g groundtruth, f forecast

```python
import numpy as np
errors = forecast - truth
```

- Mean squared error (mse):

$$mse(f,g) = \frac{100\%}{n}\sum_{t=1}^{n}(f_t - g_t)^2.$$

```python
mse     = np.square(errors).mean()

```

- Root mean squared error (rmse):

$$rmse(f,g) = \sqrt{mse(f,g)}.$$

```python
  rmse    = np.sqrt(mse)

```

Basic metrics

- Mean absolute error (mae):

$$mae(f,g) = \frac{100\%}{n} \sum_{t=1}^{n} |(f_t - g_t)|.$$

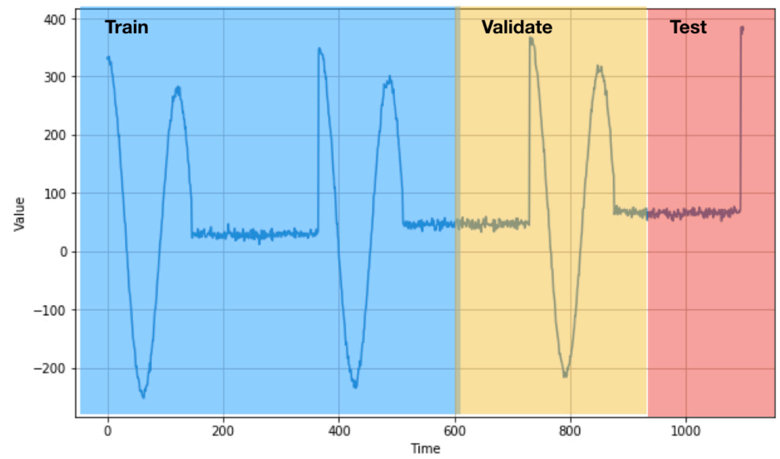
```
mae       =   np.abs(errors).mean()

```

- Mean absolute percentage error (mape):

$$mape(f,g) = \frac{100\%}{n} \sum_{t=1}^{n} |\frac{f_t - g_t}{g_t}|.$$

```
 mape   = np.abs(errors/x_test).mean()

```

Train, validate and test



- Alternatively, we can split the data into
  1. Training set
  2. Validation set in order to evaluate hyperparameters, useful for minimizing overfitting
  3. Test set

```
x_train = series[:split1]
x_validate= series[split1:split2]
x_test = series[split2:]
```

Naive forecasting

- Suppose we wish to predict $x(t+1)$ given past values (and possibly other time-series).
- **Naive forecasting:**

$$\boxed{x(t+1) \leftarrow x(t).}$$

- Example: Suppose we wish to predict Torino's temperature next hour.

  Prediction: The temperature in the next hour is going to be the same as now.

- Competitive baseline!

More null models

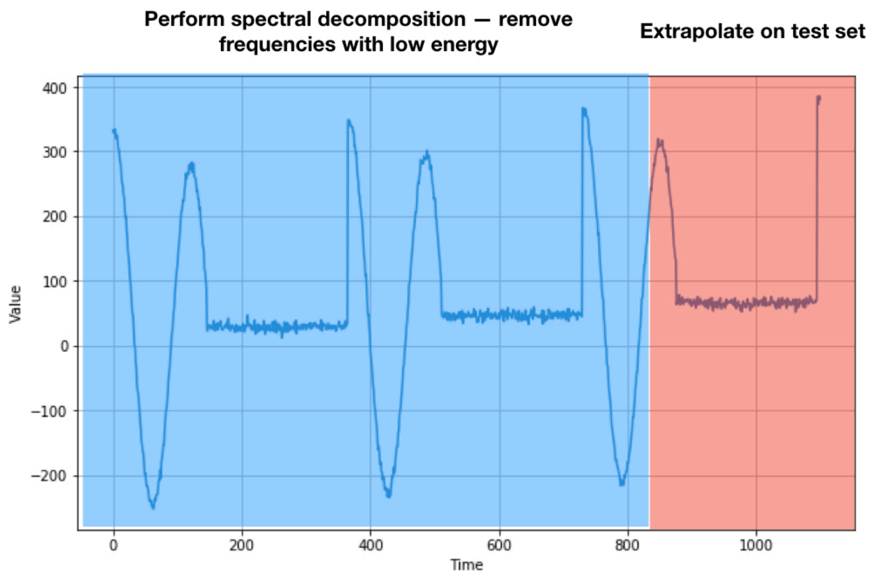
- Constant values, e.g., global average or a frequently occurring value in the dataset.
- Average/median of past most recent w values where w can be small or spanning the whole dataset

$$\boxed{x(t+1) \leftarrow \frac{\sum_{j=1}^{w} x(t-j)}{w}.}$$

- Remark: $w = 1$, i.e., naive forecasting, tends to be the most competitive baseline.
- Exponential smoothing (more importance to most recent measurements)
- Regression and extrapolation
- ...

Spectral forecasting

1. We pretend the training dataset is periodic.
2. We perform a spectral decomposition of the signal in sinusoidal functions.
3. We retain the frequencies that hold most but **not all** of the energy
4. **Rule of thumb:** keeping 80% of the energy is a good choice, but more values should be tried out.
5. Extrapolate the periodic signal to the test set.



Spectral forecasting – Training

```
def spectral_prediction(timeseries, n_predict, perc = 0.8):
    n = timeseries.size
    x_freqdom = fft(timeseries)
    f = np.fft.fftfreq(n)
    indexes = list(range(n))
    harm_squares = np.square(np.abs(x_freqdom))
    sum_squares = harm_squares.sum()
    sortd = np.argsort(-harm_squares)
    cum_sum = 0
    i = 0
    n_harm = 0  # number of harmonics in model, keep perc% of energy
    while cum_sum < sum_squares*perc:
        cum_sum += harm_squares[sortd[i]]
        i+=1
        n_harm+=1
```

Spectral forecasting – Training

```
    my_signal = np.zeros(t.size)
    for i in indexes:
        # This is the amplitude of the extrapolated frequency
        ampli = np.absolute(x_freqdom[i]) / n
        # This is the phase of the extrapolated frequency
        phase = np.angle(x_freqdom[i])
        # we add this component to the extrapolated signal
        my_signal += ampli * np.cos(2 * np.pi * f[i] * t + phase)

return my_signal
```

ARMA

- An ARMA(p, q) process $\{X_t\}$ is a stationary process that satisfies

$$\boxed{X_t - \phi_1 X_{t-1} - \ldots - \phi_p X_{t-p} = W_t + \theta_1 W_{t-1} + \ldots + W_{t-q},}$$

  where $W_t \sim WN(0, \sigma^2)$.
- **Observations**

  1. $ARMA(p, 0)$ is same as $AR(p)$.
  2. $ARMA(0, q)$ is same as $MA(q)$.

ARMA

- ARMA process are very important because of the following (informally) stated theorem:

Theorem (Informal statement)

For any stationary process $\{Y_t\}$ with autocovariance γ, and any $k > 0$ there is an ARMA process $\{X_t\}$ that fits $\{Y_t\}$ well.

- This is one **big** reason why we try to transform a time-series into a stationary process.
- Many null models are special cases of this model.
- Interpretable.
- For certain types of data it is performing well but for volatile, non-stationary time-series it is not.
- **Always** fundamental, non-trivial baseline.

ARMA - Lowest terms

Consider a process $\{X_t\}$ such that $X_t = W_t$ where $W_t \sim WN(0, \sigma^2)$.

$$X_t - 3X_{t-1} + 4X_{t-2} = W_t - 3W_{t-1} + 4W_{t-2}$$

- This looks like an ARMA(2,2) process! But in reality it is not, since it is not.
- Something called *characteristic polynomials* are not in lowest terms, they share common factors. Actually, here both the LHS and the RHS have the same characteristic polynomial $\phi(B) = 1 - 3B + 4B^2$.
- Lowest terms means that the following fraction cannot be not be simplified further

$$\frac{\phi_{RHS}(B)}{\phi_{LHS}(B)}.$$

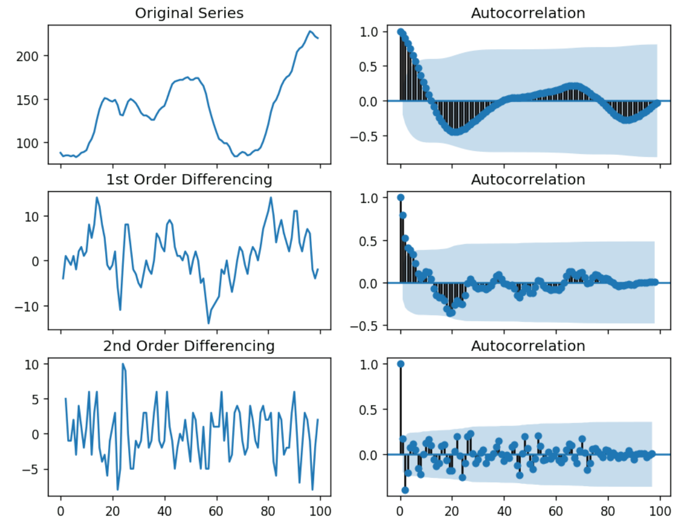
ARIMA

- ARIMA stands for "Auto Regressive Integrated Moving Average"
- In addition to parameters p, q we have another parameter d
- d is the number of differencing required to make the time series stationary

  E.g., when $d = 0$ we have a standard ARMA model.

  Remark: differencing does not always succeed, and overdifferencing may produce "bad" time-series.

ARIMA



- ARIMA extends to SARIMA by including a seasonal component.
- This involves also p, d, q parameters but also the seasonality parameter s.

```
model=sm.tsa.statespace.SARIMAX( data['PSV -
    Italian gas\n[   /MWh]'], order=(1,1,1),

    seasonal_order=(1,1,1,1),
    enforce_invertibility=False).fit(disp=-1)
fitted1 = model.fittedvalues
```

SARIMA - Parameter search

- ACF, PACF plots but also exhaustive search

```
def parameter_search():
    p = range(1, 11), q = range(1, 11)
    d=range(1,3)
    Ps = [1,2,3], D= range(1,3), Qs = [1,2,3]
    s = 7
    parameters = product(p,d,q, Ps, D, Qs)
    parameters_list = list(parameters)
    results = []
    best_aic = float("inf")
    for param in tqdm(parameters_list):
            model=SARIMAX(data, order=(param[0],
     param[1], param[2]), seasonal_order=(param
    [3], param[4], param[5], s)).fit(disp=-1)
        aic = model.aic
        if aic < best_aic:
            print('Best model so far :', param)
```

SARIMA(X) – Summary

- Summary
    1. Very important family of forecasting methods
    2. Well developed packages in Python and R
    3. Lots of theory behind them
    4. Under stationarity, they come with lots of nice properties (e.g., confidence intervals)
    5. However stationarity is not always there...
    6. Therefore, it is always a baseline.
    7. SARIMAX also supports exogenous variables, hence the X.

VAR

- For multivariate series, these concepts have natural generalizations.
- For example, the next equation defines a vector autoregressive process of order 1.

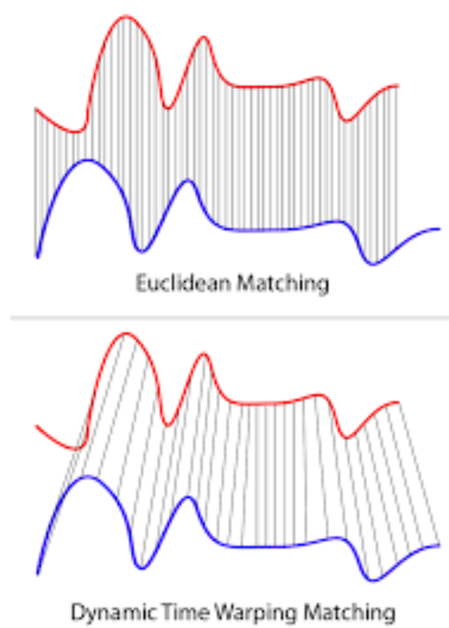
$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \cdot \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \tag{1}$$

- Python *statsmodels* package has VAR ready for us.

```
from statsmodels.tsa.vector_ar.var_model import
    VAR
model = VAR(endog=train)
myfit = model.fit()
prediction = myfit.forecast(y=data[-1,:],steps
    =1)
```

Similarity search

- When are two time series $x(t), y(t), t = 1, \ldots, n$ similar?
  How do we quantify their similarity?
- There exist **two** major families of distances:
  1. Euclidean and ℓ_p norms
     - Euclidean distance $\sum_{t=1}^{n}(x(t) - y(t))^2$ (ℓ_2 distance)
     - Manhattan distance $\sum_{t=1}^{n} |x(t) - y(t)|$ (ℓ_1 distance)
     - Dot product $\langle x, y \rangle$
     - ...
  2. Time warping (DTW) and variations



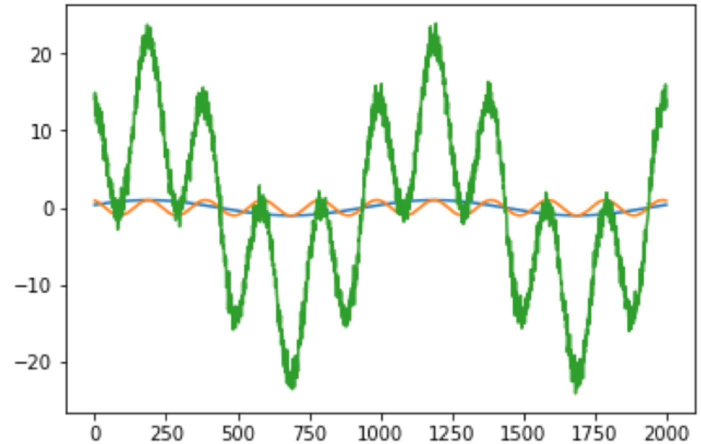
Similarity search

- In Python, again things are pretty straight-forward.

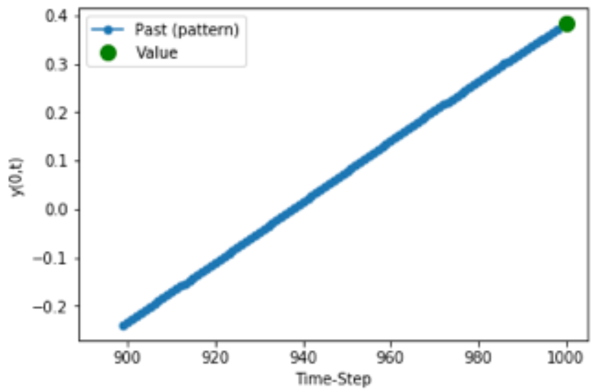
```python
def euclid(points):
    return np.linalg.norm(points[0]-points[1])

def l1_distance(points):
    return np.linalg.norm(points[0]-points[1],
   ord=1)

from dtw import dtw
d, cost_matrix, acc_cost_matrix, path = dtw(x, y
    , dist=l1_distance)
```

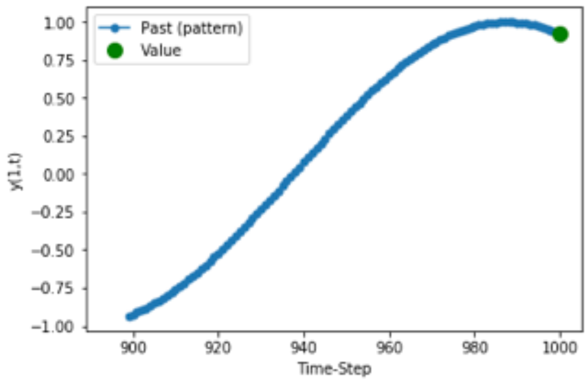
Similarity search - Toy dataset

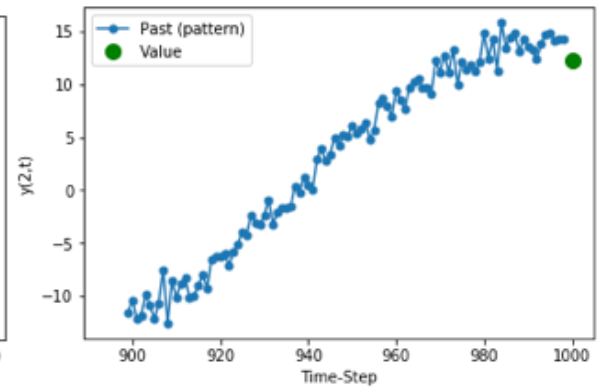
```
def demo1():
    x = np.linspace(-np.pi, np.pi, 2001)
    s1= np.sin(2*x+np.pi/8)
    s2= np.cos(10*x+np.pi/8)
    target = 12*s1+10*s2+np.random.normal(loc
=0.0,scale=1,size=len(x))
    dataset = pd.DataFrame({'f1': s1, 'f2': s2,
'target':target})
    df = dataset
    data = df.values.T
    return data
```



Similarity search - Windowing around timestamp



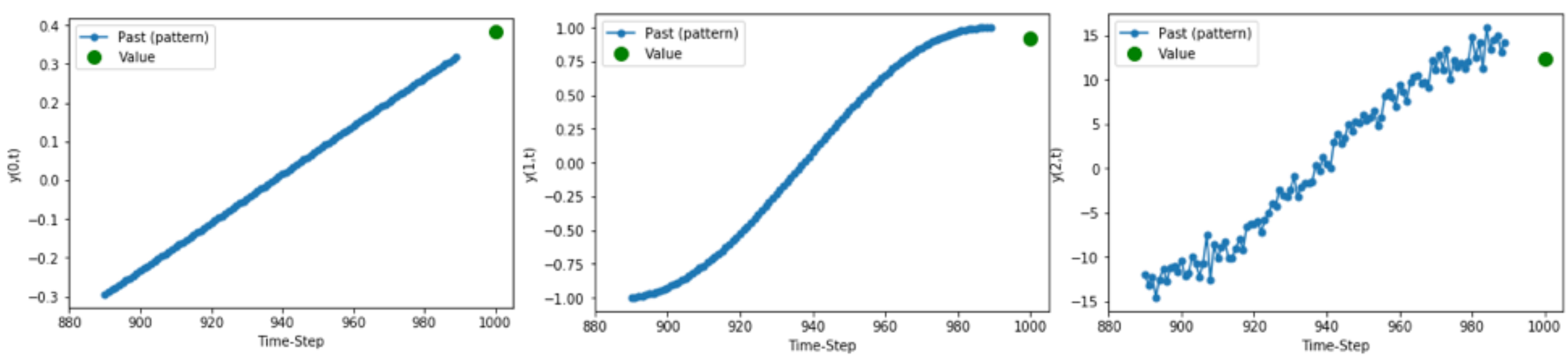




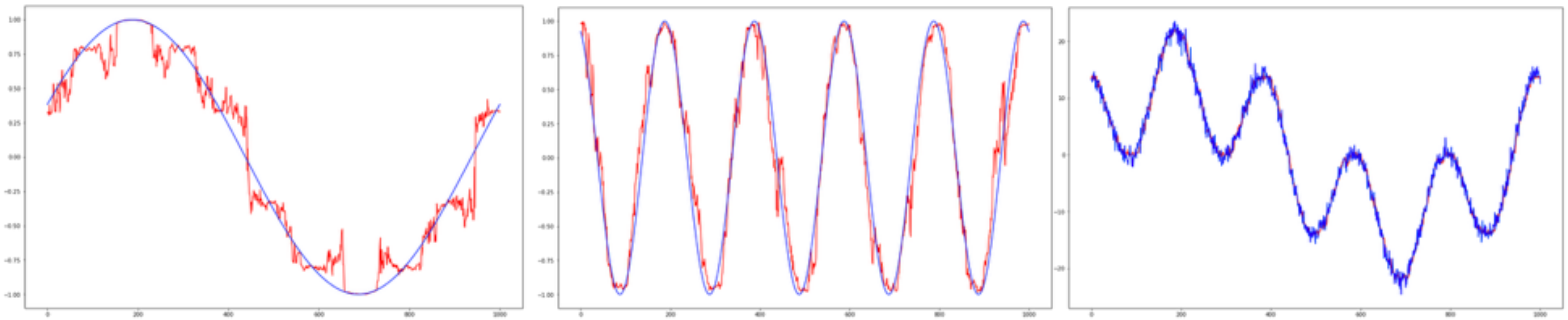
- We simultaneously predict all time series, not just the target time series
- We look into the w previous values and create a vector with $dims \times w$ coordinates.
- We search for the $k \geq 1$ most similar such vector in the training dataset (k small).
- We output the mean/median of the top-k nearest neighbors.

Similarity search

- Sometimes, we don't have the immediate w past values available.
- Then, we just use the w most recent past values that we have available for each time-series.



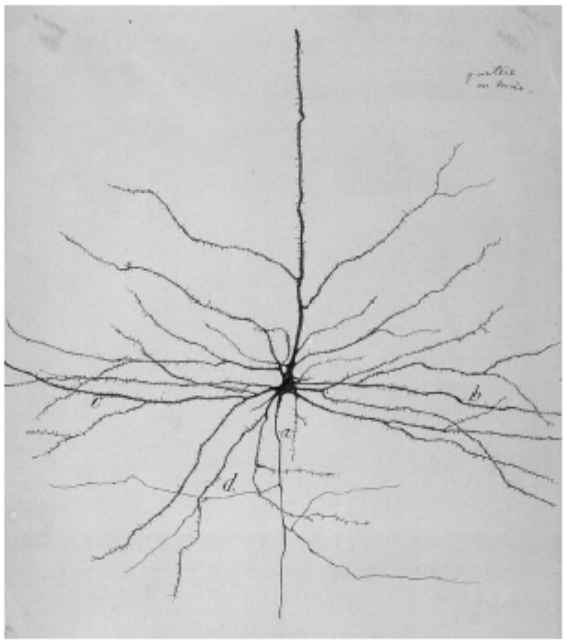
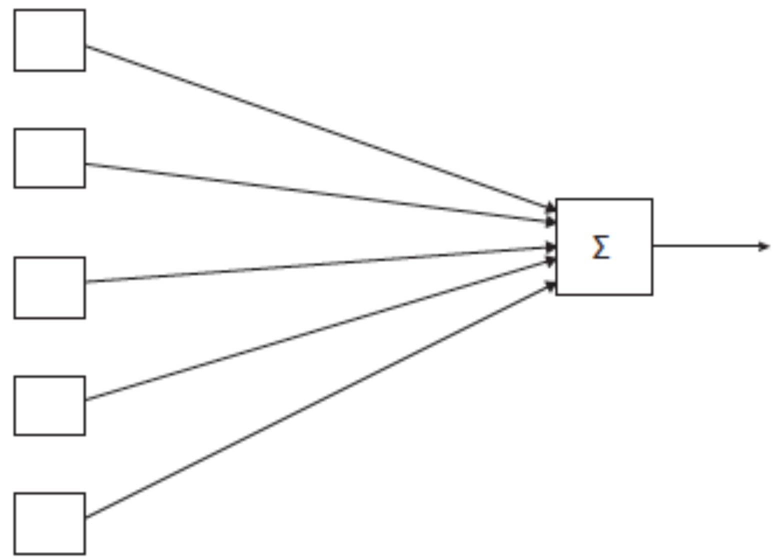
Similarity search – Prediction results



- In this toy example the prediction results are good overall.
- Similarity search is a useful framework that is able to either provide decent predictions or show why other methods fail to product good outputs

  i.e., lots of variance among similar windowed patterns.
- Finding the "right" notion of distance is an important component, as well as the right window length.
- For high-dimensional search, LSH can be used.

Neurons and perceptrons

A human brain neuron and a perceptron

Tensorflow – Open source for deep learning

Tensorflow

- Tensorflow is an end-to-end open source machine learning platform...
- but is also a symbolic math library that can be used even to run a favorite optimization algorithm on your problem.

```python
import tensorflow as tf
x = tf.constant("Hello world")
sess = tf.Session()
print(sess.run(x)
```

Deep learning

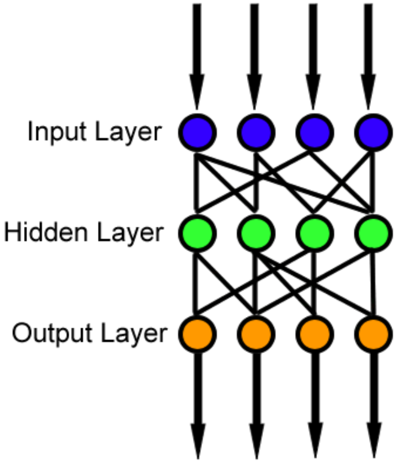
- We have a bunch of inputs associated with an output.
- Can we see the right output for a new input, i.e., an input we have not seen?
- **Deep learning** is a leading approach to data science, that uses multiple layers to automatically extract high-quality, higher-level features from the raw inputs.
  - deep neural networks, recurrent neural networks, convolutional neural networks etc.
- Why does it work? Increasing amount of research to understand why does it work, even when neurons use simple activation functions (piecewise flat.)

Deep feedforward neural networks (DNNs)

Deep feedforward networks feedforward , also often called neural networks, or multilayer perceptrons (MLPs), are the quintessential deep learning models.

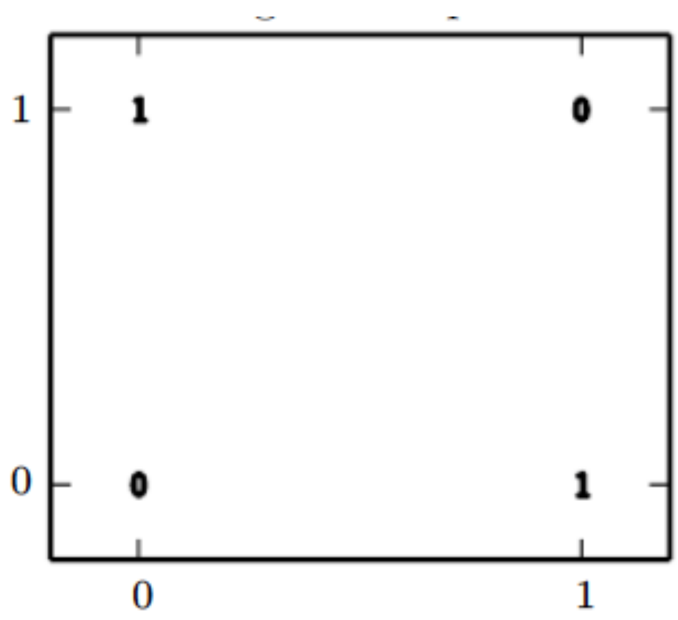
- The goal of a feedforward network is to approximate some function f^* by composing layers, e.g.,

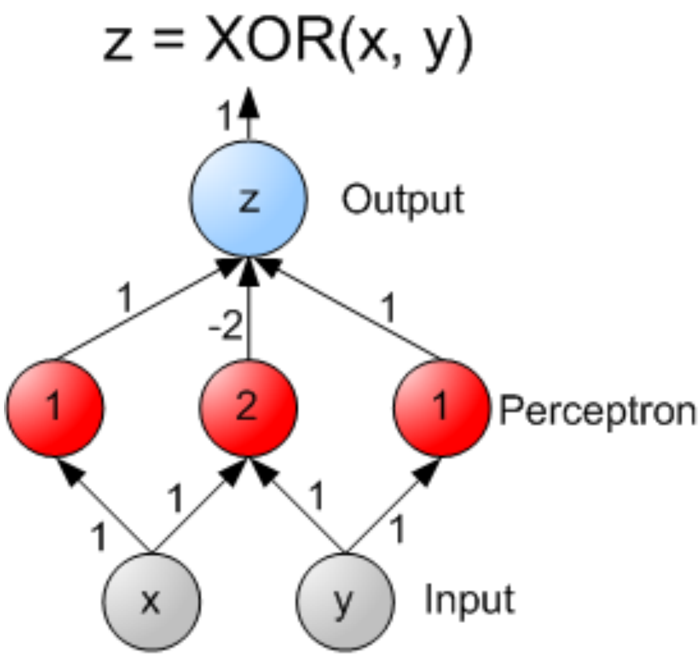
$$\tilde{f}(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))).$$



Deep feedforward neural networks (DNNs) – Example

- Classic example: DNNs can learn the XOR function.





DNNs for time-series – Step 1

1. Prepare time series data for DNN by creating a windowed dataset:
   - For each timestamp, let $x(t)$ be the value/label.
   - The previous w values could be seen as the input features.

- Lots of space for feature engineering (Fourier coeffs, wavelet coeffs, min/max/median etc.)

```
dataset = tf.data.Dataset.range(20)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1:]))
for x,y in dataset:
  print(x.numpy(), y.numpy())
```

DNNs for time-series – Step 1

```
[0 1 2 3] [4]
[1 2 3 4] [5]
[2 3 4 5] [6]
[3 4 5 6] [7]
[4 5 6 7] [8]
[5 6 7 8] [9]
[6 7 8 9] [10]
[ 7  8  9 10] [11]
[ 8  9 10 11] [12]
[ 9 10 11 12] [13]
[10 11 12 13] [14]
[11 12 13 14] [15]
[12 13 14 15] [16]
[13 14 15 16] [17]
[14 15 16 17] [18]
[15 16 17 18] [19]
```

DNNs for time-series – Step 1b (batching)

```
dataset = dataset.shuffle()
dataset = dataset.batch(2).prefetch(1)
x =  [[5 6 7 8] [1 2 3 4]]
y =  [[9] [5]]
x =  [[ 9 10 11 12] [12 13 14 15]]
y =  [[13] [16]]
x =  [[11 12 13 14] [13 14 15 16]]
y =  [[15] [17]]
x =  [[10 11 12 13] [ 4  5  6  7]]
y =  [[14]  [ 8]]
x =  [[ 7  8  9 10]  [14 15 16 17]]
y =  [[11] [18]]
x =  [[3 4 5 6] [0 1 2 3]]
y =  [[7] [4]]
x =  [[15 16 17 18] [ 6  7  8  9]]
y =  [[19]  [10]]
x =  [[ 8  9 10 11]  [ 2  3  4  5]]
y =  [[12] [ 6]]
```

DNNs for time-series – Step 2

2. Split the data into training and test sets.

```
time = np.arange(3 * 365, dtype="float32")
split_time = 2*365+31+28+30
time_train = time[:split_time]
x_train = series[:split_time]
time_test = time[split_time:]
x_test = series[split_time:]
```

DNNs for time-series – Steps 3, 4

3. Setup a DNN architecture (here single layer)
4. Train it

4b Inspect the layer weights

```
dataset = windowed_dataset(x_train, window_size,
    batch_size)
layer = tf.keras.layers.Dense(1, input_shape=[
    window_size])
model = tf.keras.models.Sequential([layer])
model.compile(loss="mse", optimizer=tf.keras.
    optimizers.SGD(lr=1e-6, momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
print("Layer weights {}".format(layer.
    get_weights()))
```

DNNs for time-series – Steps 3, 4

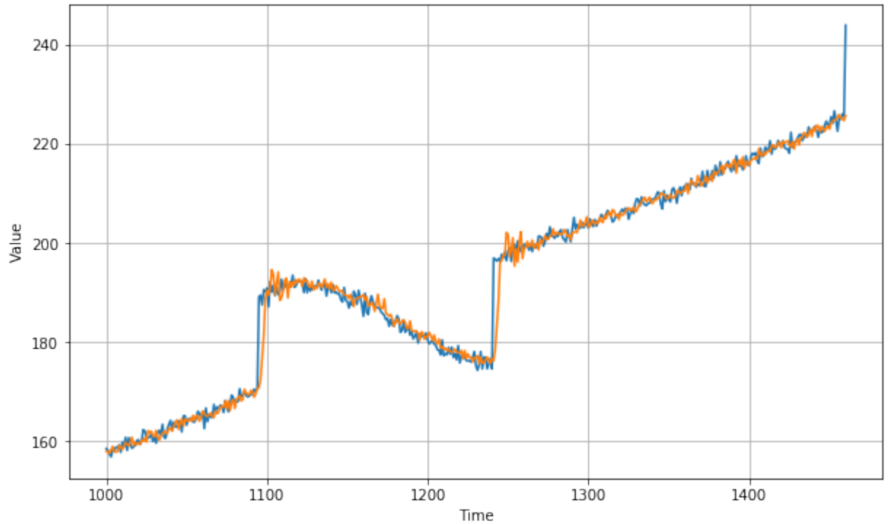
```
Layer weights [array([[-0.03136637],
    [-0.04519343],
        [ 0.05885418],        [ 0.05017925],
        [-0.01355846],       [-0.07844295],
        [ 0.04997651],       [ 0.03857101],
        [-0.01163514],       [ 0.02050608],
        [-0.01307715],         [-0.05444159],
        [ 0.01337368],         [ 0.05073489],
        [ 0.01348611],         [-0.02583448],
        [ 0.08094374],         [ 0.23290157],
        [ 0.20300445],          [ 0.46333405]],
    dtype=float32), array([0.01811779], dtype=
    float32)]

```

- These weights are what we need to forecast a value given a new input of 20 values.

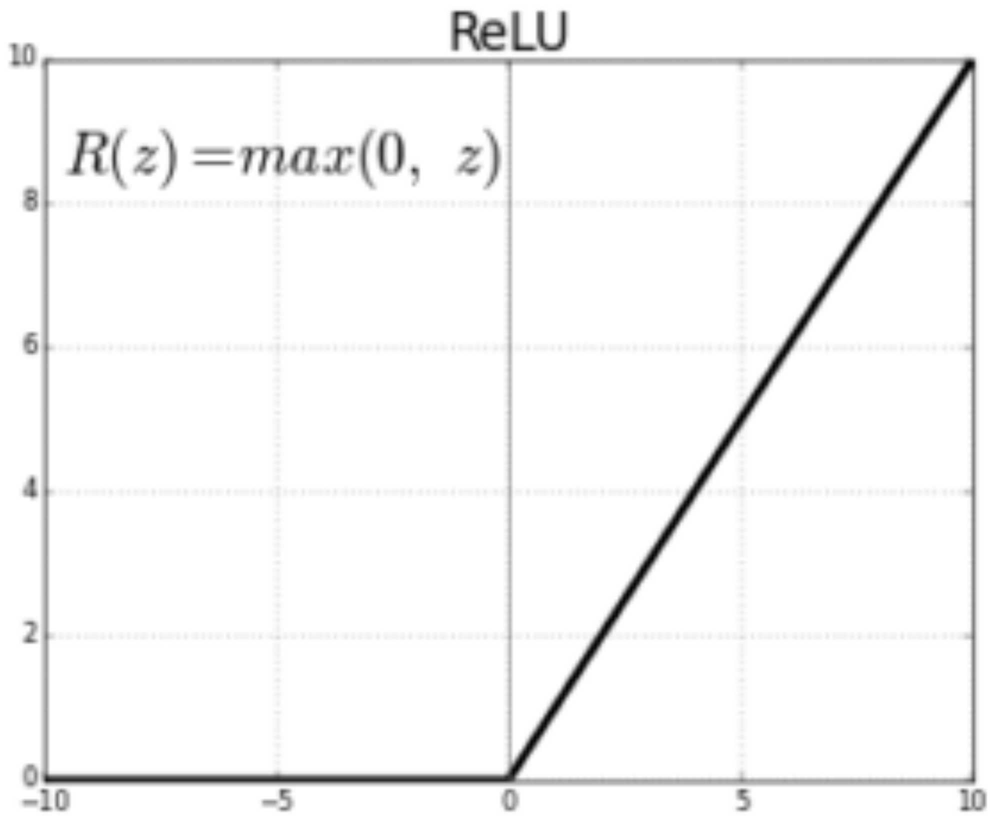
DNNs for time-series – Step 5

5. Forecast

```
forecast = []
for time in range(len(series) - window_size):
  forecast.append(model.predict(series[time:time
      + window_size][np.newaxis]))
forecast = forecast[split_time-window_size:]
plt.plot(time_test, forecasts)
plt.plot(time_test, x_test)
tf.keras.metrics.mean_absolute_error(x_valid,
    results).numpy()
```



Quick comment - Activation function ReLU

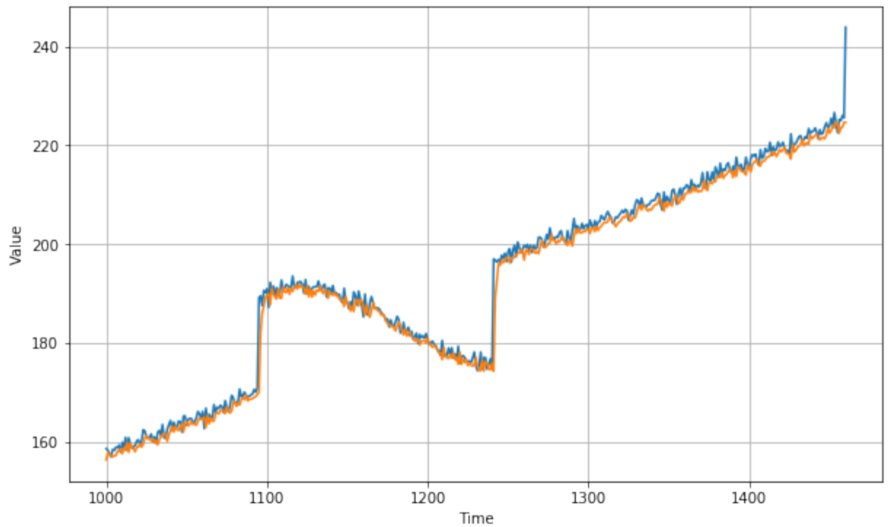


- ReLU is an important activation function, especially because it is simple, yet sparse-friendly.

DNNs for time-series – Adding layers

- Our example uses one layer. We can easily add layers as follows.

```
model = tf.keras.models.Sequential([tf.keras.
    layers.Dense(10, input_shape=[window_size],
    activation="relu"), tf.keras.layers.Dense(10,
     activation="relu"), tf.keras.layers.Dense(1)
     ])

```



DNNs for time-series – Did the extra layers help?

- In addition to the visualization, Keras provides an easy way to compute the various metrics.

```
tf.keras.metrics.mean_absolute_error(x_test,
    results_shallow).numpy()
>> 1.412784
tf.keras.metrics.mean_absolute_error(x_test,
    results_deep).numpy()
>>1.3640015
```

- It is worth outlining that this is not always the case.
- The number of layers is one of the hyperparameters to be optimized.

Remarks on DNNs

- For multivariate time series, we change the input shape from a vector to a matrix (batch size $\times$ number of time-series/dimensions)
- Lots of other parameters need to be fine-tuned, including the weight initializer, use or not of batch normalization, dropout rates, learning rates of SGD etc.
- Feature selection earlier is also useful for DNNs.
- Not suited for sequential data, but for supervised framework.

- Next: Deep RNNs

Last important remark about DNNs

- The approach we described for analyzing time-series can be extended in two different directions.

1. Create more features from the window. For instance add dimensions for the maximum value observed.

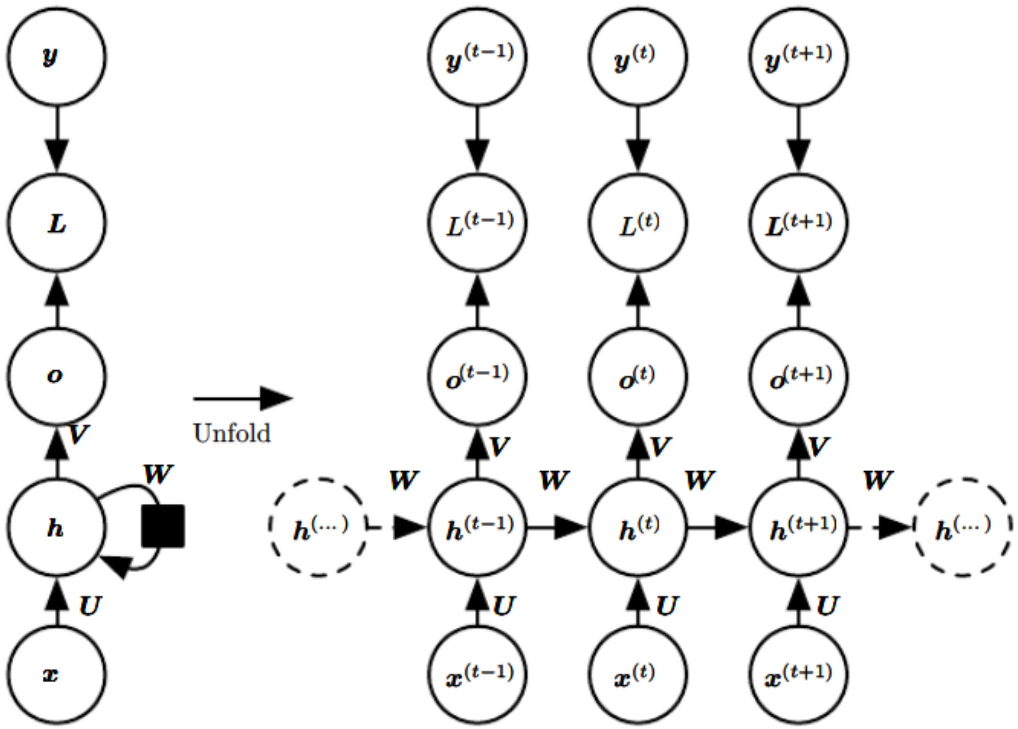
```
from  tsfresh import extract_features
```

   tsfresh is a library that allows for fast extraction of hundreds of features from each window.

2. We can use any other supervised ML method including SVM regressors, regression random forests, Gaussian processes etc.

Deep recursive neural networks (RNNs)

- RNNs are neural networks for **sequential** data.



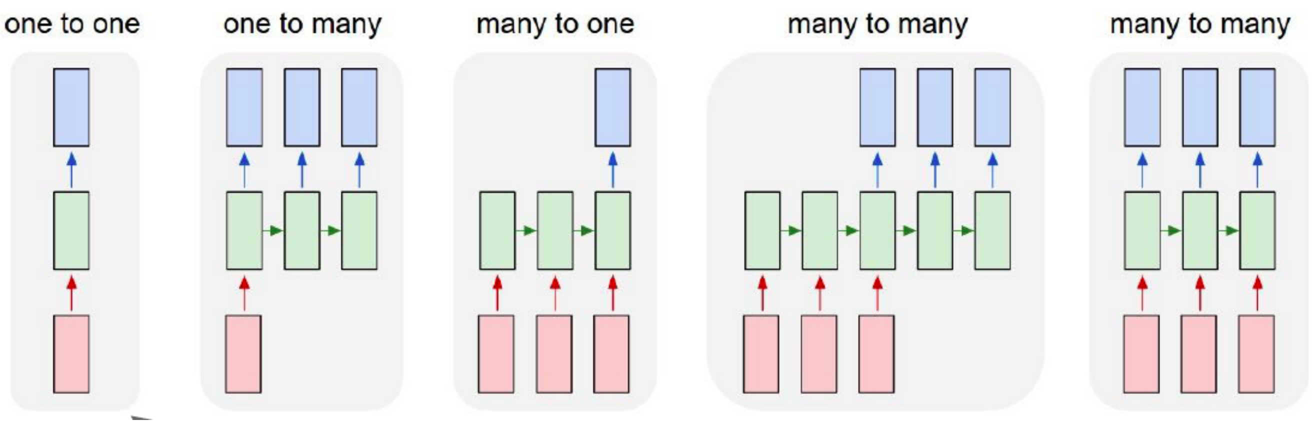
Source: Deep Learning book

$$h_t = f_W(h_{t-1}, x_t).$$

- Same function and same set of parameters W are used at each time step.

Deep recursive neural networks (RNNs)

- They have a distributed hidden state that allows them to store a information about the past.
- RNNs offer a lot of flexibility that makes them valuable in diverse applications



Source: Andrej Karpathy

- An RNN can be thought of as multiple copies of the same network, each passing a message to a successor.

Deep recursive neural networks (RNNs) – TensorFlow

- In constrast to DNNs, the input shape is

$$[batch\ size, \#timesteps, \#dimensions]$$

```
model = tf.keras.models.Sequential([
  tf.keras.layers.SimpleRNN(40, return_sequences
    =True),
  tf.keras.layers.SimpleRNN(40),
  tf.keras.layers.Dense(1)
])

optimizer = tf.keras.optimizers.SGD(lr=5e-5,
    momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
model.fit(dataset,epochs=100)
```

Deep recursive neural networks (RNNs) – Tensorflow

- Prediction is done in the same way, and evaluation too, e.g., using

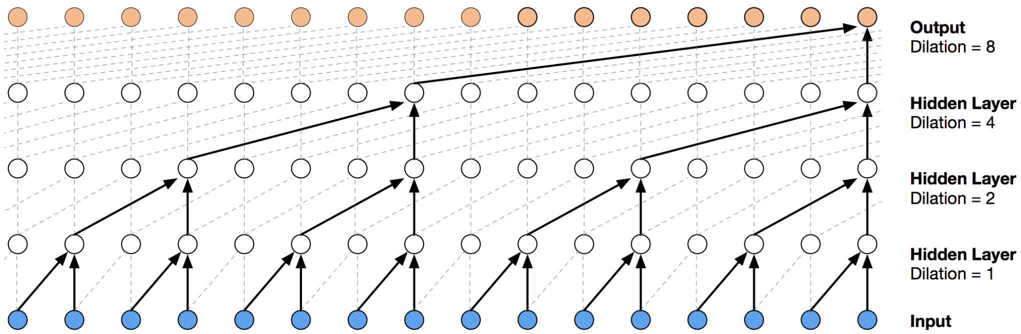
```
tf.keras.metrics.mean_absolute_error(x_test,
    forecast_results).numpy()
```

- The key method is MODEL.PREDICT again:

```
forecast=[]
for time in range(len(series) - window_size):
  forecast.append(model.predict(series[time:time
    + window_size][np.newaxis]))

forecast = forecast[split_time-window_size:]
```

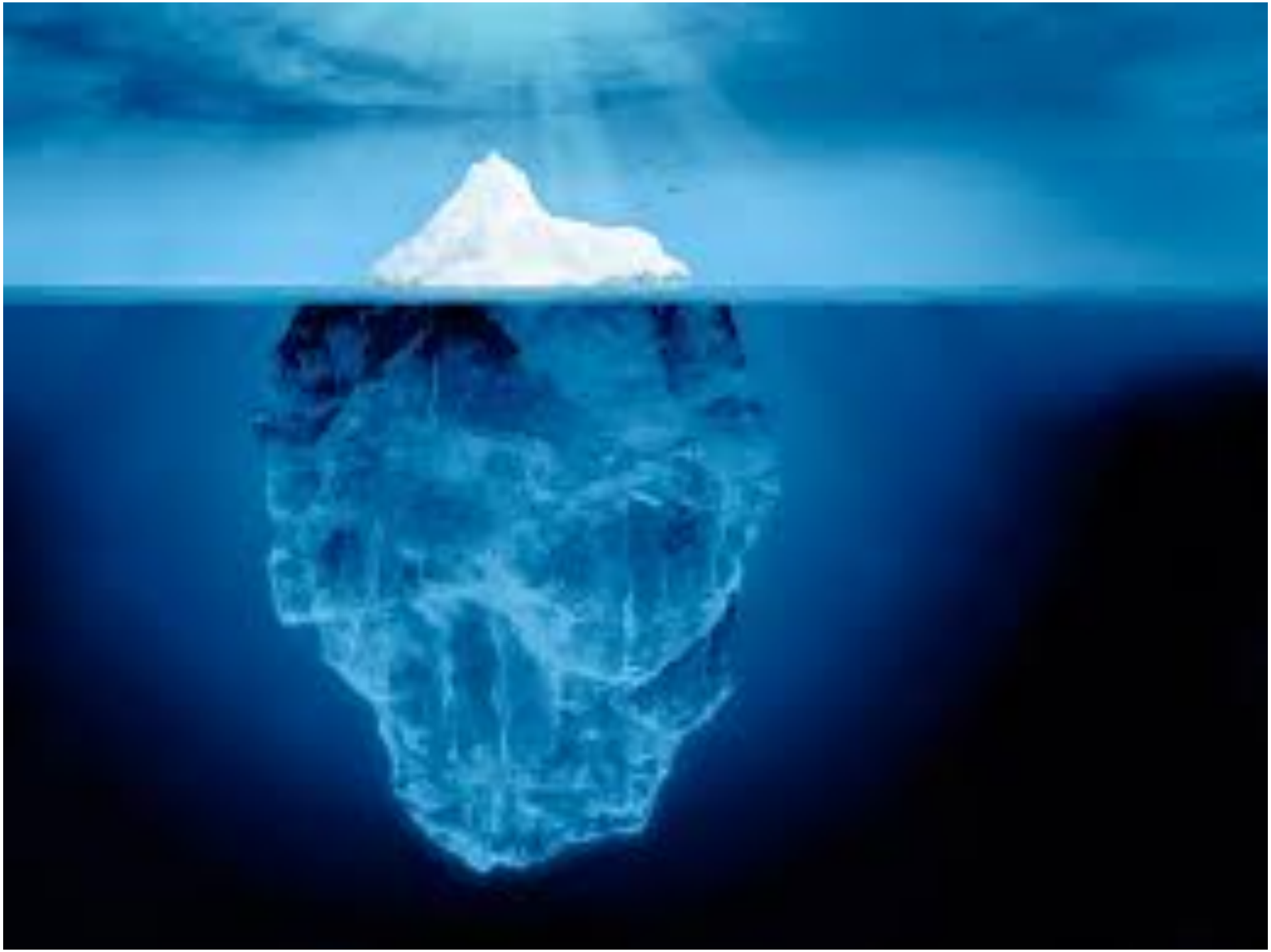
Remark: LSTMs and dilated convolutions

- RNNs are supposed to remember from past, but in practice they forget easily.
- LSTMs are units that are able to capture better long term dependencies.
- State-of-the-art time-series prediction methods combine convolutions, typically used in image processing, with recurrent neural networks.



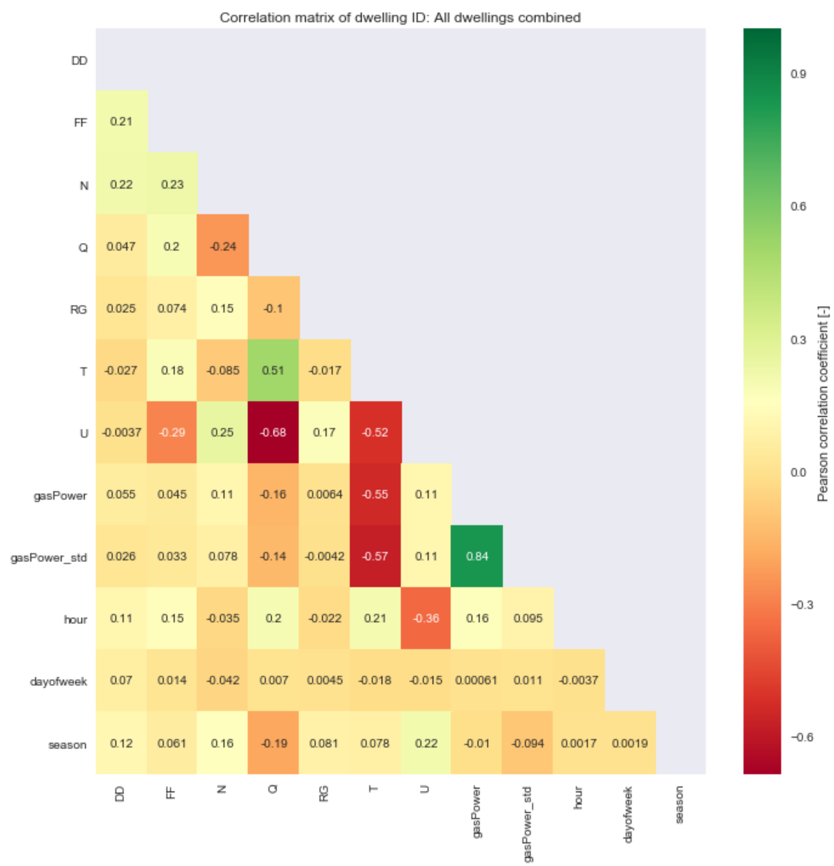
Dilated convolutions

Source: original paper, *Temporal convolutional networks* by Bai et al. Arxiv:1803.01271



Tip of iceberg – Lots more to talk about!

- Today I have scratched the tip of the iceberg.
- E.g., in multivariate analysis how do we select the most relevant time-series to the target series we wish to forecast?



Thank you! Questions?

web page: http://tsourakakis.com
email: ctsourak@bu.edu