Advanced Topics in Signal Processing Project 1

Name: Siatiras Evangelos

AM: EN2190001

The project has been implemented using python 3.6.

In a python environment run the below commands before the execution of the script.

pip install spectrum

pip install scipy

pip install matplotlib

pip install numpy

Exercise_A_1

The power spectrum is defined as the fourrier transform of the autocorrelation function. In the discrete notation the power spectrum equation becomes:

$S_x(\omega) \triangleq \sum_{n=-\infty}^{+\infty} r_{xx}(n)e^{-j\omega n}$, $j = \sqrt{-1}$ and $r_{xx}(n)$ is the autocorrelation function.

Assume

X(n) = $3\sin(0{,}4\,\pi n + \varphi 1)+2\sin(0.5\pi n + \varphi 2) + u(n)$

We define

$x_1(n) = 3\sin(0{,}4\,\pi n + \varphi 1)$

$x_2(n) = 2\sin(0.5\pi n + \varphi 2)$

$x_3(n) = u(n)$

So that X(n) = $x_1(n) + x_2(n) + x_3(n)$

The autocorrelation function: $r_{xx}(n) = r_{x1x1}(n) + r_{x2x2}(n) + r_{x3x3}(n)$

So Let's define $r_{x1x1}, r_{x2x2}, r_{x3x3}$

$r_{x1x1}(n_1,n_2) = E[x_1(n_1)\cdot x_1^*(n_2)] = E[3\ \sin(0{,}4\,\pi n_1 + \varphi 1)\cdot 3\ \sin(0{,}4\,\pi n_2 + \varphi 1)]$

Note: $\sin a\ \cdot\ \sin b = \frac{1}{2}(\cos(a-b) - \cos(a+b)$

$r_{x1x1}(n_1,n_2) = \frac{9}{2}E[\cos 0.4\pi(n_1-n_2) - \cos 0.4\pi(n_1+n_2) + 2\varphi 1]$

Note: The Expectation value is subject to the random variable $\varphi 1$

$r_{x1x1}(n_1,n_2) = \frac{9}{2}\cos 0.4\pi(n_1-n_2) - \frac{9}{2}E[\cos 0.4\pi(n_1+n_2) + 2\varphi 1]$ (1)

Note: In general case

$$E[\cos(k+l)\omega_0 + 2\varphi] \xrightarrow{\varphi\sim\cup(-\pi,\pi)} \int_{-\pi}^{\pi} \cos(k+l)\omega_0 + 2a\, f_\varphi(\alpha)\, d\alpha$$

$$= \frac{1}{2\pi}\int_{-\pi}^{\pi} \cos(k+l)\omega_0 + 2a\, d\alpha = \frac{1}{4\pi}[\sin(k+l)\omega_0 + 2a]\quad \bar{}_{\pi}^{\pi}$$

$$= \frac{1}{4\pi}[\sin(k+l)\omega_0 + 2\pi + \sin(k+l)\omega_0 - 2\pi]$$

$$= \frac{1}{4\pi} 2\cos(k+l)\omega_0 \sin 2\pi = 0$$

So (1) becomes

$r_{x1x1}(n_1, n_2) = \frac{9}{2}\cos 0.4\pi(n_1 - n_2)$ (2)

as $E[\cos 0.4\pi(n_1 + n_2) + 2\varphi 1] = 0$

By following the same rational

$r_{x2x2}(n_1, n_2) = 2\cos 0.5\pi(n_1 - n_2)$ (3)

The autocorrelation of the white noise u(n) equals to:

$r_{x3x3}(n_1, n_2) = r_u(|n_1 - n_2|) = \sigma_u^2 \delta(n_1 - n_2) = \delta(n_1 - n_2)$ (4)

$$r_{xx}(n) \overset{(2),(3),(4)}{\Longleftrightarrow} \frac{9}{2}\cos 0.4\pi n + 2\cos 0.5\pi n + \delta(n)$$

The Fourrier transform of the autocorrelation function equals to the power spectrum is:

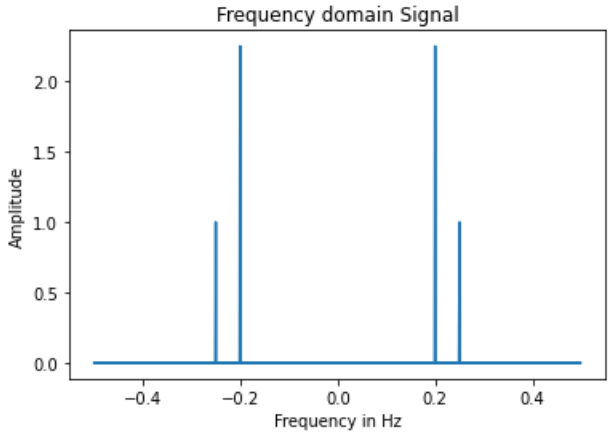
$$S_x(\omega) = \frac{9}{4}\pi[\delta(\omega - 0.4\pi) + \delta(\omega + 0.4\pi)] + \pi\,[\delta(\omega - 0.5\pi) + \delta(\omega + 0.5\pi)] + 1$$

Or equivalently

$$S_x(f) = \frac{9}{4}[\delta(f - 0.2) + \delta(f + 0.2)] + \,[\delta(f - 0.25) + \delta(f + 0.25)] + 1$$

Let's plot the power Spectrum:

```
N = 1000
ni = np.arange(N)

time_data = 4.5*np.cos(0.4*np.pi*ni)+2*np.cos(0.5*np.pi*ni)+signal.unit_impulse(N)

frequency = np.fft.fftfreq(N)
freq_data = np.fft.fft(time_data)
y = 1/N * np.abs (freq_data )


# Plotting the FFT spectrum
plt.plot(frequency, y)
plt.title('Frequency domain Signal')
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude')
plt.show()

```



From the plot we confirm that the autocorrelation function (the input signal in time domain) concentrates its energy around ± 0.2 *and* ± 0.25 *Hz with their related Amplitudes as per the equation* $S_x(f)$.

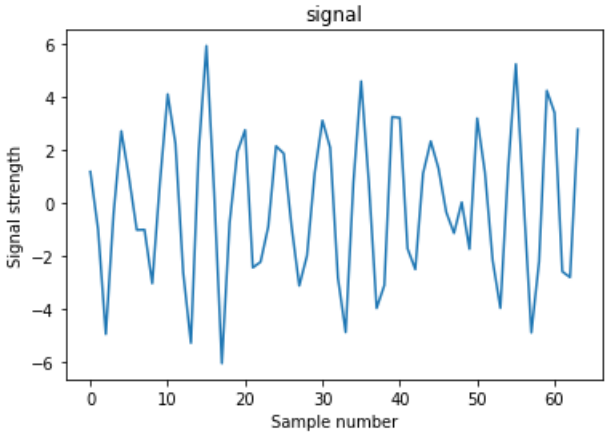
Exercise_A_2

At first we get the 64 required samples.

```
N = 64

ni = np.arange(N)
fi1 =  np.random.uniform(-np.pi,np.pi,1)
fi2 =  np.random.uniform(-np.pi,np.pi,1)

noise=np.random.normal(loc = 0, scale = 1, size = N)

x = 3*np.sin(0.4*np.pi*ni+fi1)+2*np.sin(0.5*np.pi*ni+fi2)
x += noise

plt.plot((x[:64]))
plt.title('signal')
plt.ylabel('Signal strength')
plt.xlabel('Sample number')
```

Note that: fi1,fi2 $\sim \cup (-\pi, \pi)$ and noise $\sim N(\mu, \sigma^2)$ where μ=loc=0 and $\sigma^2 = scale = 1$

So the Resulted Signal is plotted below



The Yule-Walker method, which is also known as the autocorrelation method, determines first the autocorrelation sequence $R(\tau)$ of the input signal. Then, the AR model parameters are optimally computed by solving a set of linear normal equations.

According to the YW methodology we must compute the sample autocorrelation function:

$$r_k = \frac{1}{(n-k)\sigma^2}\sum_{t=1}^{n-k}(X_t - \mu)(X_{t+k} - \mu)$$

```
def autocorr(self, lag=16):
  c = np.correlate(self.X, self.X, 'full
  mid = len(c)//2
  # print (c[mid:mid+lag])
  acov = c[mid:mid+lag]
  acor = acov/acov[0]
  return(acor)
```

The next step is to formulate the YW equations $r = Ra$ based on the sample autocorrelation function r_k.

In details we have formulate the following matrixes

$$\begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_p \end{pmatrix} = \begin{pmatrix} r_0 & r_1 & \cdots & r_{p-1} \\ r_1 & r_0 & \cdots & r_{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ r_{p-1} & r_{p-2} & \cdots & r_0 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix}$$

Note that the vectors R and a have the same numbers of constrains (equations R's rows) as the number of the unknowns (the elements a_j of the unknown vector a).Also R is full-rank and symmetric so that invertibility is guaranteed. Further for lag ≥ 1 $r(-lag) = r(lag)$ so we get:

$$a = R^{-1}r$$

```
def compute_coeffs(self, p=15):
  self.p = p
  #
  ac = self.autocorr(p+1)
  R = linalg.toeplitz(ac[:p])
  r = ac[1:p+1]
  self.a = linalg.inv(R).dot(r)
  return self.a
```

Next, we compute the power spectrum of the AR Model using the coefficients of the AR model calculated above from the following equation:

$$S(f) = \frac{\sigma^2}{\left|1 - \sum_{k=1}^{p} a_k e^{-j2\pi k f}\right|^2}$$

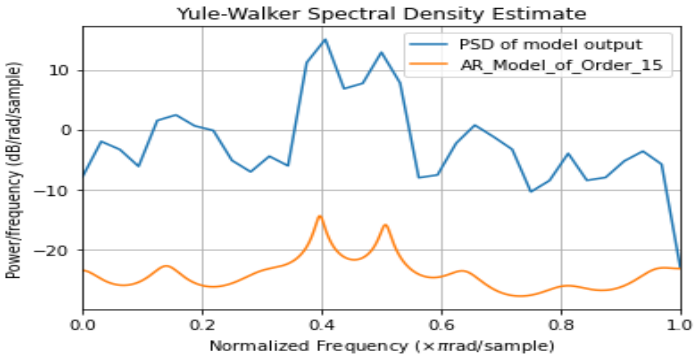
Which is calculated easily by scipy.signal.freqz

```
def power_spectrum(self):
    a = np.concatenate([np.ones(1), -self.a])
    w, h = signal.freqz(1, a)
    h_db = 10*np.log10(2*(np.abs(h)/len(h)))
    plt.plot(w/np.pi, h_db, label='AR_Model_of_Order_%s' % self.p)
    plt.xlabel(r'Normalized Frequency ($\times \pi$rad/sample)')
    plt.ylabel(r'Power/frequency (dB/rad/sample)')
    plt.title(r'Yule-Walker Spectral Density Estimate')
    plt.legend()
```

Our goal is to estimate the parameters of the model that best fits the data of the discrete random process by choosing the right order of the AR Model.

```
p = sp.Periodogram(x, sampling=2)
p()
p.plot(label='PSD of model output')
ar15 = YW(x)
ar15.compute_coeffs(15)
ar15.power_spectrum()
print (ar15.a)
print (sp.aryule(x,15)[0])
```

With an AR model of order 15 we can assume that the model is relatively fit with our data as per the below plot.



Exercise_A_3

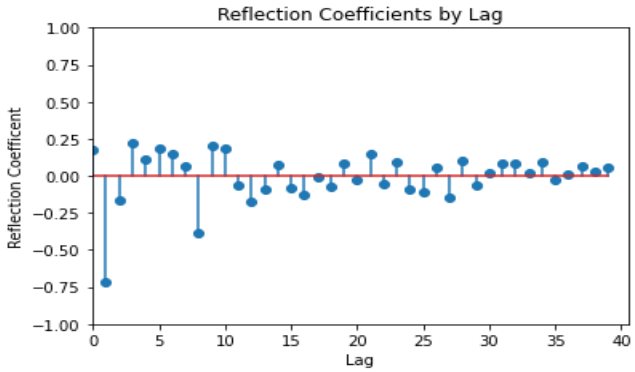
Our goal is to find the optimal order of the AR Model thus finding the optimal AR model coefficients.

At first let's arbitrarily select the order for our AR model to be 40.

```
AR_Model_Order = 40
AR, P, k = aryule(x, AR_Model_Order)
```

Let's use the Yule-Walker method to fit an AR model of order 40 model to our our samples and plot the reflection coefficients, whose negative is the partial autocorrelation sequence.

```
pacf = -k

stem(pacf)
axis([-0.05, AR_Model_Order+0.5,-1,1])
title('Reflection Coefficients by Lag')
xlabel('Lag')
ylabel('Reflection Coefficent')
show()
```



Now based on the number of our samples we define a large-sample 95% confidence intervals.

```
conf = 1.96 /np.sqrt(N)
print ("The Confidence interval for the given number of samples is: ", conf)
for i in range(0,pacf.size):
  if pacf[i] > 0:
    if pacf[i] > conf:
      last_coeff = i
      print ("Reflection Coefficient ",i," is outsize confidence intervals" )
  if pacf[i] < 0:
    if pacf[i] < -conf:
      last_coeff = i
      print ("Reflection Coefficient ",i," is outsize confidence intervals" )
```

We print the occurring lags where the values of the partial autocorrelation sequence are outside the 95% confidence bounds. So for these specific samples we get:

```
                                Lag
 The Confidence interval for the given number of samples is:  0.245
 Reflection Coefficient  1  is outsize confidence intervals
 Reflection Coefficient  8  is outsize confidence intervals
```

This indicating that an AR(40) model significantly overestimates the time dependence in the data.

The optimal AR Model order for the 64 specific samples is the 8^{th} and the AR Coefficients of the Model are :

```
AR_Model_Order = last_coeff
AR, P, k = aryule(x, AR_Model_Order)
print ("the AR model coefficients are: ",AR)
```

```
The AR model coefficients are:  [-0.13489794  0.55820588  0.17430912 -0.28707691 -0.16607093 -0.19989361
 -0.14158002 -0.06754914]
```

The input-output relation of an autoregressive model (AR) process is:

$$y(n) = -\sum_{\kappa=1}^{p} a_\kappa y(n-k) + b_0 x(n) \Leftrightarrow$$

$$y(n) = -a_1 y(n-1) - a_2 y(n-2) - \cdots - a_p y(n-p) + b_0 x(n)$$

The previous input-output relation is for both frequency and time domains.a_k are the values of the AR model parameters and $x(n)$ is a sequence of stationary uncorrelated sequences (white noise) with zero mean and unity variance. Also it is proven that b_0^2 represents the estimated variance $\sigma_x^2 = \sigma_u^2 = 1$ of the white noise input to the AR model. So the above equation becomes :

$$y(n) = -a_1 y(n-1) - a_2 y(n-2) - \cdots - a_p y(n-p) + u(n)$$

The output based on the above calculated coefficients is calculated as per below and is and N=64 samples vector.

```
y = np.zeros(N)
for n in range(0,N):
  for k in range(0,AR_Model_Order):
    if n-k>=0:
      y[n] = - AR[k] * y[n-k]
    y[n] += noise[n]
print (y)
```

The YW set of linear, normal equation for AR Model of order p is as follows:

$$r_x(k) + \sum_{i=1}^{p} a_p(l) r_x(k-l) = \sigma_u^2 b_0^2 \delta(k), k \geq 0$$

With respect to the above assumptions for σ_u^2 *and* b_0^2 the equations for an AR model of order p we have:

$$r_0 = 1$$

$$r_1 = a_1 r_0 + a_2 r_1 + a_3 r_2 + a_4 r_3 + a_5 r_4 + a_6 r_5 + a_7 r_6 + a_8 r_7$$

$$r_2 = a_1 r_1 + a_2 r_0 + a_3 r_1 + a_4 r_2 + a_5 r_3 + a_6 r_4 + a_7 r_5 + a_8 r_6$$

$$r_3 = a_1 r_2 + a_2 r_1 + a_3 r_0 + a_4 r_1 + a_5 r_2 + a_6 r_3 + a_7 r_4 + a_8 r_5$$

$$\vdots$$

$$\vdots$$

$$r_{p-1} = a_1 r_{p-2} + a_2 r_{p-3} + a_3 r_{p-4} + \ldots + a_{p-1} r_0 + a_p r_1$$

$$r_p = a_1 r_{p-1} + a_2 r_{p-2} + a_3 r_{p-3} + \ldots + a_{p-1} r_1 + a_p r_0$$

Similarly and in Matrix form the second set of equations is :

$$\begin{pmatrix} \sigma_u^2 |b_0|^2 = 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} r_0 & r_1 & \cdots & r_{p-1} \\ r_1 & r_0 & \cdots & r_{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ r_{p-1} & r_{p-2} & \cdots & r_0 \end{pmatrix} \begin{pmatrix} 1 \\ a_1 \\ \vdots \\ a_p \end{pmatrix}$$

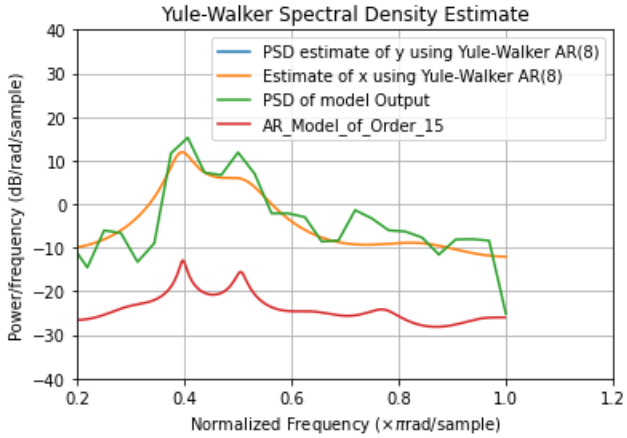
Exercise_A_4

At first with the use of the scipy.signal.lfilter we create an output signal (y) with the MA part of the process equals to 1 as the order of MA process(part) q=0 ,the optimal AR coefficients calculated in 1_3 and the noise samples of 1_2 .

```
from pylab import *
import scipy.signal as sc
from spectrum import *

y = scipy.signal.lfilter([1], list(AR), np.transpose(noise.reshape(-1,1)))

p = pyule(y[0], AR_Model_Order) #Calculation of PSD
p()
p.plot(label='PSD estimate of y using Yule-Walker AR('+str(AR_Model_Order)+')')
############### Model from Question 3 ##################
PSD = arma2psd(AR, NFFT=512)
PSD = PSD[len(PSD):len(PSD)//2:-1]
plot(linspace(0, 1, len(PSD)), 10*log10(abs(PSD)*2./(2.*pi)),
    label='Estimate of x using Yule-Walker AR(8)')
############### Model from Question 1 ##################
p = Periodogram(x,sampling=2)
p()
p.plot(label='PSD of model Output')
############### Model from Question 2 ##################
ar15.power_spectrum()

axis([0.2, 1.2,-40,40])
```

Then given the output signal (y) as input to pyule with the related to the AR coefficients order and we plot the PSD.In the same plot is combined the PSD of the optimal AR model calculated in 1_3 using the aryule as well as the model calculated by following the YW methodology in 1_2 and at last the PSD of our original process.

```
(0.2, 1.2, -40.0, 40.0)
```



At first the green line shows the behavior with the given samples of our true model. Then we try to estimate an AR model with the right order able to fit the data or in other words to see some similarity in the frequencies where the energy is concentrated. We see that the red line has it's peak power spectral density around 0.4π and 0.5π same like our true model. Then knowing that the coefficients of the AR model are directly connected with the covariance function of the process we choose the optimal order that the AR Model should have to fit our data. We saw that the optimality is achieved with a lower order of the AR model and with the specific samples an 8^{th} order.By plotting the PSD with the orange line we see a fitting error as well but we can assume that it is perfoming good. The assumption is because of the limitation in the samples. The optimality is showed within a big number of samples and a small fitting error. In the end with the blue line is shown the PSD of the output of the filter which is performing very bad as it is out of the plot as its energy is not concentrated in the above-mentioned frequencies. With the above mentioned rational we should be able to see its plot as the PSD is calculated with the predefined noise samples and predefined AR model coefficients constructed by the same noise samples. From a couple or runs with different number of samples we conclude that the Yule-Walker method performs adequately only for long data records. The inadequate performance in case of short data

records is usually due to the data windowing applied by the Yule-Walker algorithm. Moreover, the Yule-Walker method may introduce a large bias in the AR estimated coefficients since it does not guarantee a stable solution of the model.

Exercise_B

At first in terms of Linear Regression using Least Squares will use an a n^{th} degree polynomial trying to establish a general relationship between the independent variable x and dependent variable y.The goal is to minimize the cost function given by:

$$J(\theta) = \frac{1}{2\mu}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$$

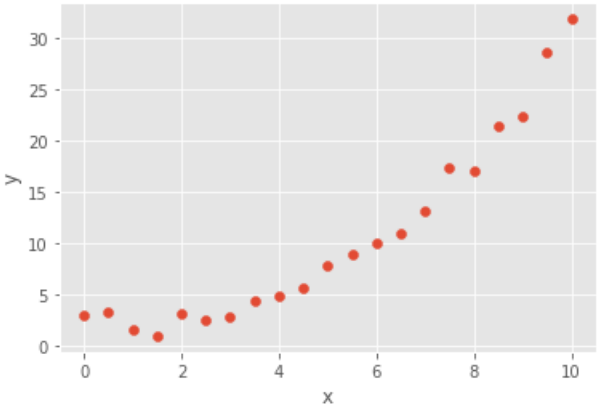
Where the hypothesis in this case is given by the linear model:

$$h_\theta(\chi) = \theta^T x = \theta_0 x + \theta_1 \chi^2 + \theta_2 \chi^3 + \ldots + \theta_n \chi^n$$

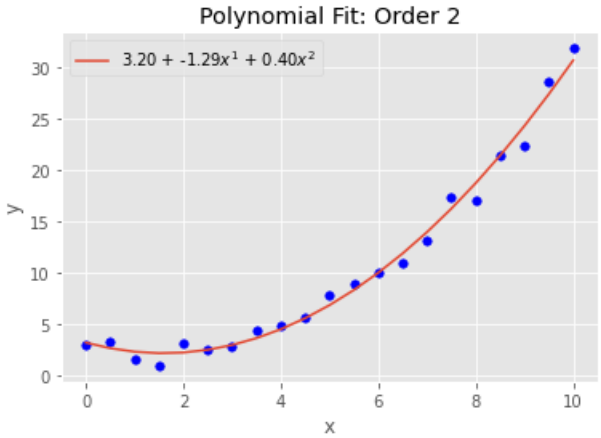
So our goal is to estimate the optimal ϑ in the rational of least squares, estimated by the following normal equation:

$$\theta = (X^T X)^{-1} X^T y$$

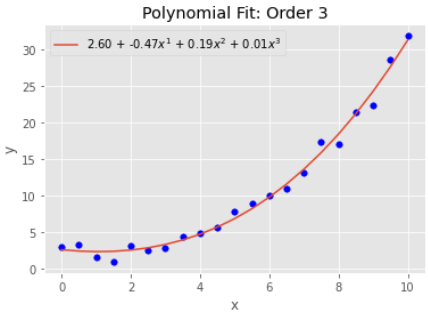
At first let's plot our data

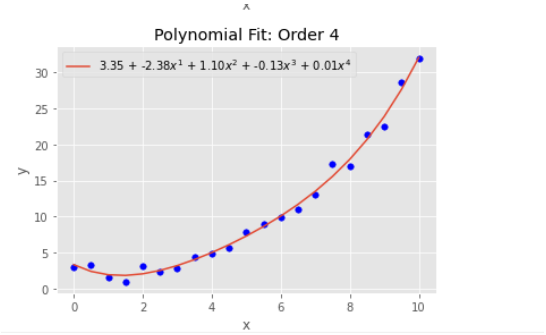


After some experimentations with the polynomial order but also intuitively as well, would say that a polynomial in the 2^{nd} order with the right coefficients would fit the data.



The polynomial in the legend with theta(0) =3.20 ,theta(1) = -1.29 and theta(2) = 0.40 performs good enough.Let's try with a higher order polynomial let's say order 3 and order 4.





Both of the equations described in the legends of the above plots try to fit the data with a better way by minimizing the cost function.This is our goal but when we increase the polynomial order without noticing significant changes in the values of the coefficients the model starts to overfit the data as becoming more an more "sensitive" to small random changes e.g from noise.

Next we will try to estimate the function in a non linear way and specifically exponentially.

We define the following model:

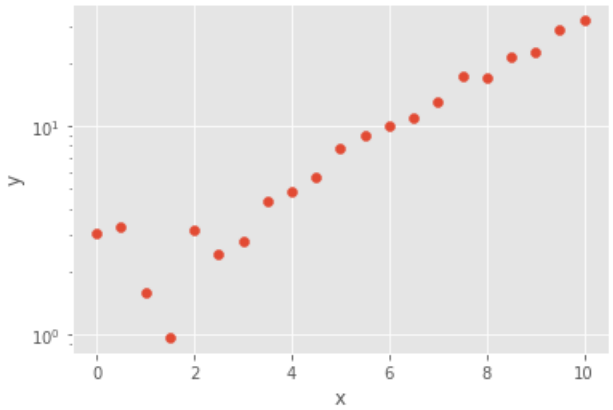
$$y = \theta_0 e^{\theta_1 \chi}$$

Note that if we apply the log to both sides we end up with the following equivalent model:

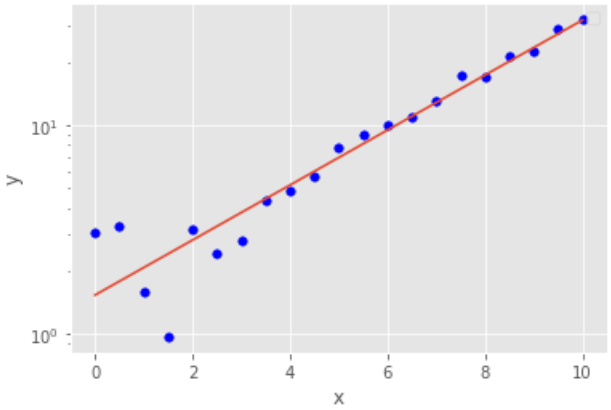
$$\ln(y) = \ln(\theta_0) + \theta_1 \chi$$

So, our goal exactly like above is to estimate the optimal ϑ vector.

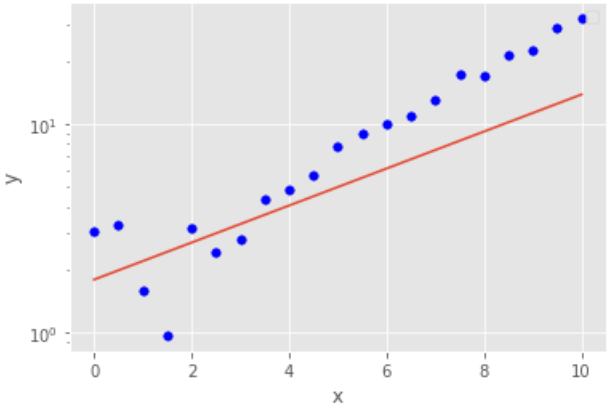
Our data in a logarithmic scale appeared in the following plot.



After fitting our model to the data the ϑ vector is calculated as: [0.42863532 0.30273125] so the resulting equation is and if we plot the function we have in logarithmic scale:



The Resulting function performing pretty good. Also this model has been tested with an increased order of the polynomial inside the exponent introducing a very big fitting error compared to the above result as it is starting to underfit our data.



So in this specific model, the choice of not introducing more coefficients thus more patterns in the model is optimal in the terms of least squares.