Curve_fitting_Python_light

November 5, 2022

1 General information

- Author: Joris Busink, Junior Teacher Physics Education.
- Date: Fri, 4th Nov.
- About: Data-analysis and data-visualization using Python 3.

For questions and suggestions please email: J.Busink@vu.nl. Github

1.1 Structure of the notebook

Example 1, Curve_fit: 1. Load packages. 2. Model and mock data. 3. Plot data. 4. Fit data to model.

Example 2, Lmfit: 5. Example: Lmfit.

2 Example: Analysis of a pendulum (SHO)

2.1 Load packages

I load the following packages: * Numpy, numerical Python. * Matplotlib.pyplot, for creating static, animated, and interactive visualizations in Python. * Pandas. A fast and efficient library to handle DataFrame objects.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

np.set_printoptions(precision=4,threshold =9,suppress=True)  #Compact display.
```

2.2 Pendulum model and mock data

We are interested in x-position of the pendulum as a function of time. For abitrary (starting) angle $\theta(t)$ the system has a non-linear response. However, for small angles we can approximate $\sin(\theta(t)) \approx \theta$ and we have a linear system, thus:

$$x(t) = Ae^{-\gamma t}\sin(\omega t) \tag{1}$$

The harmonic response is given by $\sin(\omega t)$, with $\omega^2 = \frac{g}{l}$. A is the amplitude of the oscillation. g the gravitational acceleration and l the length of the cord. The damping term is modelled by an exponential decay $e^{-\gamma t}$, with γ a damping constant.

To start of, I create a mock-data set using the model of the simple pendulum (equation 1). The x-data is an array (a 'list' that allows nummerical manipulations) that represents the time-domain of the oscillator that ranges from 0 to 10, with $\delta t = 0.2$. The y-data is the x-position of the oscillator, we create this by evaluating the Simple Harmonic Oscillator model function using the following input parameters:

$$x(t) = 2e^{-0.2t}\sin(3t) + 0.2\sqrt{t}\zeta(t). \tag{2}$$

I included a stochastic term, $0.2\sqrt{t}\zeta(t)$, to make our data non-ideal/noisy. $\zeta(t)$ is a random uniform number (float) from -1 to 1. The exact details of the process are not important.

- Line 1: create random seed (to replicate noise).
- Line 3-4: generate t-data and yxdata.
- Line 5-6: noise in t-data and x-data.

[ ]:
```
np.random.seed(2) #initialize random number generator seed.

t = np.arange(0,10.2,0.2) # t-axis data, an array ('list') from [0,0.2...9.8,10],
# this is equivalent to: t = np.array([ 0.,0.2 , 0.4 , ... , 9.8 ,  10.0])
x = 2*np.exp(-0.2*t)*np.sin(3*t) + 0.2*np.sqrt(t)*np.random.uniform(-1,1,len(t)) # x-axis data.
terr = 0.1 # constant error in time.
xerr = 0*t +0.2 # The error in x-position, please note that the shape of xerr must be of the same size as x.
```

2.2.1 Digital data

In case our data is only digital available (for example, as a .csv file), on option is to import the data as a DataFrame (df):

df = pd.read_csv('path\to\data.csv', delimiter='fill_in') Read_csv.

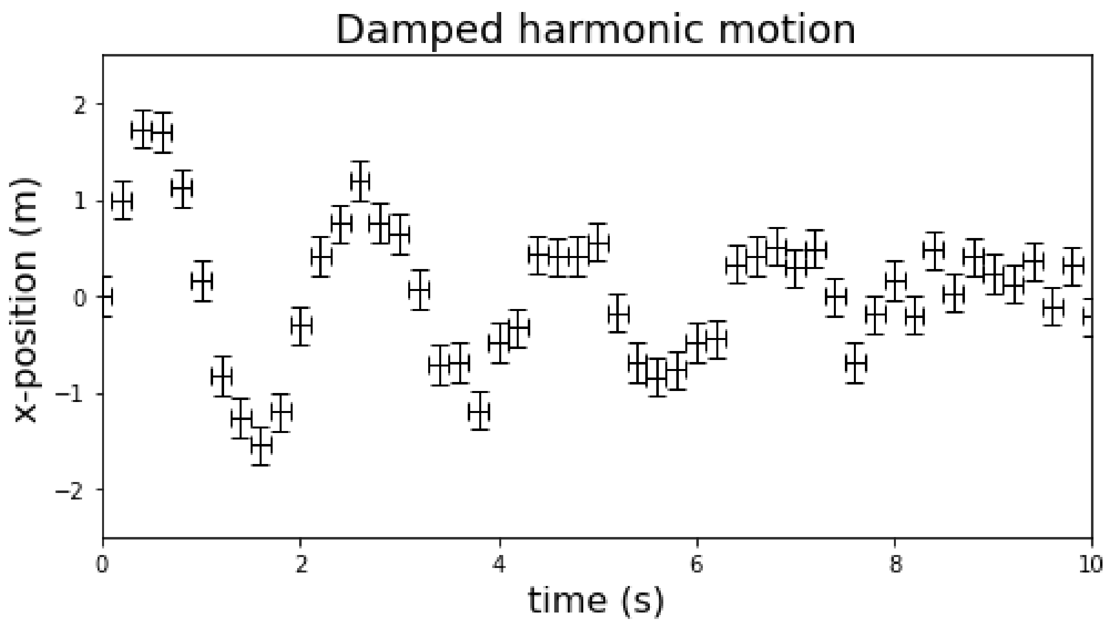
Another options is to use Numpy genfromtxt: data = genfromtxt('my_file.csv', delimiter=',')

2.3 plot the data

In the next lines of code I plot the data. Our data is 2 dimensional (x,y) and contains errors, so we use an errorbar plot.

- Line 1: create 1 figure (1 row and 1 column). Figure documentation
- Line 2: plot data using an errorbarplot. Errorbar documentation
- Line 3-8: optional commands.

```
[ ]: fig, axes = plt.subplots(1,1,figsize=(8,4)) #Creates a single figure with␣
      ↪dimensions (12 by 6 (inch))
     axes.
      ↪errorbar(t,x,xerr=terr,yerr=xerr,fmt='none',ecolor='black',capsize=4,elinewidth=1)
     axes.set_title('Damped harmonic motion', fontsize=18)
     axes.set_xlabel('time (s)',fontsize=16)
     axes.set_ylabel('x-position (m)',fontsize=16)
     axes.set_xlim(np.min(t),np.max(t))
     axes.set_ylim(-2.5,2.5)
     plt.show()
```



2.4 Fit data to model

We want to fit our model, equation 1, to our data, we use the scipy.optimize libary, and in particular we use the Curve_fit function.

- Line 1-2: Here we define our model (sinus_model()) using a Python function: a function starts with def and ends with return. Sinus_model() is our user defined model based on the theory. We vary the time coordinate (the independent variable, t) and the parameters (x_0, A, ω and ϕ) are determined by a fit. Be careful that the independent variable (t) must appear before the parameters, otherwise an error message will appear.
- Line 4: Apply a fit using the curve_fit function. Curve_fit documentation

Curve_fit asks for a minimum of three input arguments: (model, independent variable (time), dependent variable (x-coordinate)), more input arguments are optional. In our case, I provide an initial guess of the optimal parameters (p_0). Other options include: bounds, weights, method, etc. Note that providing a good initial guess of p_0 and apply parameter bounds significantly reduces the complexity of finding the optimal parameters, if possible, provided them! The output of Curve_fit are the optimal parameters popt and the covariance matrix $\mathcal{K}_{p_i p_j}$ pcov, where $\mathcal{K}_{p_i p_j}$ is defined as:

$$\mathcal{K}_{p_i p_j} = \begin{bmatrix} s_{p_a p_a} & s_{p_b p_a} \\ s_{p_a p_b} & s_{p_b p_b} \end{bmatrix} \tag{2}$$

For example, $s_{p_a p_a}$ is the (co)variance of parameter a. To obtain the standard deviation σ_a of parameter a we take the square root of the variance:

$$\sigma_a = \sqrt{s_{p_a p_a}} \tag{3}$$

- Line 6-8: Show the optimal fit values and corresponding standard deviations.

[ ]:
```
from scipy.optimize import curve_fit

def sinus_model(t,A,gamma,omega,phi):
    return A*np.exp(-gamma*t)*np.sin(omega*t+phi)

popt, pcov = curve_fit(sinus_model,t,x,sigma = xerr,p0=[1,0.1,4.12,0.5])

parameter=['Amplitude (m)','Gamma (1/s)', 'Frequency (rad/s)', 'phase (rad)']
for i in range (4):
    print("The optimal value for ", parameter[i], 'is:', "{:.4f}".
 ↪format(popt[i]), '\u00B1', "{:.6f}".format(np.sqrt(pcov[i,i])))
```

```
The optimal value for  Amplitude (m) is: 2.0380 ± 0.144019
The optimal value for  Gamma (1/s) is: 0.2198 ± 0.023873
The optimal value for  Frequency (rad/s) is: 2.9934 ± 0.022534
The optimal value for  phase (rad) is: -0.0094 ± 0.063073
```

- Line 10-15: Make figure, plot the data (black) and the fit (orange).

We plot the fit by making a new dataset, called tfit. We evaluate the the function using the optimal parameters at the values of tfit.

- Line 17-25: Optional commands.
- Line 27-28: Save the figure.

Note that the figure is saved as a .svg extension. A .svg extension stands for Scalable Vector Image, the image is saved as an object (and not as a collection of pixels). One can import .svg picture in e.g. inkscape, powerpoint or adobe photoshop to manipulate these, try it!

[ ]:
```
tfit = np.linspace(0,max(t),1000) #create new data for fit.

#Plot data + fit
fig,axes=plt.subplots(1,1,figsize=(8,4),sharex=True,sharey=True)
axes.plot(tfit,sinus_model(tfit,*popt), color ='darkorange', lw = 4,zorder =1)␣
 ↪#plot fit
```

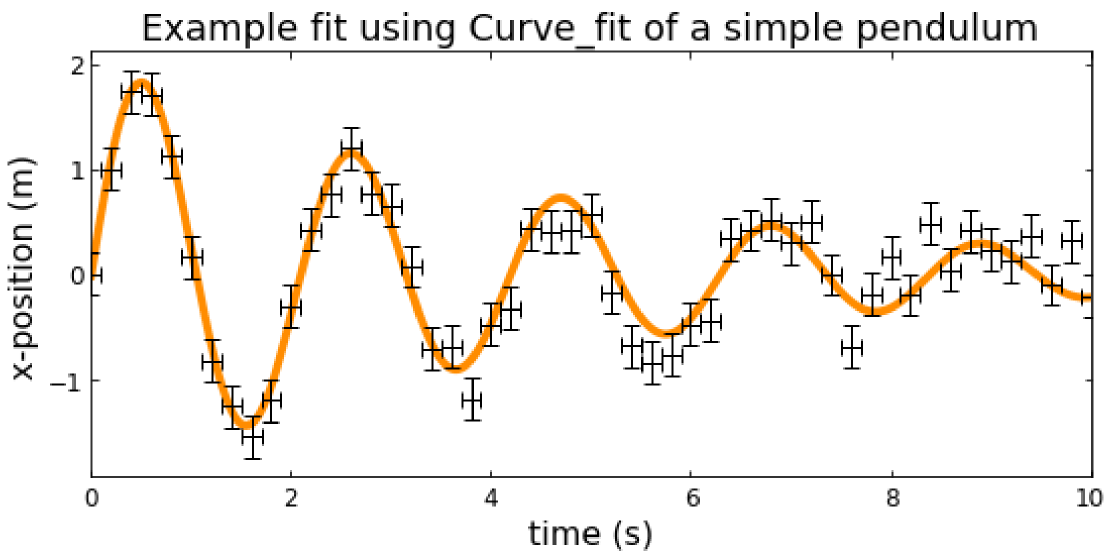
```
axes.
 ↪errorbar(t,x,xerr=terr,yerr=xerr,fmt='none',ecolor='black',capsize=4,elinewidth=1)

#plot axis labels and limits
axes.set_title('Example fit using Curve_fit of a simple pendulum',fontsize=18)
axes.set_xlabel('time (s)',fontsize=16)
axes.set_ylabel('x-position (m)',fontsize=16)
axes.set_xlim(0,10)

#Some plot settings
axes.tick_params(direction="in",labelsize=12,bottom = True,top = True,left=␣
 ↪True,right=True) #inward direction of tick-lines
plt.tight_layout() #creates optimal padding levels for figure (especially␣
 ↪usefull for side-by-side figures)

# location='Path' #Path to your prefered location, e.g. User/Downloads/
# plt.savefig(location+'simple_pendulum.svg') #Extension can be .png/.jpg/.etc␣
 ↪or .svg/.pdf (Vector Image)

plt.show()
```



3 Example: fit data using lmfit

We can also use a package called 'lmfit' to fit our data to a model. On the canvas page there is an extensive document that guides you through the process of fitting using lmfit, for questions please visit the canvas page.

In the code below you find a minimum working example using lmfit:

- Line 4: define model using λ function.
- Line 7: Initialize model parameters system parameters (p_0).
- Line 10: Provide bound to e.g. amplitude parameter.
- Line 13: Apply fit.

[ ]:
```python
from lmfit import models

#Define model:
model = models.Model(lambda t, Amplitude, Gamma, Omega,Phi:
    Amplitude * np.exp(-Gamma * t) * np.sin(Omega*t+Phi))

#Initialize parameters (p0)
params = model.make_params(Amplitude=2, Gamma=0.2, Omega=3,Phi=2)

#Provide bounds
params['Amplitude'].min = 0

#Fit data using model.fit
fit = model.fit(x, t=t, params=params,weights = 1/xerr)
```

We can show the fit using f.best_fit and show the fit statistics using fit.fit_report(). The fit_report() provides the optimal parameters, the error on each parameter and the $\tilde{\chi}^2$.

[ ]:
```python
fig,axes=plt.subplots(1,1,figsize=(8,4),sharex=True,sharey=True)
axes.plot(t, fit.best_fit, color='darkorange',lw=4,zorder =1)#plot fit
axes.
 ↪errorbar(t,x,xerr=terr,yerr=xerr,fmt='none',ecolor='black',capsize=4,elinewidth=1)

#plot axis labels and limits
axes.set_title('Example fit using Lmfit',fontsize=18)
axes.set_xlabel('Time (s)',fontsize=16)
axes.set_ylabel('x-position (m)',fontsize=16)
axes.set_xlim(0,10)
# plt.show()
plt.close()
#print fit report
# print(fit.fit_report())
```

4 Your Experiment starts here

- Author: Student name.
- Date: Date.
- About: Your experiment.
- TA:

Please ask your TA if you need to hand in your code at the end of the practical course.

[ ]: