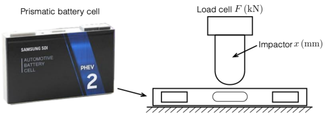
OPT python exercise 1

In this exercise, we want to use linear and polynomial regression and compare them to the built-in curve fitting tool from scipy.

The underlying problem we are looking at is linked to mechanical testing of prismatic battery cells (see Figure below). Several experiments were carried out where an hemispherical impactor was used to deform prismatic cells. From these experiments, both the displacement of the impactor (referred to as *intrusion*) as well as the force were registered.



In this exercise we want to use optimization to determine:

1. the initial stiffness of the prismatic battery cell which can be estimated when the intrusion (i.e. displacement) is lower than 1.0 mm
2. a mathematical model representing the relationship between force and intrusion for later use in finite element analysis for example.

The initial stiffness can be obtained by a linear regression of the data for small intrusion levels as well as curve fitting a linear model defined as:

$$y = ax + b$$

The force-intrusion response of the battery cell can be represented (as far as the data suggest) by a second order polynomial given by:

$$y = a_1 + a_2 x + a_3 x^2$$

The parameters of this second order polynomial can be obtained by polynomial regression or by curve fitting as well.

To obtain what we are looking for, you need to:

1. import the data from the experiments using `numpy loadtxt`, this cell is already coded.
2. extract all intrusions x and forces y from the data to determine the mathematical model of the battery cells, these vectors will be referred to as *large intrusion data*. This cell is already coded.
3. create smaller arrays for the initial stiffness computations where $x \leq 1.0mm$
4. visualize the data in a subplot where the small intrusion and large intrusion data are plotted independently. You should use a `scatter` plot here.
5. define a function for the linear model where the independent variable x is given as input as well as its parameters a and b
6. define a function for the second order polynomial model where the independent variable x is given as input as well as its parameters a_1, a_2 and a_3
7. determine the parameters of the linear model using the `curve_fit` module of `scipy` using the small data set (question 3)
8. determine the parameters of the second order polynomial model using the `curve_fit` module of `scipy` using the large data set (question 2)
9. plot the fitted models using a subplot (similar to question 4)
10. compute and print the goodness of the fit R^2 for the different models
11. plot the second order polynomial model up to a very large intrusion level (i.e. 26.5 mm)

Bonus questions:

1. Determine the parameters for the linear model using the formula presented in the lectures. This model is used for the small intrusion levels.
2. Determine the parameters for the second order polynomial model using the formula presented in the lectures. This model is used for the large intrusion levels.
3. print out the parameters of the different models obtained with the analytical and scipy module.
4. compute and print the goodness of the fit R^2 for the different models

First we import our mandatory modules.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve
from scipy.optimize import curve_fit
```

Question 1

we use the numpy loadtxt function to import the text file which contains our experimental data. The columns of this text file are separated with commas meaning that we need to specify it through the keyword `delimiter`. This file contains the name of the variables as the first line. To avoid problem when reading the file, we need to specify that the first line should not be read. To this end we use the `skiprows` keyword.

In [ ]:
```
data = np.loadtxt('data_OPT_1.csv',delimiter=',',skiprows=1)
```

Question 2

we extract the intrusion x and force from the data array using slicing. To avoid confusion with arrays later in the notebook we will name them _large

In [ ]:
```
x_large = data[:,0]
y_large = data[:,1]
```

Question 3

from the entire dataset we extract the data related to small intrusions via array slicing.

In [ ]:
```
x_small = x_large[np.where(x_large<=1.0)]
y_small = y_large[np.where(x_large<=1.0)]
```

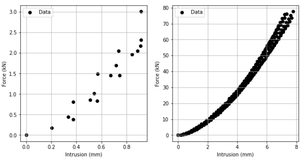
Question 4

We then visualize the datasets, both large and small intrusions using subplots

In [ ]:
```
fig,axs = plt.subplots(1,2,figsize=(10,5))
axs[0].scatter(x_small,y_small,label='Data',c='k')
axs[0].set_xlabel('Intrusion (mm)')
axs[0].set_ylabel('Force (kN)')
axs[0].grid()
axs[0].legend()

axs[1].scatter(x_large,y_large,label='Data',c='k')
axs[1].set_xlabel('Intrusion (mm)')
axs[1].set_ylabel('Force (kN)')
axs[1].grid()
axs[1].legend()
```

Out [ ]:
```
<matplotlib.legend.Legend at 0x13037bca0>
```



Question 5 & 6

We then define the linear and polynomial models using dedicated functions. The functions take as input the intrusion x and the corresponding parameters. It is important here to follow the requirements from the scipy module curve_fit where the first variable must be the independent variable followed by the parameters of the function.

In [ ]:
```
def model_1(x,a,b):
    return a*x+b

def model_2(x,a1,a2,a3):
    return a1+a2*x+a3*x**2.0
```

Question 7 & 8

Based on our datasets and the functions defined earlier, we can find our parameters using curve_fit.

In [ ]:
```
# Fit parameters using scipy module curve_fit
popt_1, pcov_1 = curve_fit(model_1, x_small, y_small)
# Fit parameters using scipy module curve_fit
popt_2, pcov_2 = curve_fit(model_2, x_large, y_large)
```

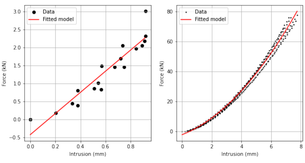
Question 9

We can then plot the fitted model as well as the experimental data using a subplot again.

In [ ]:
```
fig,axs = plt.subplots(1,2,figsize=(10,5))
axs[0].scatter(x_small,y_small,label='Data',c='k')
axs[0].plot(x_small,model_1(x_small,*popt_1),label='Fitted model',c='r')
axs[0].set_xlabel('Intrusion (mm)')
axs[0].set_ylabel('Force (kN)')
axs[0].grid()
axs[0].legend()

axs[1].scatter(x_large,y_large,label='Data',c='k',s=2)
axs[1].plot(x_large,model_2(x_large,*popt_2),label='Fitted model',c='r')
axs[1].set_xlabel('Intrusion (mm)')
axs[1].set_ylabel('Force (kN)')
axs[1].grid()
axs[1].legend()
```

Out [ ]:
```
<matplotlib.legend.Legend at 0x1370fc8b0>
```



Question 10

To compute the goodness of the fit R^2 we code a simple function to avoid coding the same equation several times.

This function is then taking as input:

- the measured values y
- the predicted values from the considered model

The goodness of the linear and polynomial models are then evaluated in an independent cell.

In [ ]:
```
def compute_R2(ym,yc):
    Se = np.sum((yc-ym)**2.0)
    St = np.sum((yc-np.mean(ym))**2.0)
    return 1.0-Se/St
```

In [ ]:
```
print('R^2 for the small intrusion model: {}'.format(compute_R2(y_small,model_1(x_small,*popt_1))))
print('R^2 for the large intrusion model: {}'.format(compute_R2(y_large,model_2(x_large,*popt_2))))
```

```
R^2 for the small intrusion model: 0.8603835650440311
R^2 for the large intrusion model: 0.9922877340248164
```

Bonus questions

Question 12

The parameters for the linear model are found using the formula presented during the lectures. These formulas are implemented using a dedicated function which is thereafter applied to the small intrusion dataset.

In [ ]:
```
def linear(x,y):
    n = float(len(x))
    a = (np.sum(x*y)-(np.sum(x)*np.sum(y))/n)/(np.sum(x**2.0)-(np.sum(x)**2.0/n))
    b = np.mean(y)-a*np.mean(x)
    return a,b
```

In [ ]:
```
a,b = linear(x_small,y_small)
popt_1_A = [a,b]
```

Question 13

Similar to question 12, the formula from the lecture is implemented into a dedicated function and is applied to the large intrusion dataset in a dedicated cell.

In [ ]:
```
def polynom(x,y):
    # Setup matrix
    A,C = np.zeros((3,3)),np.zeros((3))
    n = len(x)
    # Compute parameters according to analytical solutions
    A[0,0] = n
    A[1,0] = np.sum(x)
    A[2,0] = np.sum(x**2.0)
    #
    A[0,1] = np.sum(x)
    A[1,1] = np.sum(x**2.0)
    A[2,1] = np.sum(x**3.0)
    #
    A[0,2] = np.sum(x**2.0)
    A[1,2] = np.sum(x**3.0)
    A[2,2] = np.sum(x**4.0)
    #
    C[0] = np.sum(y)
    C[1] = np.sum(x*y)
    C[2] = np.sum(x**2.0*y)
    #
    b = solve(A,C,)
    return b[0],b[1],b[2]
```

In [ ]:
```
a1,a2,a3 = polynom(x_large,y_large)
popt_2_A = [a1,a2,a3]
```

Question 14

The parameters obtained for the different models (linear and polynomial) using the scipy module and the formulas from the lecture are printed in the next cell.

We can see that the obtained parameters are nearly identical between the two approaches.

In [ ]:
```
parameters_1 = ['a','b']
print('Parameters for the small intrusion levels')
for parameter,p1,p2 in zip(parameters_1,popt_1_A,popt_1):
    print('{} = {} , {}'.format(parameter,p1,p2))

parameters_2 = ['a1','a2','a3']
print('Parameters for the large intrusion levels')
for parameter,p1,p2 in zip(parameters_2,popt_2_A,popt_2):
    print('{} = {} , {}'.format(parameter,p1,p2))
```

```
Parameters for the small intrusion levels
a = 2.942711286010639 , 2.9427112885592925
b = -0.4244548995268926 , -0.42445489956797395
Parameters for the large intrusion levels
a1 = -2.0611610176692648 , -2.0611610164320394
a2 = 3.802585935744651 , 3.8025858264909016
a3 = 0.8763335770389465 , 0.8763335910603329
```

Question 15

The parameters obtained using the formulas from the lecture gives the following R^2. We can see that the values are again nearly identical to the one obtained with the scipy module.

In [ ]:
```
print('R^2 for the small intrusion model: {}'.format(compute_R2(y_small,model_1(x_small,*popt_1_A))))
print('R^2 for the large intrusion model: {}'.format(compute_R2(y_large,model_2(x_large,*popt_2_A))))
```

```
R^2 for the small intrusion model: 0.8603835656160477
R^2 for the large intrusion model: 0.9922877340149213
```