4

July 27, 2024

0.1 Linear FIT

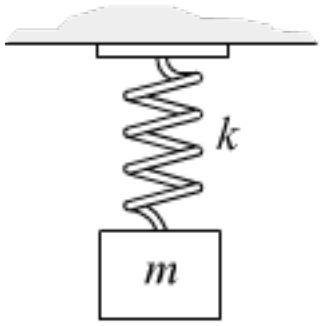
This notebook would like to be a tool for a lesson on the linear fit of experimental data, introducing the χ^2 test reduced.

1 Hooke's law

Hooke's law defines the relationship between an applied force and the elongation of the spring subjected to the force. In the case in question, we will assume the spring is hanging on a ceiling with a body attached.

[ ]:
```
from IPython.display import Image
Image("molla.png",width=200, height=200)
```

[ ]:



Hooke's law is formally expressed with

$$F = -k \cdot \Delta l$$

with k the elastic constant.

In the present case this relationship becomes

$$mg = -k \cdot (l - l_0)$$

and therefore

$$l = \frac{g}{k} \cdot m + l_0$$

which expresses the linear dependence between the final length of the spring and the mass hanging on it.

Now let's make some measurements and import the data

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import ipywidgets as widgets
from ipywidgets import interact, interactive, fixed, interact_manual
from IPython.display import display
```

[ ]:
```
import csv
with open("./linear_fit2.csv", newline="", encoding="ISO-8859-1") as filecsv:
    f = csv.reader(filecsv,delimiter=",")
    header = next(f)
    print(header)
```

```
['m', 'l', 's_l']
```

[ ]:
```
import io
df = pd.read_csv("linear_fit2.csv")
```

[ ]:
```
df
```

[ ]:

	m	l	s_l
0	200	3.1	0.3
1	300	3.5	0.3
2	400	3.9	0.3
3	500	4.8	0.3
4	600	5.4	0.3
5	700	5.5	0.3
6	800	6.6	0.3
7	900	7.4	0.3

[ ]:
```
%pylab inline
from scipy.optimize import curve_fit
```

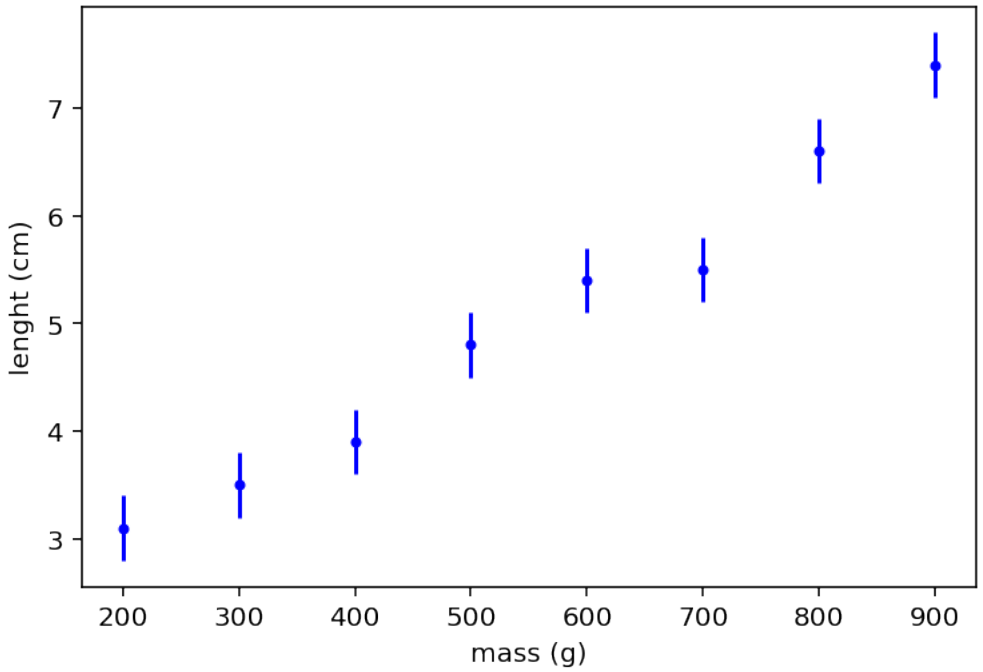
```
Populating the interactive namespace from numpy and matplotlib

/usr/local/lib/python3.6/dist-packages/IPython/core/magics/pylab.py:160:
UserWarning: pylab import has clobbered these variables: ['interactive', 'f']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

[ ]:
```
errorbar(df['m'],df['l'],df['s_l'],fmt='b.')
ylabel("lenght (cm)")
```

```
xlabel("mass (g)");
```

[ ]:



We now want to determine the best straight line that describes the trend of the experimental data

[ ]:
```
def f(m,q):
  errorbar(df['m'],df['l'],df['s_l'],fmt='b.')
  x = np.linspace(0, 900)
  xlabel("mass (g)")
  ylabel("lenght (cm)")
  plt.plot(x, m*x+q, color = "yellow",linewidth=5,label="Linear fit")
  plt.legend(loc="upper left")
  plt.show();

interactive_plot = widgets.interact(f, m=(0.001, 0.01,0.0005),q=(-1,4,0.1))
display(interactive_plot);
```

[ ]:
```
interactive(children=(FloatSlider(value=0.0055000000000000005, description='m',
max=0.01, min=0.001, step=0.00...
```

[ ]:
```
<function __main__.f(m, q)>
```

As can be seen by assigning different values to the two parameters of the linear model, the determination of the best straight line is not immediate. Let's understand from a mathematical point of view how it is possible to obtain it.

1.1 χ^2 reduced method

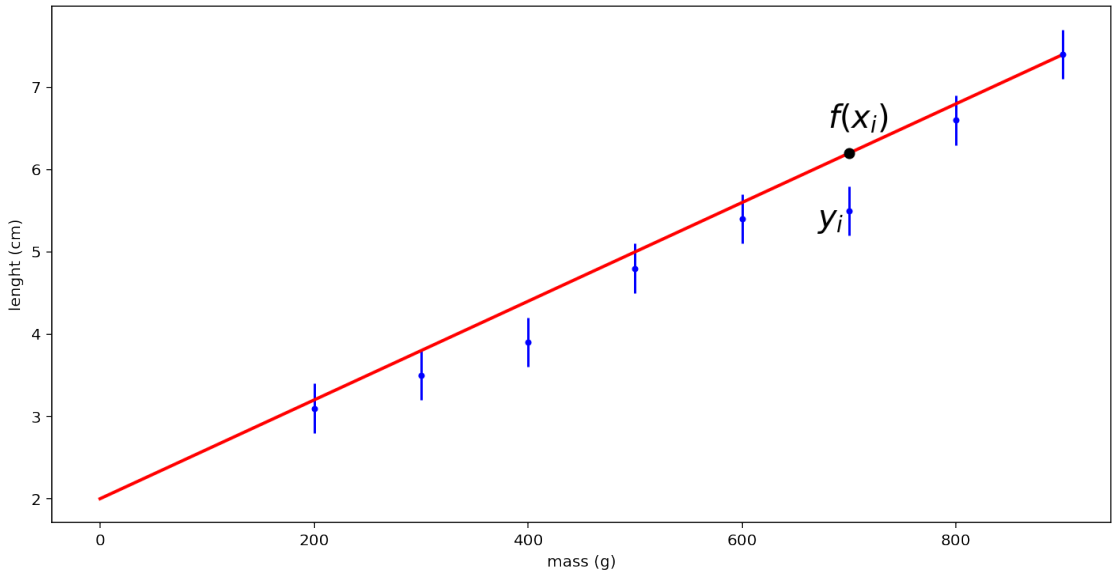
Suppose, therefore, that we have made a number $ N $ of measures of quantities $ x $ and $ y $ which are thought or known to be linked by a functional relationship $y = f(x, p)$, where $ p $ corresponds to a certain number of unknown parameters.

Performing a Fit means giving an estimate of the "true" value of these parameters by looking for those values for which the distance between the values $ y_ {i} $ measured and those theoretically predicted by $ f (x_ {i}) $ is minimal.

[ ]:
```
fig, ax = plt.subplots(figsize=(12, 6))
errorbar(df['m'],df['l'],df['s_l'],fmt='b.')
x = np.linspace(0, 900)
plt.plot(x, 0.006*x+2, color = "red",linewidth=2)
x=700
y=6.2
plt.plot(x,y, color="black", marker="o")
plt.text(680, 6.5, r'$f(x_{i})$',
         fontsize=20)
plt.text(670, 5.3, r'$y_{i}$',
         fontsize=20)

ylabel("lenght (cm)")
xlabel("mass (g)");
```

[ ]:



It may seem appropriate to set the sum of the residuals equal to zero

$$\sum_{i=0}^{n}(y_i - f(x_i)) = 0$$

However, in this case, having to take into account also negative values, the sum may be zero even if the function is not able to describe the behaviour of the experimental data, as can be seen in the following graph.

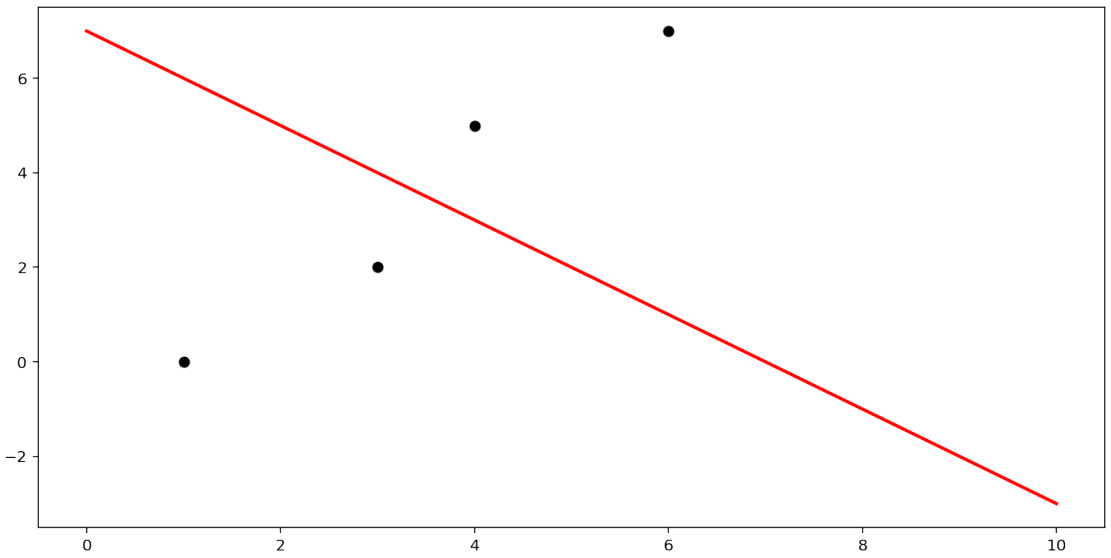
[ ]:
```
fig, ax = plt.subplots(figsize=(12, 6))
x = np.linspace(0, 10)

plt.plot(x, -x+7, color = "red",linewidth=2)
x=[1,3,4,6]
y=[0,2,5,7]

plt.plot(x,y, color="black", marker="o",linestyle="")
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f96921a7198>]
```

[ ]:



To solve this problem, we can minimize the relationship (1)

$\forall a \geq 0$

$\forall b \geq 0$

$$a \leq b \Leftrightarrow a^2 \leq b^2$$

hence

$$0 \leq |\sum_{i=0}^{n}(y_i - f(x_i))| \leq \sum_{i=0}^{n}(y_i - f(x_i))^2 \quad (1)$$

In general this requires higher computational knowledge, but in this case we want to minimize

$$S(m,q) = \sum_{i=0}^{n} (y_i - mx_i - q)^2$$

which means determining the vertex of the corresponding parabola (see https://www.mat.unical.it/~pls/Slides-FDellac.pdf)

It is interesting to note that we can carry out these calculations from the second year when students encounter second degree problems

To be more precise, we have to minimize

$$\chi^2 = \sum_{i=0}^{n} (\frac{y_i - f(x_i)}{\sigma_y})^2$$

with σ_y standard deviation, that is

$$\sigma_y = \frac{\sum (y_i - \overline{y})}{n}$$

See now how to obtain this with Python

http://sigmaquality.pl/uncategorized/fit-curve-to-data_-scipy-optimize-curve_fit-en220120201529/

[ ]:
```
from scipy.optimize import curve_fit
#The scipy function "scipy.optimize.curve_fit" adopts the type of curve to which␣
 ↪you want to fit the data

def linear_model(x,m,q): #parameter[0]=m, parameter[1]=q
    return m*x+q

y= df["l"]
x= df["m"]
x_prevision= range(1100) # It allows to describe the behavior beyond the real␣
 ↪data

lin_fit = curve_fit(linear_model,x,y,sigma=df['s_l'])
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.
 ↪html
print(lin_fit)
```

```
(array([0.00607143, 1.68571428]), array([[ 1.40022674e-07, -7.70124708e-05],
       [-7.70124708e-05,  4.97080494e-02]]))
```

[ ]:
```
errorbar(df['m'],df['l'],df['s_l'],fmt='b.')
plt.plot(x_prevision, [linear_model(i,lin_fit[0][0],lin_fit[0][1]) for i in␣
 ↪x_prevision], label="linear model" )
plt.legend()
```

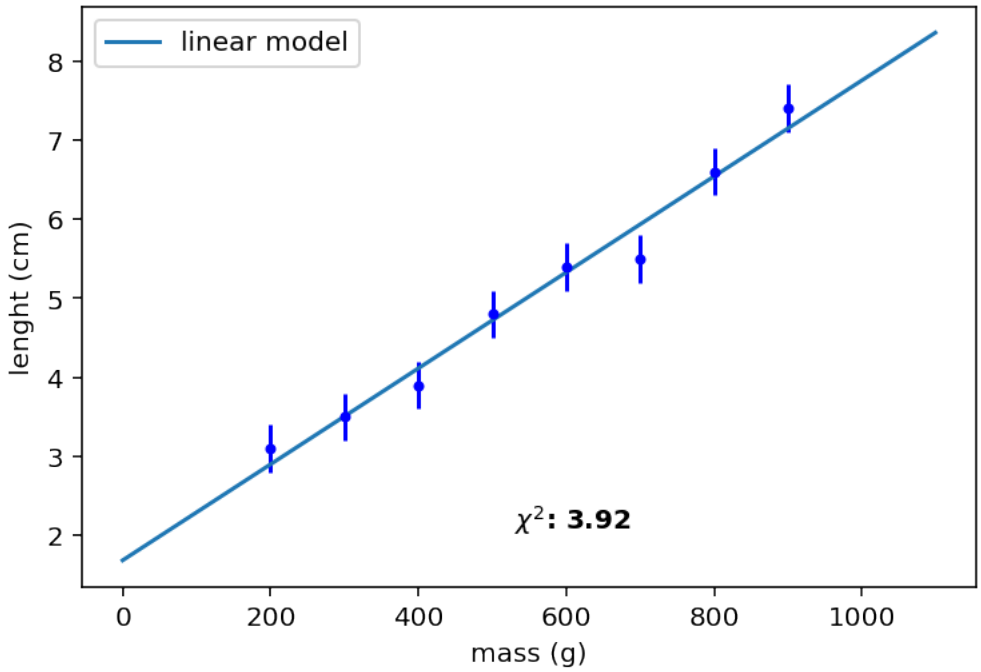
```
plt.xlabel("mass (g)")
plt.ylabel("lenght (cm)")

print("m: %.2f"%(lin_fit[0][0]))
print("q: %.2f"%(lin_fit[0][1]))

chisq = sum((df['l'] - linear_model(df['m'],lin_fit[0][0],lin_fit[0][1]))**2/
 ↪df['s_l']**2)
figtext(0.5,0.2,"$\chi^{2}$: %.2f"%chisq,fontweight="bold");
```

```
m: 0.01
q: 1.69
```

[ ]:



If $\chi^2 \leq n$, the fit can be considered correct.

[ ]: