Week 3: Fitting Data to Models

February 5, 2019

Abstract

This week's Python assignment will focus on the following topics:

- Reading data from files and parsing them
  - Analysing the data to extract information
  - Study the effect of noise on the fitting process
  - Plotting graphs

Plotting graphs

You can use `matplotlib` to plot sophisticated graphs in Python.

Here is a simple program to plot $J_0(x)$ for $0 < x < 10$. (type it in and see)

```
In [48]: from pylab import *; from scipy.special import *
In [49]: x=arange(0,10,.1)
In [50]: y=jv(0,x)
In [51]: plot(x,y)
In [52]: show()
```

The `import` keyword imports a module as already discussed. The "pylab" module is a super module that imports everything needed to make python seem to be like Matlab.

The actual code is four lines. One defines the x values. The second computes the Bessel function. The third plots the curve while the last line displays the graphic.

The plot command has the following syntax:

```
plot(x1,y1,style1,x2,y2,style2,...)
```

helping you to plot multiple curves in the same plot.

The Assignment

1. The site has a python script `generate_data.py`. Download the script and run it to generate a set of data. The data is written out to a file whose name is `fitting.dat`.

1 ⟨* 1⟩≡

```
# script to generate data files for the least squares assignment
from pylab import *
import scipy.special as sp
N=101 # no of data points
k=9   # no of sets of data with varying noise
# generate the data points and add noise
```

```
t=linspace(0,10,N)      # t vector
y=1.05*sp.jn(2,t)-0.105*t # f(t) vector
Y=meshgrid(y,ones(k),indexing='ij')[0] # make k copies
scl=logspace(-1,-3,k)  # noise stdev
n=dot(randn(N,k),diag(scl)) # generate k vectors
yy=Y+n    # add noise to signal
# shadow plot
plot(t,yy)
xlabel(r'$t$',size=20)
ylabel(r'$f(t)+n$',size=20)
title(r'Plot of the data to be fitted')
grid(True)
savetxt("fitting.dat",c_[t,yy]) # write out matrix to file
show()
```

2. Load `fitting.dat` (look up `loadtxt`). The data consists of 10 columns. The first column is time, while the remaining columns are data. Extract these.

3. The data columns correspond to the function

$$f(t) = 1.05J_2(t) - 0.105t + n(t)$$

with different amounts of noise added. What is noise? For us here, it is random fluctuations in the value due to many small random effects. The noise is given to be normally distributed, i.e., its probability distribution is given by

$$\Pr(n(t)|\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{n(t)^2}{2\sigma^2}\right)$$

with σ given by

```
sigma=logspace(-1,-3,9)
```

Plot the curves in Figure 0 and add labels to indicate the amount of noise in each. (Look up `plot` and `legend`.)

4. We want to fit a function to this data. Which function? The function has the same general shape as the data but with unknown coefficients:

$$g(t;A,B) = AJ_2(t) + Bt$$

Create a python function `g(t,A,B)` that computes $g(t;A,B)$ for given A and B. Plot it in Figure 0 for $A = 1.05$, $B = -0.105$ (this should be labelled as the true value.)

5. Generate a plot of the first column of data with error bars. Plot every 5th data item to make the plot readable. Also plot the exact curve to see how much the data diverges.

   This is not difficult. Suppose you know the standard deviation of your data and you have the data itself, you can plot the error bars with red dots using

```
errorbar(t,data,stdev,fmt='ro')
```

   Here, 't' and 'data' contain the data, while 'stdev' contains σ_n for the noise. In order to show every fifth data point, you can instead use

```
errorbar(t[::5],data[::5],stdev,fmt='ro')
```

   After plotting the errorbars, plot the exact curve using the function written in part 4, and annotate the graph.

6. For our problem, the values of t are discrete and known (from the datafile). Obtain $g(t,A,B)$ as a column vector by creating a matrix equation:

$$g(t,A,B) = \begin{pmatrix} J_2(t_1) & t_1 \\ \cdots & \cdots \\ J_2(t_m) & t_m \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} \equiv M \cdot p$$

   Construct M and then generate the vector $M \cdot \begin{pmatrix} A_0 \\ B_0 \end{pmatrix}$ and verify that it is equal to `g(t,A0,B0)`. How will you confirm that two vectors are equal?

   Note: To construct a matrix out of column vectors, create the column vectors first and then use `c_[...]`. For instance, if you have two vectors `x` and `y`, use `M=c_[x,y]`

7. For $A = 0, 0.1, \ldots, 2$ and $B = -0.2, -0.19, \ldots, 0$, for the data given in columns 1 and 2 of the file, compute

$$\varepsilon_{ij} = \frac{1}{101} \sum_{k=0}^{101} (f_k - g(t_k, A_i, B_j))^2$$

This is known as the "mean squared error" between the data (f_k) and the assumed model. Use the first column of `data` as f_k for this part.

8. Plot a contour plot of ε_{ij} and see its structure. Does it have a minimum? Does it have several?

9. Use the Python function `lstsq` from `scipy.linalg` to obtain the best estimate of A and B. The array you created in part 6 is what you need. This is sent to the least squares program.

10. Repeat this with the different columns (i.e., columns 1 and i). Each column has the same function above, with a different amount of noise added as mentioned above. Plot the *error in the estimate* of A and B for different data files versus the noise σ. Is the *error in the estimate* growing linearly with the noise?

11. Replot the above curves using `loglog`. Is the error varying linearly? What does this mean?

Linear Fitting to Data

Perhaps the most common engineering use of a computer is the modelling of real data. That is to say, some device, say a tachometer on an induction motor, or a temperature sensor of an oven or the current through a photodiode provides us with real-time data. This data is usually digitised very early on in the acquisition process to preserve the information, leaving us with time sequences,

$$(t, \mathbf{x}) = \{t_i, x_i\}_{i=1}^{N}$$

If the device has been well engineered, and if the sensor is to be useful in diagnosing and controlling the device, we must also have a model for the acquired data:

$$f(t; p_1, ..., p_N)$$

For example, our model could be

$$f(t; p_1, p_2) = p_1 + p_2 \sin(\pi t^2)$$

Our task is to accurately predict $p_1, ..., p_N$ given the real-time data. This would allow us to say things like, "Third harmonic content in the load is reaching dangerous levels".

The general problem of the estimation of model parameters is going to be tackled in many later courses. Here we will tackle a very simple version of the problem. Suppose my model is "linear in the parameters", i.e.,

$$f(t; p_1, ..., p_N) = \sum_{i=1}^{N} p_i F_i(t)$$

where $F_i(t)$ are arbitrary functions of time. Our example above fits this model:

$$\begin{aligned} p1, p2: &\quad \text{parameters to be fitted} \\ F_1: &\quad 1 \\ F_2: &\quad \sin(\pi t^2) \end{aligned}$$

For each measurement, we obtain an equation:

$$\begin{array}{lll} 1 \cdot p_1 & +\sin(\pi t_1^2) p_2 & = x_1 \\ 1 \cdot p_1 & +\sin(\pi t_2^2) p_2 & = x_2 \\ \ldots & \ldots & \ldots \\ 1 \cdot p_1 & +\sin(\pi t_M^2) p_2 & = x_M \end{array}$$

Clearly the general problem reduces to the inversion of a matrix problem:

$$\begin{pmatrix} F_1(t_1) & F_2(t_1) & ... & F_N(t_1) \\ F_1(t_2) & F_2(t_2) & ... & F_N(t_2) \\ ... & ... & ... & ... \\ F_1(t_M) & F_2(t_M) & ... & F_N(t_M) \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ ... \\ p_N \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_M \end{pmatrix}$$

However, the number of parameters, N, is usually far less than the number of observations, M. In the absence of measurement errors and noise, any non-singular $N \times N$ submatrix can be used to determine the coefficients p_i. When noise is present, what can we do?

The matrix equation now becomes

$$F \cdot \vec{p} = \vec{x}_0 + \vec{n} = \vec{x}$$

where $\vec{n}$ is the added noise. $\vec{x}_0$ is the (unknown) ideal measurement, while $\vec{x}$ is the actual noisy measurement we make. We have to assume something about the noise, and we assume that it is as likely to be positive as negative (zero mean) and it has a standard deviation σ_n. Clearly the above equation cannot be exactly satisfied in the presence of noise, since the rank of $\mathbf{F}$ is N wheras the number of observations is $M \gg N$.

We also make a very important assumption, namely that the noise added to each observation x_i, namely n_i, is "independent" of the noise added to any other observation. Let us see where this gets us.

We wish to get the "best" guess for $\vec{p}$. For us, this means that we need to minimize the L_2 norm of the error. The error is given by

$$\varepsilon = F \cdot \vec{p} - \vec{x}$$

The norm of the error is given by

$$\vec{\varepsilon}^T \cdot \vec{\varepsilon} = \sum_i \varepsilon_i^2$$

This norm can we written in matrix form as

$$(F \cdot \vec{p} - \vec{x})^T \cdot (F \cdot \vec{p} - \vec{x})$$

which is what must be minimized. Writing out the terms

$$\vec{p}^T \left(F^T F\right) \vec{p} + \vec{x}^T \vec{x} - \vec{p}^T F^T \vec{x} - \vec{x}^T F \vec{p}$$

Suppose the minimum is reached at some $\vec{p}_0$. Then near it, the above norm should be greater than that minimum. If we plotted the error, we expect the surface plot to look like a cup. The gradient of the error at the minimum should therefore be zero. Let us take the gradient of the expression for the norm. We write $F^T F = M$, and write out the error in "Einstein notation":

$$\text{error} = p_i M_{ij} p_j + x_j x_j - p_i F_{ij}^T x_j - x_i F_{ij} p_j$$

Here we have suppressed the summation signs over i and j. If an index repeats in an expression, it is assumed to be summed over. Differentiating with respect to p_k, *and assuming that* $\partial p_i / \partial p_k = \delta_{ik}$, we get

$$\begin{aligned} \frac{\partial \text{error}}{\partial p_k} &= \delta_{ki} M_{ij} p_j + p_i M_{ij} \delta_{jk} - \delta_{ki} F_{ij}^T x_j - x_i F_{ij} \delta_{jk} = 0 \\ &= M_{kj} p_j + p_i M_{ik} - F_{kj}^T x_j - x_i F_{ik} \\ &= \sum_j \left(M_{kj} + M_{jk}\right) p_j - 2 \sum_j F_{kj}^T x_j \end{aligned}$$

Now the matrix M is symmetric (just take the transpose and see). So the equation finally becomes, written as a vector expression

$$\nabla \text{error}(\vec{p}) = 2 \left(F^T F\right) \vec{p}_0 - 2 F^T \vec{x} = 0$$

i.e.,

$$\vec{p}_0 = \left(F^T F\right)^{-1} F^T \vec{x}$$

This result is very famous. It is so commonly used that scientific packages have the operation as a built in command. In Python, it is a library function called `lstsq`:

```
from scipy.linalg import lstsq
p,resid,rank,sig=lstsq(F,x)
```

where `p` returns the best fit, and `sig`, `resid` and `rank` return information about the process.

In Scilab or Matlab, it take the form:

```
p0 = F\x;
```

Here `F` is the coefficient matrix and `x` is the vector of observations. When we write this, Scilab is actually calculating

```
p0=inv(F'*F)*F'*x;
```

What would happen if the inverse did not exist? This can happen if the number of observations are too few, or if the vectors $f_i(x_j)$ (i.e., the columns of F) are linearly dependent. Both situations are the user's fault. He should have a model with linearly independent terms (else just merge them together). And he should have enough measurements.

`p0` obtained from the above formula is a prediction of the exact parameters. How accurate can we expect `p0` to be? If `x` were `x0` we should recover the exact answer limited only by computer accuracy. But when noise is present, the estimate is approximate. The more noise, the more approximate.

Where did we use all those properties of the noise? We assumed that the noise in different measurements was the same when we defined the error norm. Suppose some measurements are more noisy. Then we should give less importance to those measurements, i.e., weight them less. That would change the formula we just derived. If the noise samples were not independent, we would need equations to explain just how they depended on each other. That too changes the formula. Ofcourse, if the model is more complicated things get really difficult. For example, so simple a problem as estimating the *frequency* of the following model

$$y = A\sin\omega t + B\cos\omega t + n$$

is an extremely nontrivial problem. That is because it is no longer a "linear" estimation problem. Such problems are discussed in advanced courses like *Estimation Theory*. The simpler problems of correlated, non-uniform noise will be discussed in *Digital Communication* since that theory is needed for a cell phone to estimate the signal sent by the tower.

In this assignment (and in this course), we assume independent, uniform error that is normally distributed. For that noise, as mentioned above, Python already provides the answer:

```
p=lstsq(F,x)
```

Even with all these simplifications, the problem can become ill posed. Let us look at the solution again:

$$\vec{p}_0 = \left(F^T F\right)^{-1} F^T \vec{x}$$

Note that we have to invert $F^T F$. This is the coefficient matrix. For instance, if we were trying to estimate A and B of the following model:

$$y = A\sin\omega_0 t + B\cos\omega_0 t$$

(not the frequency - that makes it a nonlinear problem), the matrix F becomes

$$F = \begin{pmatrix} \sin\omega_0 t_1 & \cos\omega_0 t_1 \\ \sin\omega_0 t_2 & \cos\omega_0 t_2 \\ \cdots & \cdots \\ \sin\omega_0 t_n & \cos\omega_0 t_n \end{pmatrix}$$

Hence, $F^T F$ is a 2 by 2 matrix with the following elements

$$F^T F = \begin{pmatrix} \sum\sin^2\omega_0 t_i & \sum\sin\omega_0 t_i\cos\omega_0 t_i \\ \sum\cos\omega_0 t_i\sin\omega_0 t_i & \sum\cos^2\omega_0 t_i \end{pmatrix}$$

Whether this matrix is invertible depends only on the functions $\sin\omega_0 t$ and $\cos\omega_0 t$ and the times at which they are sampled. For instance, if the $t_k = 2k\pi$, the matrix becomes

$$F^T F = \begin{pmatrix} 0 & 0 \\ 0 & n \end{pmatrix}$$

since $\sin\omega_0 t_k \equiv 0$ for all the measurement times. Clearly this is not an invertible matrix, even though the *functions* are independent. Sometimes the functions are "nearly" dependent. The inverse exists, but it cannot be accurately calculated. To characterise this, when an inverse is calculated, the "condition number" is also returned. A poor condition number means that the inversion is not to be depended on and the estimation is going to be poor.

We will not use these ideas in this lab. Here we will do a very simple problem only, to study how the **amount** of noise affects the quality of the estimate. We will also study what happens if we use the wrong model.