Week 4: Fitting Data to Models

February 19, 2018

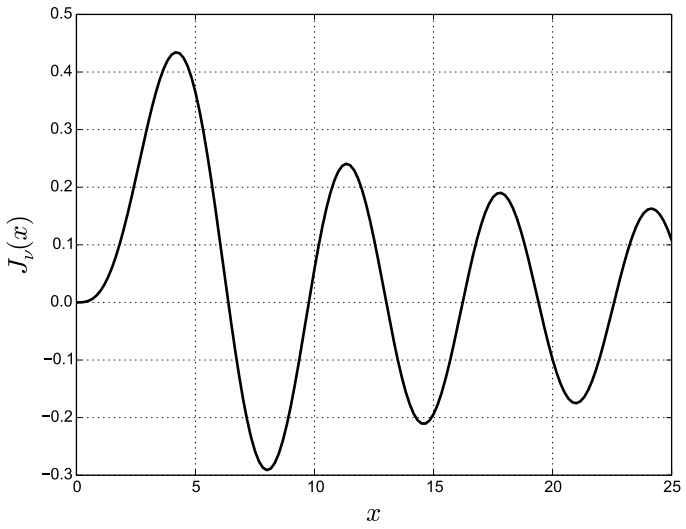
Abstract

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

- Study fitting of data using models
- Study the effect of noise on the fitting process

1 The Assignment

1. Bessel functions are very often seen in cylindrical geometry. They come in several forms, and we will look at the Bessel function of the first type, $J_\nu(x)$.



For large x,

$$J_\nu(x) \approx \sqrt{\frac{2}{\pi x}} \cos\left(x - \frac{\nu\pi}{2} - \frac{\pi}{4}\right) \tag{1}$$

(a) Generate a vector of 41 values from 0 to 20 and obtain a vector of $J_1(x)$ values.

(b) for different $x_0 = 0.5, 1, \ldots, 18$ extract the subvectors of x and $J_1(x)$ that correspond to $x \geq x_0$. For each x_0, construct the matrix corresponding to

$$A\cos(x_i) + B\sin(x_i) \approx J_1(x_i)$$

and obtain the best fit A and B. Obtain the ϕ corresponding to the solution (divide by $\sqrt{A^2+B^2}$ and identify $A/\sqrt{A^2+B^2}$ to be $\cos\phi$). Hence predict ν in Eq. (**??**).

(c) Repeat with a model that also includes the amplitude:

$$A\frac{\cos(x_i)}{\sqrt{x_i}} + B\frac{\sin(x_i)}{\sqrt{x_i}} \approx J_1(x_i)$$

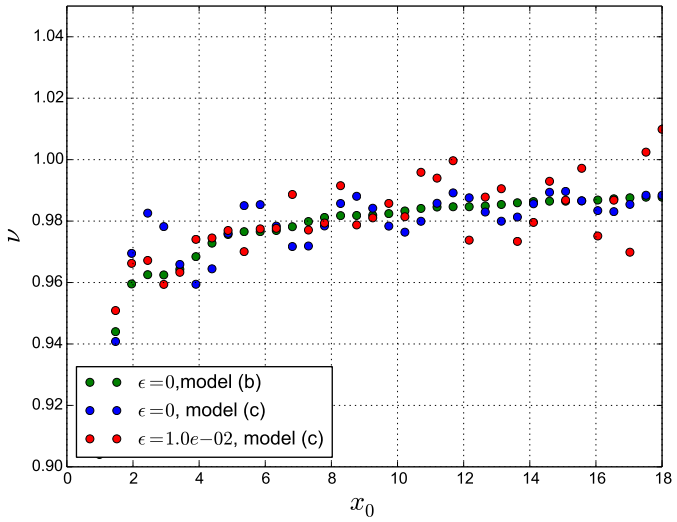
How do the two fits compare?

(d) Convert the above to a function that can be called via

```
nu=calcnu(x,x0,'r',eps,model)
```

where eps is the amount of noise added (next problem) and model is whether to run (b) or (c) above.

(e) Now add noise using *eps*randn(10)* where *eps* is the amount of the noise. See the effect on the fit. Plot the fit for eps of 0.01.



You should get something like the above. The blue dots are for problem (b), while the green dots are for problem (c). Both work but green works better, since it is a better model. When noise is added you get the red dots. Given that the correct answer is 1.0 you can see the problem - adding noise makes things worse at large x_0. But large x_0 is where the noiseless fit is best.

(f) Try varying the number of measurements (keeping the range of x the same). What happens?

(g) Discuss the effect of model accuracy, number of measurements, and the effect of noise on the quality of the fit.

2 Python Details

2.1 Creating Functions

This was discussed last time

```
def f(x):
   return sin(x)
```

To return multiple return values, return a "tuple" containing the values:

```
def f(x):
   return (sin(x),cos(x))
```

3 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:

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

For each measurement, we obtain an equation:

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

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 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. 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

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

i.e.,

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

This result is so commonly used that scientific packages have the operation as a built in command. In Python it take the form:

```
p0 = lstsq(F,x)
```

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

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 probably 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 and in Measurements), we assume independent, uniform error that is normally distributed. For that noise, as mentioned above, Python already provides the answer:

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

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.

5 ⟨* 5⟩≡
⟨*q1* (never defined)⟩