Introduction to Scientific Computing
Lecture 4

Professor Hanno Rein

Last updated: September 24, 2016

4 Fitting

4.1 Linear least square fit

One task where one needs to solve a linear set of equations is a least square fit. There is in general no closed solution for a non-linear least square fit and it requires iteration. In this section we will keep it simple and only discuss the linear version of a least square fit.

Suppose we are given a set of 500 temperature measurements. The data was taken here at the UTSC weather station over several months. The figure below shows the temperature as a function of time of day. One can clearly see a trend, temperatures are warmest around 3pm on an average day. The above data is given to us as a set of x and y values with N pairs.

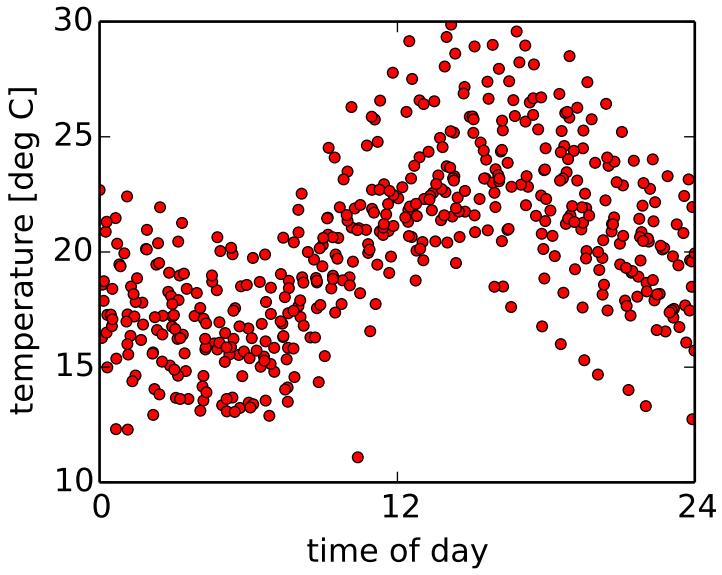

Figure 1: Temperature measurements over several month as a function of time of day. Source: UTSC weather station, `http://weather.utsc.utoronto.ca`.

Let us assume the data is described by a function of the following form

$$f(t) = a_0 + a_1 \sin\left(\frac{t}{24}2\pi\right) + a_2 \cos\left(\frac{t}{24}2\pi\right)$$

We use one hour as the unit of time. Also note that the sin and cos functions expect an argument in radians, not degrees. This is a convention that we'll use for the remainder of the course. Finally, note that we use two sin functions instead of one. We really only want one sin function, but we don't know the phase. We can add a phase argument to a sine function, but then the problem would not be linear anymore. Instead, we use two sine functions and the identity

$$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta,$$

which allows us to convert one sin function with a phase to two sin and cos functions without a phase.

If we had as many unknowns as equations (we have 3 and 500 respectively) we could just write down the equation system as

$$\begin{aligned}
y_0 &= f(x_0) = a_0 + a_1\sin\left(\frac{x_0}{24}2\pi\right) + a_2\cos\left(\frac{x_0}{24}2\pi\right)\\
y_1 &= f(x_1) = a_0 + a_1\sin\left(\frac{x_1}{24}2\pi\right) + a_2\cos\left(\frac{x_1}{24}2\pi\right)\\
y_2 &= f(x_2) = a_0 + a_1\sin\left(\frac{x_2}{24}2\pi\right) + a_2\cos\left(\frac{x_2}{24}2\pi\right)
\end{aligned}$$

or in matrix form

$$\begin{pmatrix} y_0\\ y_1\\ y_2\end{pmatrix} = \begin{pmatrix} 1 & \sin\left(\frac{x_0}{24}2\pi\right) & \cos\left(\frac{x_0}{24}2\pi\right)\\ 1 & \sin\left(\frac{x_1}{24}2\pi\right) & \cos\left(\frac{x_1}{24}2\pi\right)\\ 1 & \sin\left(\frac{x_2}{24}2\pi\right) & \cos\left(\frac{x_2}{24}2\pi\right)\end{pmatrix}\begin{pmatrix} a_0\\ a_1\\ a_2\end{pmatrix}.$$

These equations have exactly one solution, but only if we are given exactly 3 data value pairs for 3 unkowns. Most of the times, we have a few unknown parameters (in our case a_0, a_1 and a_2) but are given many more, let's say N, data value pairs. Then, we have an over-defined system. We solve this by performing a *fit*. We want to minimize the average (vertical) distance of any point (x_i, y_i) from our function f. One way to do this is a least square fit.

In matrix and vector notation, this relates to

$$\begin{pmatrix} y_0\\ y_1\\ \vdots\\ y_{N-1}\end{pmatrix} = \underbrace{\begin{pmatrix} 1 & \sin\left(\frac{x_0}{24}2\pi\right) & \cos\left(\frac{x_0}{24}2\pi\right)\\ 1 & \sin\left(\frac{x_1}{24}2\pi\right) & \cos\left(\frac{x_1}{24}2\pi\right)\\ & \vdots & \\ 1 & \sin\left(\frac{x_{N-1}}{24}2\pi\right) & \cos\left(\frac{x_{N-1}}{24}2\pi\right)\end{pmatrix}}_{\equiv C}\begin{pmatrix} a_0\\ a_1\\ a_2\end{pmatrix} + \begin{pmatrix} e_0\\ e_1\\ \vdots\\ e_{N-1}\end{pmatrix}$$

The values e are errors or residuals. This is what we want to minimize. If we have a perfect fit, $e_i = 0$ for all i. Let us define the total residual as

$$S = \sum_{i=0}^{N-1} e_i^2$$

and switch to a slightly more general case: instead of assuming exactly 3 free parameters, we now assume we have m free parameters. When S is minimized, its gradient vector is zero. The derivatives of S can be easily calculated (we replace e_i by the values from the above matrix equation) to get

$$\begin{aligned}
0 &= \frac{\partial S}{\partial a_j} = 2\sum_{i=0}^{N-1} e_i\frac{\partial e_i}{\partial a_j} && \forall j\\
&= 2\sum_{i=0}^{N-1}\left[y_i - \sum_{k=0}^{m-1}(a_k C_{ik})\right]C_{ij}
\end{aligned}$$

Rearranging gives

$$\sum_{i=0}^{N-1} C_{ij} y_i = \sum_{i=0}^{N-1} \sum_{k=0}^{m-1} C_{ik} C_{ij} a_k$$

In matrix notation this is

$$\underbrace{C^T \cdot y}_{b} = \underbrace{\left(C^T C\right)}_{A} \cdot a$$

We have now a linear equation system. We want to solve it for a, our coefficient a_0, a_1 and a_2 in the above example. Note that calculating the left side is trivial (it is just a matrix-vector multiplication). But we then need to solve a linear system of equations to get a. You already know how to do this, with Gaussian Elimination.

Before me move on, let's have a look at the dimensions of the vectors and matrices. The matrix C has dimensions of $N \times m$, the vector y is N elements long. Thus the left hand side (and therefore the right hand side) is a vector of length m. The matrix $C^T C$ is what goes into our Gaussian elimination algorithm. It is only a $m \times m$ matrix. So its size is only determined by the number of free parameters, not the number of data-points we want to fit. Usually $m \ll N$. This is important because Gaussian Elimination will be slow for large matricies as it scales as $O(m^3)$.

5 Root finding algorithms

5.1 Intermediate Value Theorem

The intermediate value theorem is a fundamental mathematical theorem which we will not prove, but it will be important for the methods we derive and you will need to understand its consequences.

Let $I = [a, b]$ be an interval with real numbers a, b and let f be a continuous function from the interval I into the space of real numbers, $f : I \to \mathbb{R}$. For any number c for which

$$f(a) < c < f(b),$$

then there is a number $d \in [a, b]$ such that

$$f(d) = c.$$

5.2 The problem of root finding

We have already encounter one problem of root finding, the least square fit. In that case, we found the root of the equation

$$\frac{\partial S(a)}{\partial a} = 0$$

Because we only considered a *linear* least square fit, we were able to solve the problem exactly using Gaussian elimination. In general it will not be possible to solve a root finding problem this easily (or at all!).

So what do we mean by root finding? A root finding algorithm finds the value x for which a given function $f(x)$ is zero. In the least square fit example, the function $f(x)$ was $\partial S/\partial a$ and the variable x was the vector a.

The following might seem obvious but its worth pointing out. The way we defined root finding, we are looking for a function argument that makes a function evaluate to zero. However, this is equivalent to trying to solve the equation

$$f(x) = g(x)$$

for two arbitrary function f and g. We just need to bring one to the other side to get it back into the canonical form $f(x) - g(x) = 0$.

5.3 Bisection method

All the methods we will discuss to solve the root finding problem are iterative. That means we start with a guess and improve upon our guess during every iteration. The bisection method is one such example.

Let's formulate the problem a bit more precisely. You are given a one dimensional real function on the interval $I = [a, b]$. You can assume that the function is continuous. The problem is then to find the value $d \in [a, b]$ for which $f(d) = c$. You are given c, usually we have $c = 0$.

The bisection method works as follows. We first divide the interval $[a, b]$ into two intervals $[a, \frac{a+b}{2}]$ and $[\frac{a+b}{2}, b]$. We then evaluate the function at the middle point $\frac{a+b}{2}$. If the value $f(\frac{a+b}{2})$ is smaller than c, then we know that our answer must lie in the upper interval $[\frac{a+b}{2}, b]$. Similarly, if the value $f(\frac{a+b}{2})$ is larger than c, then we know that our answer must lie in the lower interval $[a, \frac{a+b}{2}]$. We now have a better estimate of the value d. By simply repeating the process, we can make it more and more accurate. Because we divide the interval in half every time, the method converges quickly.

Below is an illustration of the bisection method and an implementation in pseudo code.

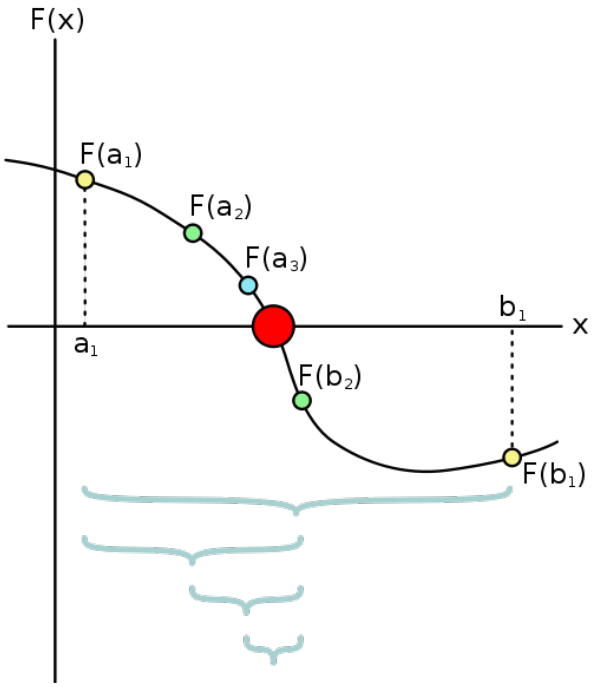

Figure 2: Bisection methods. Source: Wikipedia.

```
set a
set b
fa = f(a)
fb = f(b)
while ( (b-a) > epsilon ){
    m = (a+b)/2
    fm = f(m)
    if ( (fm>0 and fa<0) or (fm<0 and fa>0) ){
        b = m
        fb = f(b)
    }else{
        a = m
        fa = f(a)
    }
}
print [a:b]
```

Code 1: Pseudo code of the bisection method.

Ok. So the method improves the result at every iteration and the interval gets small. The question is: Using standard double floating point precision, how many iteration steps do we roughly need to converge to machine precision? The answer is 52 as there are 52 bits in the mantissa.

Note that we didn't use the actual value of $f(\frac{a+b}{2})$, just its sign. We're throwing away information that we could have used. There are much better methods than the bisection method that make use of this information.

5.4 Linear interpolation method

The next best thing one can do is the linear interpolation method, also known as the double false position method.

This method works similarly to the bisection method by shrinking the interval $[a, b]$, but instead of always dividing it in half, this method makes does a better estimate. First, it calculates the values $f(a)$ and $f(b)$. Then it interpolates the function f between these two values with a linear function. We can easily calculate the root of the linear function, let's call it c. We then use c as the new middle point to create two intervals $[a, c]$ and $[c, b]$. From there on we proceed the same way as in the bisection method. We calculate $f(c)$ and decide which interval is the interesting one. This method is slighlty faster at converging as we make use of the function evaluation $f(c)$ at the point c.

5.5 Newton's method

Newton's method is another iterative way of finding a root of a function $f(x)$. Contrary to the other methods we have talked about before, we have to be able to evaluate the derivate of f as well as the function f itself at any point. This is not always possible, depending on how complicated the function is. If no analytic function can be written down for the derivate, one can try to calculate the derivate numerically (e.g. using finite differences). However, evaluating the derivatives can be very expensive. On the positive side, if it is easy to calculate the derivative, then Newton's method is very fast and converges very quickly.

Let's spend a bit of time discussing how to numerically calculate a derivative. A common method is called *finite difference* which approximate the derivative $f'(x)$ by the expression

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

Which spacing h to choose is difficult to answer in general. The smaller it is the closer the estimate to the real value of the derivative. However, the h has to remain finite and if it gets too small, we might run into floating point issues. This is an important example of why it it important to understand the limitations of floating point numbers. The term $f(x+h)$ and $f(x)$ are almost identical if h is small. Thus, calculating the difference will result in a large floating point error.

The expression above is a one sided finite difference. One can also make a central difference:

$$f'(x) \approx \frac{f(x+0.5h) - f(x-0.5h)}{h},$$

which has often better properties. But let's go back to our root finding method.

In Newton's method we need one starting point of x, a guess. Let's call this x_0. During every iteration, we improve upon our guess using the formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

We continue iterating until we have achieved a precision that is good enough for our purpose. Often, people look at the change from x_n to x_{n+1} as a measure of how close we are to the real root.

There are several problems with Newton's method. We already mentioned that the function needs to be differentiable on the entire interval. Furthermore, it turns out that Newton's method might sometimes not converge at all if the first derivate is not *well* behaved. In such a case we might overshoot and never converge. One such example is the function

$$f(x) = |x|^{1/4}.$$

Another problem are stationary points of the function f (i.e. the derivative is zero). In that case we divide by zero and the method breaks down.

One of the most important uses of Newton's method is in optimization. Optimization refers to minimization or maximization of functions. We already know one example, the least square fit.

5.6 Multiple roots

In many cases, functions have multiple roots. For example

$$f(x) = x^2 - 1$$

has roots at $x_0 = 1$ and $x_1 = -1$. How can we find multiple roots? All the above methods give us only one root. Well, the short answer is that in general we might not be able to find all the root of a function. Especially if you think of a function like $\sin(x)$ which has an infinite number of roots.

However, there is a trick that allows us to find at least multiple roots in simple functions. Suppose the function you are trying to find the root is $f(x)$ and you have already found a root x_0. Then, look at the function

$$h(x) = \frac{f(x)}{x - x_0}$$

and find its roots. The figure below shows the function $f(x) = x^2 - 1$ in red. Suppose our first attempt at root finding resulted in the value $x_0 = 1$. Then, the function h is $h(x) = f(x)/(x-1)$, which is plotted in blue. You can see that the function h has now only one root, namely the one we missed before at $x_1 = -1$.

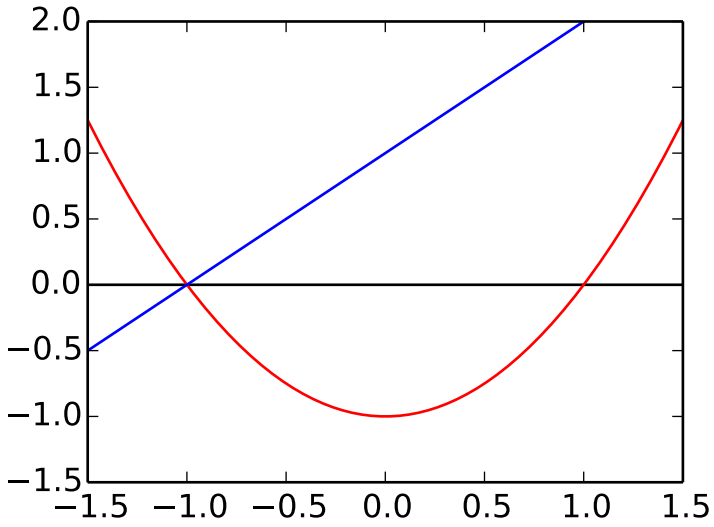

Figure 3: Finding multiple roots.