9.4 Cubic spline interpolation

Instead of going to higher and higher order, there is another way of creating a smooth function that interpolates data-points. A cubic spline is a piecewise continuous curve that passes through all of the values of a given dataset. This works particularly well for smooth datasets with no noise. Each of the piecewise curves is a cubic polynomial with coefficients a_i, b_i, c_i and d_i:

$$S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \qquad \text{for } x \in [x_i, x_{i+1}]$$

If we have N data-points, there are $N - 1$ intervals, hence $(N - 1) \cdot 4$ coefficients that we need to find. Two conditions in each interval arise because we have to match the two data-points at each end.

$$S_i(x_i) = y_i, \quad S_i(x_{i+1}) = y_{i+1}$$

How about the other two parameters? We want to have a smooth function! Thus, we require that the derivatives of the piecewise functions match at the interval boundaries:

$$S'_{i-1}(x_i) = S'_i(x_i), \quad S''_{i+1}(x_i) = S''_i(x_i)$$

We now have almost as many conditions as we have free parameters, except at the boundaries.What could we possibly do there? There are multiple choices and it depends on the problem. We use one called *natural* boundary condition which says that we set the second derivatives to zero at the boundary.

As you can probably guess, this set of equations that we are generating will become a matrix equation. Let's go through the individual steps. Finding the value for the d_is is simple. Our requirement gives us

$$d_i = S_i(x_i) = y_i$$

The condition to match the point at $i + 1$ gives

$$S_i(x_{i+1}) = a_i(x_{i+1} - x_i)^3 + b_i(x_{i+1} - x_i)^2 + c_i(x_{i+1} - x_i) + d_i = y_{i+1}$$

Let's call the derivatives at point i, D_i, i.e.

$$S'_i(x_i) = D_i = c_i$$

and therefore

$$S'_i(x_{i+1}) = D_{i+1} = 3a_i(x_{i+1} - x_i)^2 + 2b_i(x_{i+1} - x_i) + D_i$$

We can now setup an equation system for a, b, c and d:

$$\begin{aligned} a_i &= (D_{i+1} + D_i)(x_{i+1} - x_i)^{-2} - 2(y_{i+1} - y_i)(x_{i+1} - x_i)^{-3} \\ b_i &= (-D_{i+1} - 2D_i)(x_{i+1} - x_i)^{-1} + 3(y_{i+1} - y_i)(x_{i+1} - x_i)^{-2} \\ c_i &= D_i \\ d_i &= y_i \end{aligned}$$

We have one requirement left to play with, to match second derivatives at each interval. This gives us:

$$\begin{aligned} S''_{i-1}(x_i) &= S''_i(x_i) \\ S''_i(x_{x+i}) &= S''_{i+1}(x_{i+1}) \end{aligned}$$

which equates to

$$6a_i(x_{i+1} - x_i) + 2b_i = 2b_{i+1}$$

Let us now combine this with the earlier equation to get

$$\begin{aligned}3(y_i - y_{i-1})(x_i - x_{i-1})^{-1}(x_{i+1} - x_i) + 3(y_{i+1} - y_i)(x_{i+1} - x_i)^{-1}(x_i - x_{i-1}) \\ = D_{i-1}(x_{i+1} - x_i) \\ +D_i(3(x_{i+1} - x_i) + (x_i - x_{i-1})) \\ +D_{i+1}(x_i - x_{i-1})\end{aligned}$$

Let us define

$$Y_i \equiv 3\frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x_{i+1} - x_i) + 3\frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x_i - x_{i-1})$$

We can write this as a matrix equation

$$\begin{pmatrix} \ddots & & & & \\ & (x_{i+1} - x_i) & (3x_{i+1} - 2x_i - x_{i-1}) & (x_i - x_{i-1}) & \\ & & & & \ddots \end{pmatrix} \cdot \begin{pmatrix} \vdots \\ D_i \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ Y_i \\ \vdots \end{pmatrix}$$

At the end points, we have not enough information to fully determine all variables. We therefore come up with new requirements there, let the second derivatives be zero.

$$\begin{aligned} S_0''(x_0) &= 2b_0 = 0 \\ S_{N-2}''(x_{N-1}) &= 6a_{N-2}(x_{N-1} - x_{N-2}) + 2b_{N-2} = 0 \end{aligned}$$

This gives in terms of the D coefficients:

$$\begin{aligned} 2D_0 + D_1 &= 3\frac{y_1 - y_0}{x_1 - x_0} \equiv Y_0 \\ D_{N-2} + 2D_{N-1} &= 3\frac{y_{N-1} - y_{N-2}}{x_{N-1} - x_{N-2}} \equiv Y_{N-1} \end{aligned}$$

We can now complete the matrix from above to

$$\begin{pmatrix} 2 & 1 & 0 & \cdots & & \\ (x_2 - x_1) & (3x_2 - 2x_1 - x_0) & (x_1 - x_0) & 0 & \cdots & \\ 0 & (x_3 - x_2) & (3x_3 - 2x_2 - x_1) & (x_2 - x_1) & 0 & \cdots \\ & & & \ddots & & \\ & & (x_{i+1} - x_i) & (3x_{i+1} - 2x_i - x_{i-1}) & (x_i - x_{i-1}) & \\ & & & \ddots & & \\ & & & & 0 & 1 \quad 2 \end{pmatrix}$$

$$\cdot \begin{pmatrix} D_0 \\ \vdots \\ D_i \\ \vdots \\ D_{N-1} \end{pmatrix} = \begin{pmatrix} Y_0 \\ \vdots \\ Y_i \\ \vdots \\ Y_{N-1} \end{pmatrix}$$

This is a tridiagonal system and can easily be solved. Here, we just use the LU decomposition we already know about. It is easy to find more efficient ways to solve it, but we don't bother.

Once solved for the D_is, we can solve for the a_i, b_i, c_i and d_is. This then defined all parameters for the piecewise cubic function.

In the following figure, we apply this method to the average temperature in Toronto. As you can see, it gives a very smooth and reasonable fit.

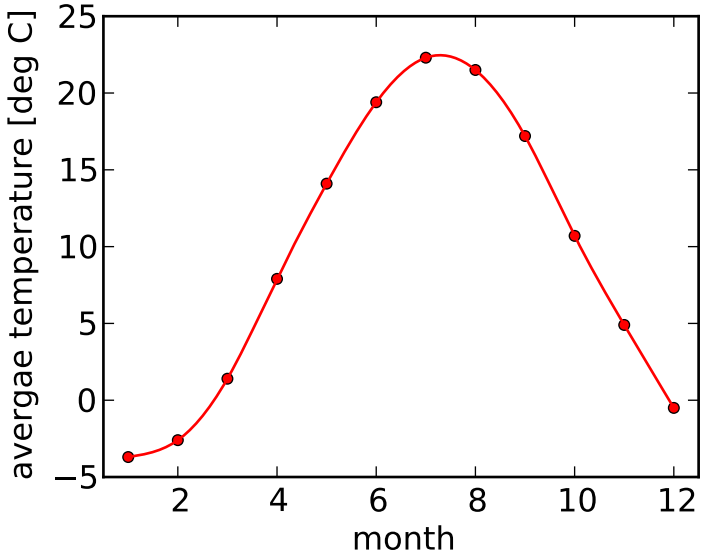

Figure 25: Temperature in Toronto. Cubic spline interpolation.

One way to further improve this spline is to take advantage of the fact that the temperature is periodic. This removes the extra criteria at the boundaries. The matrix will get some extra components (i.e. is not tridiagonal anymore).

Now, we have this great spline interpolation method. Can you use it for any problem? No. Here's a word of caution. Look at the following dataset.

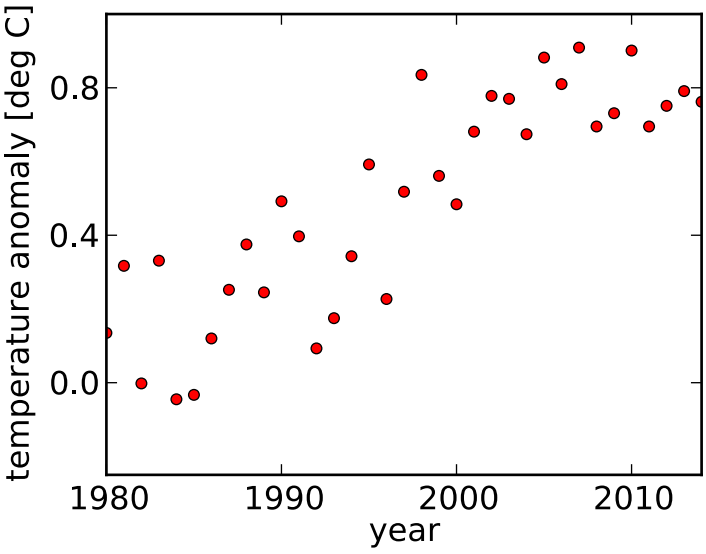

Figure 26: Global temperature anomaly. Source: Climatic Research Unit (University of East Anglia).

The above figure shows the global temperature anomaly. This is an indication of climate change. Clearly, there is a lot of noise in the data. Nevertheless, one can see a very dominant trend towards higher temperatures. We could just fit a spline to this curve. The result is shown in the following figure.

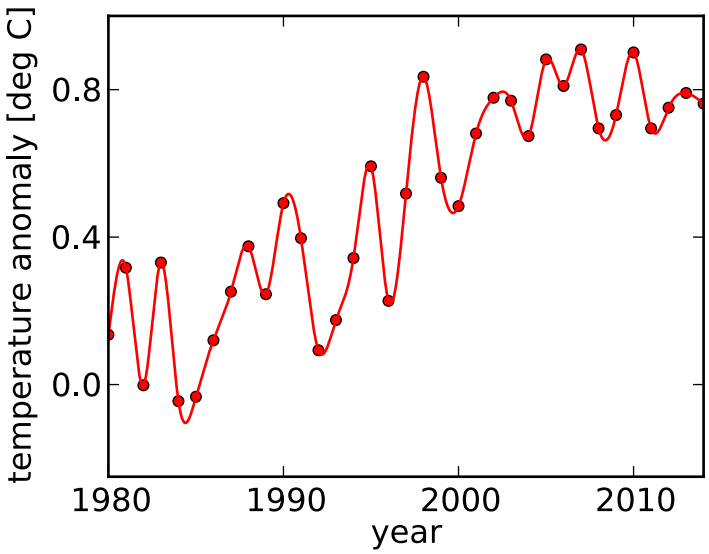

Figure 27: Global temperature anomaly fitted with a cubic spline.

Although the spline is clearly going through all data-points and the curve is smooth at every point in the interval, it is not a good indication of the trend. Why is that? We have fitted a smooth function to noisy data. It's the same issue that we encountered earlier and is sometimes called *ringing* or *Runge's phenomenon*.

Thus, interpolating this data with a constant or piece-wise linear function or even a cubic spline does not make much sense. We would effectively try to interpolate the noise, not the data. So how can we interpolate this data in a meaningful way. We can use a least square fit!

For example, a straight line fit will give us an indiction of how fast the temperature anomaly has risen over time. We can even use it to extrapolate how much temperatures will be rising in the future. The idea of a straight line fit is to find a linear function

$$f(x) = a_1 + a_2 x$$

which is the *best* fit to the data (see our earlier discussion on least square fits). Figure 28 shows the straight line for for the climate data discussed above.

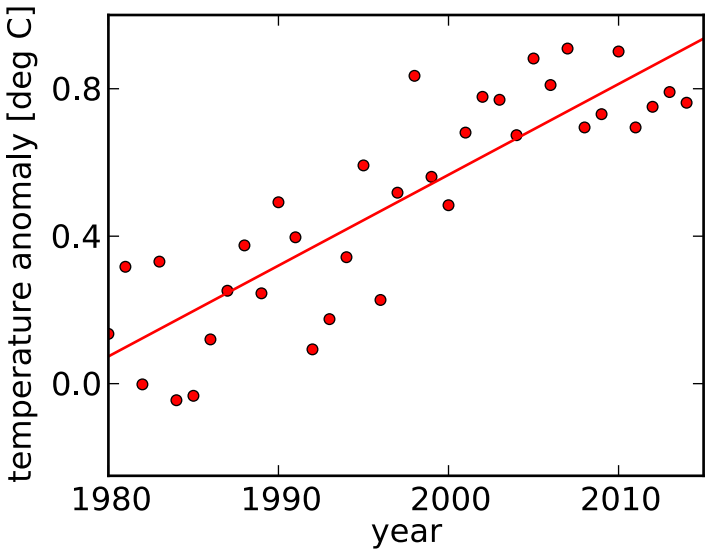

Figure 28: Global temperature anomaly with a straight line fit.

10 Roots finding algorithms

10.1 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

$$f(d) = c.$$

10.2 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 the LU decomposition. 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 $f(x) - g(x) = 0$.

10.3 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.

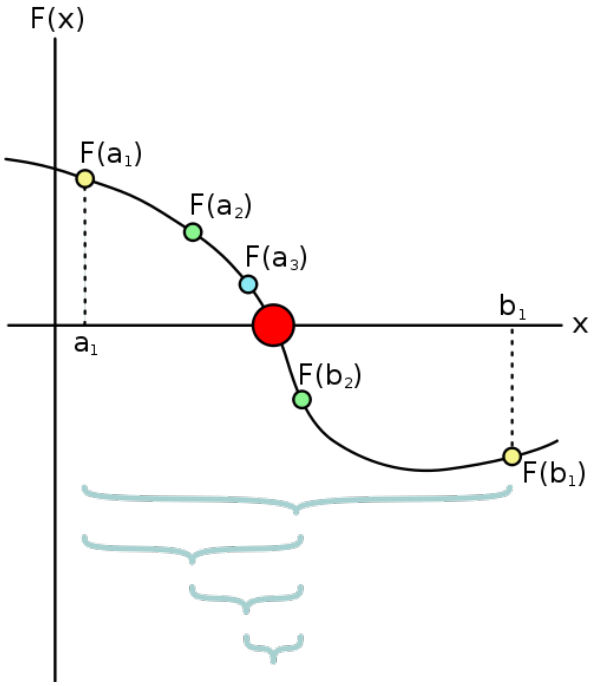

Figure 29: Bisection methods. Source: Wikipedia.

```
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 55: 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.

End of lecture 8.