# Chapter 4

# Interpolation

## 4.1 Introduction

Finding the value of a function at an arbitrary point in a range, given a set of known points that represent it, is another essential piece of a computational physicist's toolkit. This is not a fully-determined problem—the information is not there to completely specify the values between the data points—and therefore additional assumptions are needed. It is up to the physicist to decide what is adequate for the purposes.

One approach is to fit a function to the data points, as is often done with experimental data. This approximate fit will probably not go right through any of the data points, which is often what is needed, and this will be discussed in later chapters.

Another approach, which is considered here, is to make a function that connects the known points. This is interpolation. When might interpolation be needed? One example is in solving ODEs numerically, where time is typically discretised ($t_n$). The value of the solution at a time between these $t_n$ might be required. As we will see later, when solving PDEs, space is discretised into a grid and the numerical method provides the (approximate) solution at these points. Again, the solution at other points in space is often needed, and can only be obtained by interpolation.

Interpolation is an extensive subject. Here we briefly look at a few of the simplest methods. These simple methods will get you surprisingly far, and can still be found in most numerical software suites today. More advanced and powerful methods certainly exist too. One of our favourites is splines under tension,[1] where one uses an additional parameter to interpolate the interpolation scheme between linear and cubic splines. Neural networks and other machine learning algorithms are, at the end of the day, also just fancy multi-dimensional interpolators. A machine-learning algorithm uses functions trained on some known data points to produce a good guess at a quantity somewhere between the data points on which it has been trained.

As with any other numerical method, it is up to the physicist to show that the chosen algorithm is appropriate for their purposes, as we shall see.

## 4.2 Linear interpolation

Linear interpolation is simply to connect adjacent points with a straight line, connect-the-dots style. If two, adjacent, known points are $(x_i, f_i)$ and $(x_{i+1}, f_{i+1})$, to find $f$ at a point $x$ in between we simply move along the straight line between the two points:

$$\boxed{f(x) = \frac{(x_{i+1} - x)f_i + (x - x_i)f_{i+1}}{x_{i+1} - x_i}.} \tag{4.1}$$

This is the simplest form of interpolation, and is useful if the density of the provided data points is high, or if you somehow know that the function is linear in the regions between the points. Its

[1] See e.g. TSPACK, http://dl.acm.org/citation.cfm?id=151277

behaviour is certainly more predictable than other, more involved forms of interpolation. However, in the more general case, it is unlikely that a true function looks like anything that is made up of only a succession of straight segments, and the limitations of linear interpolation will be quite clear.

## 4.3 Bi-linear interpolation

Bi-linear interpolation works for a function of two variables $f(x, y)$. Actually it is just linear interpolation in two dimensions. You can probably work it out for yourself already. Imagine $f$ is known on four points which are the corners of a rectangle in the $x$–$y$ coordinate system; $(x_i, y_k)$, $(x_{i+1}, y_k)$, $(x_i, y_{k+1})$ and $(x_{i+1}, y_{k+1})$. We want $f$ at an arbitrary point $(x, y)$ within this square. Linear interpolation is first applied to $f$ in the $x$ direction, along both the bottom ($y = y_k$) and top ($y = y_{k+1}$) of the square. For instance, along the bottom, equation (4.1) is used with $(x_i, f(x_i, y_k))$ and $(x_{i+1}, f(x_{i+1}, y_k))$ to obtain the intermediate value $f(x, y_k)$. Similarly at the top linear interpolation yields $f(x, y_{k+1})$. Now these two intermediate values are linearly interpolated in the $y$-direction, using an equation analogous to (4.1). Doing interpolation in $y$ to get the intermediate values and then interpolation in $x$ yields exactly the same value.

Of course you can do this in as many dimensions as you like; multivariate interpolation is the generalisation to functions with more than one variable; $f(x, y, z, \ldots)$. In this case, you need to use a (hyper-)box around your desired position, with $2^D$ vertices, where $D$ is the dimension of your problem.

## 4.4 Lagrange polynomials

Starting with $n + 1$ distinct points $(x_i, f_i)$ where $0 \leq i \leq n$ and $x_i < x_j$, it is clear that there is a unique $n^{\text{th}}$ degree polynomial (i.e. with $n + 1$ degrees of freedom) that goes exactly through these points. This is called the Lagrange polynomial for these points, and can be written:

$$\boxed{P_n(x) = \sum_{i=0}^{n} \left( \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \right) f_i,} \tag{4.2}$$

where in the product, $j$ run over integers from 0 to $n$ but misses out the value $i$. For example, for the $n = 2$ case

$$P_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f_0 + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f_1 + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f_2. \tag{4.3}$$

Note that $x_i$ do not need to be equally spaced.

Such a polynomial does meet the requirements of being a smooth, well-behaved function that passes through all the data points, but unless it is known that the data follow a polynomial form of the correct degree (which is unlikely), the Lagrange polynomial is likely to "contort" itself to make it go through the data points, resulting in unnaturally wavy behaviour, making it a poor interpolant.

For specific cases, when the data points and the physics problem at hand suit it, the Lagrange polynomial can be extremely useful.

## 4.5 Cubic splines

Linear interpolation is nice, because it is easy and fast, and does not display the odd behaviour that the Lagrange polynomial can if you are not careful. But it is boring—the resulting approximation to the underlying function is clearly not a very good representation of reality in most cases, as it has zero second derivative and a discontinuous first derivative at every data point. No self-respecting function has discontinuous derivatives, especially not one that purports to represent a physical

quantity. The data points given to us are usually not accompanied by the statement "these points are also where the first derivatives can be discontinuous".

What can be done about this? We can introduce non-zero higher derivatives to our interpolating function, and force them to be continuous across the data-points, but to do that we need to use polynomials of higher order than one. The lowest-order option is to make all our interpolating functions quadratics. This lets us have continuous first derivatives, and non-zero second derivatives—but the second derivatives will still be discontinuous across the data points. The lowest order polynomial interpolant that allows for continuous first and second derivatives is cubic.

As a general term, a function that matches polynomials, in piecewise fashion along the data points, is called a "spline".

However, once we start introducing additional terms in the interpolating functions, we will have more free parameters to constrain for each piece of the approximating function, so we are going to have to start using more data points at a time to fit those parameters. We therefore need to start making the interpolant somewhat non-local, such that it is not just fit to the two data points either side of it, but more points, further away in each direction. The more continuous derivatives you want, the higher-order your interpolant needs to be, and the more data points you must use to constrain it.

For the cubic spline, we need to use four points at a time; one at each end of the interval being interpolated, and one more each on either side of the interval. We can start by taking the expression for the linear interpolant:

$$f(x) = A(x)f_i + B(x)f_{i+1}, \tag{4.4}$$

with

$$A(x) \equiv \frac{x_{i+1} - x}{x_{i+1} - x_i}, \tag{4.5}$$

$$B(x) \equiv 1 - A(x) = \frac{x - x_i}{x_{i+1} - x_i} \tag{4.6}$$

and supplementing it with some additional corrections, proportional to the second derivatives at each of the adjacent points:

$$f(x) = A(x)f_i + B(x)f_{i+1} + C(x)f_i'' + D(x)f_{i+1}'', \tag{4.7}$$

with

$$C(x) \equiv \tfrac{1}{6}(A(x)^3 - A(x))(x_{i+1} - x_i)^2 \tag{4.8}$$

$$D(x) \equiv \tfrac{1}{6}(B(x)^3 - B(x))(x_{i+1} - x_i)^2 \tag{4.9}$$

At this point, Eq. 4.7 seems pretty useless—how do we know the values of the second derivative of the function at the data points? We only have data on the value of the function itself, not its derivatives. The key here is to solve for these second derivatives at the points $\{x_0, x_2, x_3, \ldots, x_n\}$.

But how? First we need to impose continuity of the *first* derivative across each data point. Using

$$\frac{\mathrm{d}A}{\mathrm{d}x} = -\frac{\mathrm{d}B}{\mathrm{d}x} = -\frac{1}{x_{i+1} - x_i}, \tag{4.10}$$

$$\frac{\mathrm{d}C}{\mathrm{d}x} = \frac{1 - 3A^2}{6}(x_{i+1} - x_i) \tag{4.11}$$

$$\frac{\mathrm{d}D}{\mathrm{d}x} = \frac{3B^2 - 1}{6}(x_{i+1} - x_i) \tag{4.12}$$

we get

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} - \frac{3A^2 - 1}{6}(x_{i+1} - x_i)f_i'' + \frac{3B^2 - 1}{6}(x_{i+1} - x_i)f_{i+1}''. \tag{4.13}$$

Imposing the continuity of $\frac{\mathrm{d}f}{\mathrm{d}x}$ at $x_i$ (i.e. where intervals $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$ meet)

$$\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x\to x_i^-} = \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x\to x_i^+} \tag{4.14}$$

then gives the fundamental expression

$$\frac{x_i - x_{i-1}}{6} f''_{i-1} + \frac{x_{i+1} - x_{i-1}}{3} f''_i + \frac{x_{i+1} - x_i}{6} f''_{i+1} = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} - \frac{f_i - f_{i-1}}{x_i - x_{i-1}}. \tag{4.15}$$

This equation is valid for $i = 1\,..\,n{-}1$. This means that it constitutes a system of $n-1$ equations in $n+1$ unknowns ($f''_{0..n}$). We know from linear algebra that this means it has a two-dimensional solution space, implying that we need two more constraints to solve the system. Those constraints come from the chosen **boundary conditions**, which are up to the user of the algorithm to choose as they please. These may take the form of a specified value of the first or second derivative at each of the two end-points of the region being interpolated. The choice $f''_0 = f''_n = 0$ has a special name: the 'natural' spline. This is because if there is a thin massless and frictionless beam that is fixed such that it passes through each data point, it will take on the form that is given by the natural spline. One can also choose to use the two degrees of freedom to fix the first derivatives at the end points to pre-determined values.

The simultaneous equations for $f''_{0..n}$ that are given by this can be set up as a matrix equation, and the matrix be inverted to solve for all the second derivative values. Once you have these, you can use Eq. 4.7 to evaluate your cubic spline at any and as many values of $x$ as you want, at your leisure—without ever having to re-evaluate the second derivatives.

With any interpolation scheme, it is important that validity of the scheme for your set of data is checked, on a case-by-case basis. With that caveat, the cubic spline is a excellent algorithm for general use, with continuity and smoothness of the interpolant being guaranteed with a small number of free parameters, which is often what one needs in a physics context.

Extensions of the cubic spline into multiple dimensions, as well as more sophisticated algorithms do exist, and are implemented in many external library packages. However, from a physicist's point of view, the decision whether or not to interpolate, or to rather fit a function to the data, is often the most important—and we will look at function fitting in a later lecture.