MATH3511/6111: Scientific Computing

04. Spline Approximation

Lindon Roberts

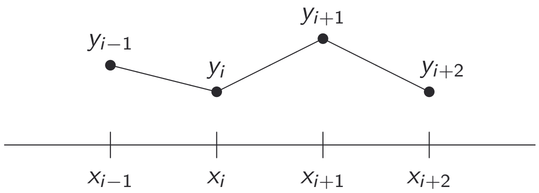
Semester 1, 2022

Office: Hanna Neumann Building #145, Room 4.87
Email: lindon.roberts@anu.edu.au
Based on lecture notes written by S. Roberts, L. Stals, Q. Jin, M. Hegland, K. Duru.

Approximation of Functions

In this section we consider a different approximation/interpolation method, using piecewise polynomials.

The simplest idea is to use continuous piecewise linear functions. In fact, this is the most common way to produce a plot of a function:



Piecewise Linear Interpolation

For example, if we do continuous piecewise linear interpolation of $f(x) = \sin x$ we get

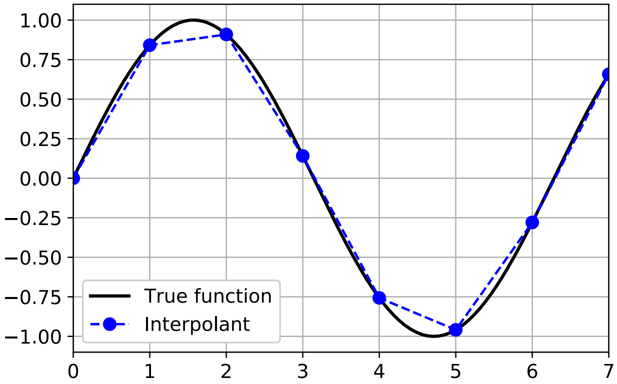


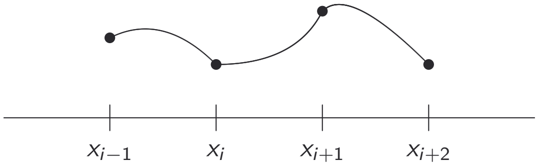
Figure 1. Linear Interpolant of $f(x) = \sin x$ for $x \in [0, 7]$.

Splines

Piecewise linear approximations are easy to construct, but they are not smooth. non-differentiable

Often we want our approximations to be smooth (especially the function we are approximating is also smooth).

To get smoothness, we use (continuous) piecewise polynomials of higher degree. Approximations by continuous piecewise polynomials are called splines.



We measure the smoothness of a spline by how many derivatives are continuous at the nodes (also called ‘knots’ when working with splines).

Cubic Splines

The most common spline is the cubic spline, which is a piecewise cubic interpolant:

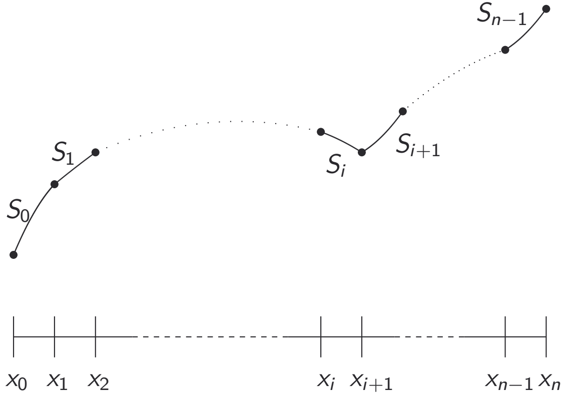
$$S(x) = \begin{cases} S_0(x), & x_0 \le x < x_1, \\ S_1(x), & x_1 \le x < x_2, \\ \vdots & \\ S_{n-1}(x), & x_{n-1} \le x \le x_n, \end{cases}$$

where each $S_i(x)$ is a cubic, and we satisfy the interpolation conditions

$$S_i(x_i) = y_i \quad \text{and} \quad S_i(x_{i+1}) = y_{i+1}, \qquad \text{for all } i = 0, \ldots, n-1.$$

2n conditions

Cubic Splines



Cubic Splines

This hasn't helped — we still have a continuous piecewise approximation that is not differentiable!

However, because we have piecewise cubics, we have more flexibility to impose more conditions on our spline.

4n unknowns

In particular, we ask that the spline has continuous first and second derivatives at each (interior) node:

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \quad \text{and} \quad S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}), \qquad \text{for } i = 0, \ldots, n-2.$$

$2n-2$ conditions added $\longrightarrow$ 2 remain

Cubic splines are implemented in SciPy: `scipy.interpolate.CubicSpline`

Cubic Splines

Our spline is comprised of n cubics, so there are $4n$ unknowns to solver for.

So far, we have stated $2n$ interpolation conditions and $2n-2$ derivative conditions. We need to impose two more conditions to have the same number of equations as unknowns.

There are three common choices, all related to derivatives of the spline near the edge nodes:

- Clamped spline: most accurate

$$S_0'(x_0) = f'(x_0) \quad \text{and} \quad S_{n-1}'(x_n) = f'(x_n),$$

need extra data

- Natural/free spline:

$$S_0''(x_0) = S_{n-1}''(x_n) = 0$$

- 'Not a knot' condition: $S'''(x)$ is continuous at x_1 and x_{n-1}.

In general, clamped splines are the best choice, but we can only construct them if we know $f'(x_0)$ and $f'(x_n)$. All three choices (and more) are available in SciPy.

Cubic Spline: Example

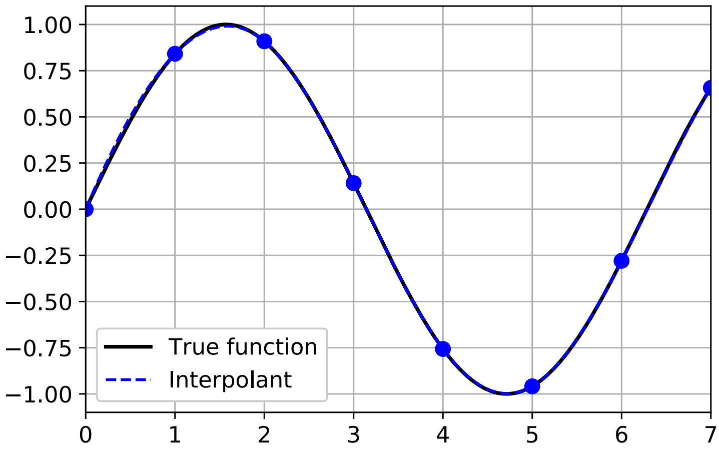


Figure 2. 'Not a knot' cubic spline interpolant of $f(x) = \sin x$ for $x \in [0, 7]$.

Why Cubic Splines?

Question

Why are cubic splines the most common choice? Why not quadratic or quartic?

Theorem

If $f(x)$ is twice continuously differentiable, then the clamped cubic spline $S(x)$ gives the best piecewise linear approximation $S''(x)$ (with nodes at x_i) to the true second derivative $f''(x)$.

Also, the error of cubic spline interpolation decreases very rapidly as we reduce the spacing between the nodes (but don't need to cluster them at the endpoints like Chebyshev points).

Theorem

If $f(x)$ is 4 times continuously differentiable with $|f^{(4)}(x)| \leq M$ for all $x \in [x_0, x_n]$, then the clamped cubic spline $S(x)$ satisfies the error bound

take the biggest gap

$$|f(x) - S(x)| \leq \frac{5M}{384} \underline{\max_{0 \leq i \leq n-1}} (x_{i+1} - x_i)^4, \qquad \text{for all } x \in [x_0, x_n].$$

best: equally spaced

Hermite Interpolation

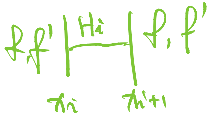
Using derivative information

For rootfinding, we saw that Newton's method converges faster than the secant method because it uses extra information (first derivatives). Can we do this for splines?

Interpolation using both $f(x)$ and $f'(x)$ is known as Hermite interpolation (Charles Hermite, 1864, but known to Pierre-Simon Laplace, 1810).

For piecewise cubic Hermite interpolation, we get 4 conditions for each cubic section: it has to match both $y_i = f(x_i)$ and $y_i' = f'(x_i)$ at both endpoints

$$H_i(x_i) = y_i, \quad \text{and} \quad H_i(x_{i+1}) = y_{i+1},$$
$$H_i'(x_i) = y_i' \quad \text{and} \quad H_i'(x_{i+1}) = y_{i+1}',$$



for $i = 0, \ldots, n-1$.

Cubic Hermite splines are implemented in SciPy: `scipy.interpolate.CubicHermiteSpline`

Hermite Interpolation

To find $H_i(x)$, it will be easier to write it as

$$H_i(x) = a_i + b_i(x - x_i) + (x - x_i)^2[c_i + d_i(x - x_{i+1})], \qquad \text{for } x_i \le x < x_{i+1}.$$

Defining $h_i = x_{i+1} - x_i$, we get

$$\begin{aligned} H_i(x_i) &= a_i, \qquad H_i'(x_i) = b_i, \\ H_i(x_{i+1}) &= a_i + b_i h_i + c_i h_i^2, \\ H_i'(x_{i+1}) &= b_i + 2c_i h_i + d_i h_i^2, \end{aligned}$$

and so applying the interpolation conditions we get

$$\begin{aligned} a_i &= y_i, \qquad b_i = y_i', \\ c_i &= \frac{y_{i+1} - y_i}{h_i^2} - \frac{y_i'}{h_i}, \\ d_i &= \frac{y_{i+1}' - y_i'}{h_i^2} - \frac{2(y_{i+1} - y_i)}{h_i^3}. \end{aligned}$$

Splines or Polynomials?

Question

When to use splines or polynomials?

It depends on what you need the interpolation for!

- Splines give good approximations for any choice of nodes, polynomial interpolation needs good spacing (e.g. Chebyshev points)
- Smooth polynomials are sometimes easier to manipulate (e.g. find maxima/minima), but sometimes splines are easy too (e.g. integration)