9. Interpolation

Outline

- Definition
- Types of interpolation
    - Polynomial
    - Piecewise polynomial
        - Linear spline
        - Cubic spline
        - Shape-preserving cubic

Definition of interpolation

- Given a set of n points $\{x_i, y_i\}, i = 1, 2, \ldots n$, find a function f such that for all i, $f(x_i) = y_i$.
- We'll assume that the points are arranged so that the x_i are in ascending order, $x_1 < x_2 < x_3 < \ldots$
- Many different types of functions can be fit to any given set of points. We can choose a type that makes sense for a particular problem, or try out a few

A graphic example

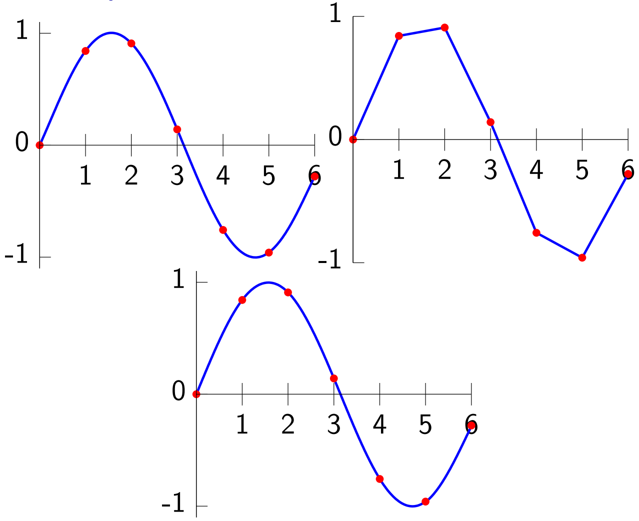

Figure: Different interpolating functions fit to one set of points: polynomial, linear spline, cubic spline

Example applications

- Estimate quantities at different locations, temperatures, etc. given a few experimental or computed data (table lookup)
- Estimate derivatives or integrals given values at a few points (by fitting an interpolating function to those points that can be readily differentiated and integrated)

Polynomial interpolation

- For any n different x values and any associated y values, $\{x_i, y_i\}$, there's a unique polynomial of (up to) degree $n-1$ that interpolates them
- Advantage is that polynomials are easy to work with
- We'll look at 3 methods of finding this polynomial, which give it in different forms: 1) Solving a linear system; 2) the Lagrange form; 3) the Newton form

1. Solving a linear system

The equations

$$\begin{gathered}
x_1^{n-1} a_{n-1} + x_1^{n-2} a_{n-2} + \ldots + x_1 a_1 + a_0 = y_1 \\
x_2^{n-1} a_{n-1} + x_2^{n-2} a_{n-2} + \ldots + x_2 a_1 + a_0 = y_2 \\
\ldots \\
x_n^{n-1} a_{n-1} + x_n^{n-2} a_{n-2} + \ldots + x_n a_1 + a_0 = y_n
\end{gathered}$$

form a linearly independent set for the unknown coefficients a_i, so can be solved using Gauss elimination to find the coefficients. The coefficient matrix for this system is called a *Vandermonde matrix*.

Example

For sample points $\begin{array}{c|rrrrr} x & -2 & -1 & 0 & 1 & 3 \\ y & 9 & -1 & -3 & -3 & -5, \end{array}$

the linear system is
$$\begin{array}{rrrrr|r} 16 & -8 & 4 & -2 & 1 & 9 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & -3 \\ 1 & 1 & 1 & 1 & 1 & -3 \\ 81 & 27 & 9 & 3 & 1 & -5, \end{array}$$
which can be solved for the coefficients a_i to give the interpolating polynomial $p(x) = \frac{2}{15}x^4 - \frac{11}{15}x^3 + \frac{13}{15}x^2 - \frac{4}{15}x - 3$.

2. Lagrange form

Add up n polynomials of degree $n-1$, each of which has the right value at one point and is zero at the other points

Each such polynomial is of the form

$$y_i \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}$$

The sum is therefore $p(x) = \sum_{i=1}^{n} y_i \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}$

For the example points, this looks like

$$\begin{aligned} p(x) = \frac{3}{10}(x+1)x(x-1)(x-3) + \frac{1}{8}(x+2)x(x-1)(x-3) \\ -\frac{1}{2}(x+2)(x+1)(x-1)(x-3) + \frac{1}{4}(x+2)(x+1)x(x-3) \\ -\frac{1}{24}(x+2)(x+1)x(x-1) \end{aligned}$$

3. Newton form

Add up one polynomial each of degree $0, 1, 2, \ldots n-1$, so that sum of the first i polynomials gives the right value at the first i points
The result is therefore P_n, where the intermediate partial sums are $P_1 = y_1$ and

$$P_i = P_{i-1} + (y_i - P_{i-1}(x_i)) \prod_{j=1}^{i-1} \frac{x - x_j}{x_i - x_j}$$

for $i = 2, \ldots n$
For the example points, this looks like

$$\begin{aligned} p(x) = P_n = 9 &- 10(x+2) + 4(x+2)(x+1) \\ &- (x+2)(x+1)x + \frac{2}{15}(x+2)(x+1)x(x-1) \end{aligned}$$

Comparing the interpolating polynomial computation methods

- Solving a linear system gives directly the coefficients of the polynomial in standard form. However, it requires more computation ($\mathcal{O}(n^3)$ arithmetic operations) compared to the other methods, and is prone to roundoff error because the condition number of Vandermonde matrices is often large.
- Finding the Lagrange or Newton forms requires $\mathcal{O}(n^2)$ operations and is less vulnerable to roundoff error
- The most important thing to know about these methods is that they all result in the same polynomial, even if written differently. For our example, $p(2) = -3.8$ for all of them.

Drawbacks of polynomial interpolation

- Especially for larger *n*, the whole interpolating polynomial may change greatly if the value at one point is only changed slightly (an example of ill conditioning). Also, the polynomial will often have large oscillations even if the *y* values of the points being interpolated are all within a small range
- Further, if more and more points are sampled within a given interval from a known function, interpolating increasingly higher-degree polynomials based on these points doesn't necessarily converge to that function (Runge's phenomenon)
- Therefore, polynomial interpolation is generally not recommended for many points ($n > 5$ or so)

Piecewise polynomials

- Interpolating function is made up of segments, each of which is a polynomial of some given (lower) degree
- Less smooth than polynomials, but avoid the drawbacks given above
    - Generally *local* and *well-conditioned* – changing one point will change the interpolating function primarily close to that point, and a slight change in the data will not greatly change the function
    - Computation is faster when the number of points is large, and not as subject to roundoff
    - Does converge to the generating function over an interval if more and more equally spaced points from it are added

Linear spline

- Piecewise linear function, where we draw straight lines connecting each adjacent pair of points
- General formula for the *i*th piece (with i going from 1 to $n-1$) is
  $s_i(x) = y_i + \frac{y_{i+1}-y_i}{x_{i+1}-x_i}(x - x_i)$ for $x_i \leq x \leq x_{i+1}$
- This linear spline is continuous, but doesn't have a continuous first derivative

Linear spline example

For the given points, the interpolating linear spline can be written as

$$S(x) = \begin{cases} -11 - 10x & \text{if } -2 \le x \le -1 \\ -3 - 2x, & \text{if } -1 \le x \le 0 \\ -3, & \text{if } 0 \le x \le 1 \\ -2 - x, & \text{if } 1 \le x \le 3. \end{cases}$$

For example, $S(2)$ would be -4.

Cubic spline

- Piecewise cubic function, where we draw cubic polynomials connecting each adjacent pair of points
- The cubic polynomials are chosen such that the first and second derivatives of the combined function are continuous, so it looks smoother than the linear spline

Finding cubic splines for given points

- The cubic spline has $n-1$ pieces, each of which is a cubic polynomial, so looks like $s_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i$ for $x_i \leq x \leq x_{i+1}$
- The interpolation and continuity conditions impose that
    - $s_i(x_i) = y_i \rightarrow d_i = y_i$
    - $s_i(x_{i+1}) = y_{i+1} \rightarrow a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i = y_{i+1}$
    - $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \rightarrow 3a_i h_i^2 + 2b_i h_i + c_i = c_{i+1}$
    - $s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}) \rightarrow 6a_i h_i + 2b_i = 2b_{i+1}$

  where $h_i \equiv x_{i+1} - x_i$
- These conditions give us $4n-6$ linear equations in the $4n-4$ unknown coefficients $\{a_i, b_i, c_i, d_i\}$. We need 2 more conditions to get a unique solution. Different choices are possible, but a common one is "natural boundary conditions", where the second derivatives at the first and last points are set to zero:
    - $s_1''(x_1) = 0 \rightarrow 2b_1 = 0$
    - $s_{n-1}''(x_n) = 0 \rightarrow 6a_{n-1}h_{n-1} + 2b_{n-1} = 0$

Linear system for the cubic spline coefficients

- We can simplify the system of equations algebraically by solving for the *a* and *c* coefficients in terms of *b*. The result is this tridiagonal system for the unknowns $\{b_i | i = 1, 2, \dots n\}$:

$$\left(\begin{array}{ccccccccc|c}
1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \\
h_1 & 2(h_1+h_2) & h_2 & 0 & \dots & 0 & 0 & 0 & 0 & 3(\Delta_2-\Delta_1) \\
0 & h_2 & 2(h_2+h_3) & h_3 & \dots & 0 & 0 & 0 & 0 & 3(\Delta_3-\Delta_2) \\
\dots & & & & & & & & & \\
0 & 0 & 0 & 0 & \dots & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} & 0 & 3(\Delta_{n-2}-\Delta_{n-3}) \\
0 & 0 & 0 & 0 & \dots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 3(\Delta_{n-1}-\Delta_{n-2}) \\
0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 & 0
\end{array}\right)$$

  Here Δ_i is short for $\frac{y_{i+1}-y_i}{h_i}$
  The first and last equations are for natural boundary conditions
  We include an extra coefficient b_n, which is not one of the cubic coefficients but stands for the $\frac{s''_{n-1}(x_n)}{2}$
- Tridiagonal $\rightarrow$ only $\mathcal{O}(n)$ operations to find coefficients
- (The lecture notes for cubic spline use a different indexing, where i for the points x_i goes from 0 to n instead of 1 to n)

Finding the remaining coefficients

- Once the b_i coefficients have been found by solving this linear system, the remaining coefficients can be found as
  - $a_i = \frac{b_{i+1} - b_i}{3h_i}$
  - $c_i = \Delta_i - h_i b_i - h_i^2 a_i$
  - $d_i = y_i$

Results for example: natural boundary conditions

The h values are $1, 1, 1, 2$

The Δ values are $-10, -2, 0, -1$

The linear system for the b values is

$$\left[\begin{array}{ccccc|c}
1 & 0 & 0 & 0 & 0 & 0 \\
1 & 4 & 1 & 0 & 0 & 24 \\
0 & 1 & 4 & 1 & 0 & 6 \\
0 & 0 & 1 & 6 & 2 & -3 \\
0 & 0 & 0 & 0 & 1 & 0
\end{array}\right]$$

which has the solution $\mathbf{b} = (0, 5.965, 0.140, -0.523, 0)^T$, based on which $\mathbf{a} = (1.988, -1.942, -0.221, 0.087)^T$ and $\mathbf{c} = (-11.988, -6.023, 0.081, -0.302)^T$

The interpolating cubic spline is thus

$$S(x) = \begin{cases} 1.988(x+2)^3 - 11.988(x+2) + 9 & \text{if } -2 \le x \le -1 \\ -1.942(x+1)^3 + 5.965(x+1)^2 - 6.023(x+1) - 1 & \text{if } -1 \le x \le 0 \\ -0.221x^3 + 0.140x^2 + 0.081x - 3 & \text{if } 0 \le x \le 1 \\ 0.087(x-1)^3 - 0.523(x-1)^2 - 0.302(x-1) - 3 & \text{if } 1 \le x \le 3 \end{cases}$$

$S(2)$ is $0.087 - 0.523 - 0.302 - 3 = -3.738$

Shape-preserving cubic

- Also piecewise cubic function, but pieces are chosen so that function is monotonic wherever the given points are
  - Like linear spline, function has no more maxima/minima than what's necessary to match points, and does not go outside range of data
  - Smoother than linear spline, but less smooth than cubic spline ($S'(x)$ is continuous but $S''(x)$ isn't)
- We won't cover how to find the coefficients, but can use `scipy.interpolate.PchipInterpolator` in Python

Results for example

`PchipInterpolator` returns

$$S(x) = \begin{cases} \frac{7}{3}(x+2)^3 + \frac{4}{3}(x+2)^2 - 14(x+2) + 9 & \text{if } -2 \le x \le -1 \\ \frac{2}{3}(x+1)^3 + \frac{2}{3}(x+1)^2 - \frac{10}{3}(x+1) - 1 & \text{if } -1 \le x \le 0 \\ -3 & \text{if } 0 \le x \le 1 \\ \frac{1}{12}(x-1)^3 - \frac{2}{3}(x-1)^2 - 3 & \text{if } 1 \le x \le 3 \end{cases}$$

$S(2)$ is $\frac{1}{12} - \frac{2}{3} - 3 = -3.583$

Python functions

- Polynomials
  - `numpy.polyfit` (solves linear system) returns vector of coefficients
  - `numpy.polyval` evaluates polynomial at given points
- Piecewise polynomials
  - `scipy.interpolate.interp1d` (with options `‘linear’` (default), `‘cubic’` (spline), `‘pchip’`, ...) returns a piecewise polynomial function
  - See the `scipy.interpolate` documentation for different options

Cubic spline boundary conditions

- By default, `scipy.interpolate.CubicSpline` in Python returns the interpolating cubic spline with so-called *not-a-knot* conditions, which are that (1) the first and second pieces are the same cubic polynomial, (2) the second-to-last and last pieces are the same cubic polynomial.(`bc_type = natural` can be used for natural boundary conditions.)
- We can find the b_i in that case
  by replacing the first and last equations in our linear system with

$$\begin{array}{cccccccccc|c} h_2 & -(h_1+h_2) & h_1 & 0 & \dots & 0 & 0 & 0 & 0 & & 0 \\ \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & h_{n-1} & -(h_{n-2}+h_{n-1}) & h_{n-2} & & 0 \end{array}$$

- For our example, these equations are

$$\begin{array}{ccccc|c} 1 & -2 & 1 & 0 & 0 & 0 \\ \dots \\ 0 & 0 & 2 & -3 & 1 & 0 \end{array}$$

and the resulting function is

$$S(x) = \begin{cases} -1.157(x+2)^3 + 7.471(x+2)^2 - 16.314(x+2) + 9, & -2 \le x \le 0 \\ -0.216x^3 + 0.529x^2 - 0.314x - 3, & 0 \le x \le 3 \end{cases}$$

with $S(2) = -3.235$