Interpolation
Interpolation

- As we've seen, we frequent have data only at a discrete number of points
  - Interpolation fills in the gaps by making an assumption about the behavior of the functional form of the data

- Many different types of interpolation exist
  - Some ensure no new extrema are introduced
  - Some match derivatives at end points
  - ...

- Generally speaking: you need to balance the number of points used (which can increase accuracy) against pathologies (like oscillations)
  - You may want to enforce some other property on the form of the interpolant

- We'll follow the discussion from Pang
Interpolation vs Fitting

- *Interpolation* seeks to fill in missing information in some small region of the whole dataset.

- *Fitting* a function to the data seeks to produce a model (guided by physical intuition) so you can learn more about the global behavior of your data.
Linear Interpolation

- Simplest idea—draw a line between two points

\[ f(x) = \frac{f_2 - f_1}{x_2 - x_1}(x - x_1) + f_1 \]

- Exactly recovers the function values at the end points
Linear Interpolation

- Actual \( f(x) \) is given by

\[
f(x) = f_i + \frac{x - x_i}{\Delta x} (f_{i+1} - f_i) + \Delta f(x)
\]

(Linear interpolant)

- Want to know the error at a point \( x = a \) in \([x_i, x_{i+1}]\)
- We can fit a quadratic to \( x_i, a, x_{i+1} \)
- Error can be shown (through lots of algebra...) to be

\[
\Delta f(x) = \left. \frac{f'''(x)}{2} (x - x_i)(x - x_{i+1}) \right|_{x=a}
\]

(at that point)

- This means error in linear interpolation \( \sim O(\Delta x^2) \)
Linear Interpolation

- Error estimate graphically
Quadratic Interpolation

- Fit a parabola—requires 3 points
  - Already saw this with Simpson's rule
- Note: can fall out of the range of $f_1$, $f_2$, or $f_3$
  - Over/undershoots can be problematic depending on the application
Example: Mass Fractions

- Higher-order is not always better.
- Practical example
  - In hydrodynamics codes, you often carry around mass fractions, $X_k$ with

$$
\sum_k X_k = 1
$$

- If you have these defined at two points: $a$ and $b$ and need them in-between, then:

$$
X_k(x) = (X_k)_a + \frac{(X_k)_b - (X_k)_a}{\Delta x}(x - a)
$$

sums to 1 for all $x$
- Higher-order interpolation can violate this constraint
Lagrange Interpolation

- General method for building a single polynomial that goes through all the points (alternate formulations exist)

- Given n points: \( x_0, x_1, \ldots, x_{n-1} \), with associated function values: \( f_0, f_1, \ldots, f_{n-1} \)
  - construct basis functions:
    \[
    l_i(x) = \prod_{j=0, i \neq j}^{n-1} \frac{x - x_j}{x_i - x_j}
    \]
    - Basis function \( l_i \) is 0 at all \( x_j \) except for \( x_i \) (where it is 1)
  - Function value at \( x \) is:
    \[
    f(x) = \sum_{i=0}^{n-1} l_i f_i
    \]
Lagrange Interpolation

- Consider a quadratic
  - Three points: \((x_0, f_0), (x_1, f_1), (x_2, f_2)\)
  - Three basis functions:

\[
\begin{align*}
l_0 &= \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_2} = \frac{(x - x_1)(x - x_2)}{2\Delta x^2} \\
l_1 &= \frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} = -\frac{(x - x_0)(x - x_2)}{\Delta x^2} \\
l_2 &= \frac{x - x_0}{x_2 - x_0} \cdot \frac{x - x_1}{x_2 - x_1} = \frac{(x - x_0)(x - x_1)}{2\Delta x^2}
\end{align*}
\]
Lagrange Interpolation

- Quadratic Lagrange polynomial:

\[ f(x) = \frac{(x - x_1)(x - x_2)}{2\Delta x^2} f_0 - \frac{(x - x_0)(x - x_2)}{\Delta x^2} f_1 + \frac{(x - x_0)(x - x_1)}{2\Delta x^2} f_2 \]

Let's look at the code
Lagrange Interpolation

- Form is easy to remember
- Not the most efficient form to compute the polynomial
  - All other forms for n-degree polynomials that pass through the specified n+1 points are equivalent
  - High-order polynomials can suffer from round-off error
Lagrange Interpolation

Lagrange interpolation of \( \tanh \) using \( n=3 \) fixed points
Lagrange Interpolation

Lagrange interpolation of tanh using n=5 fixed points
Lagrange Interpolation

Lagrange interpolation of \( \tanh \) using \( n=7 \) fixed points
Lagrange Interpolation

Lagrange interpolation of \( \tanh \) using \( n=9 \) fixed points
Lagrange Interpolation

Lagrange interpolation of tanh using n=11 fixed points
Lagrange Interpolation

Lagrange interpolation of tanh using n=13 fixed points
Lagrange Interpolation

Lagrange interpolation of tanh using n=15 fixed points
Lagrange interpolation of \( \tanh \) using \( n=17 \) fixed points
Lagrange Interpolation

Lagrange interpolation of tanh using n=19 fixed points
Lagrange Interpolation

- Notice that, after a bit, increasing the number of interpolating points increases the error
  - This is an example of the Runge phenomena (see, e.g., Wikipedia for a discussion + refs)
    - Oscillation becomes pronounced at the end of the interval with polynomial interpolation
    - Increasing the number of points causes a divergence of the error (it is getting bigger and bigger)

- We can do better by using variable spacing of the interpolating points
  - e.g., Chebyshev polynomial roots are concentrated toward the end of the interval
  - Chebyshev polynomial spacing is usually (almost always) convergent with the number of interpolating points
    - See *Six Myths of Polynomial Interpolation and Quadrature* by L. N. Trefethen
Lagrange Interpolation

Lagrange interpolation of tanh using n=3 variable points

Interpolating points are Chebyshev nodes
Lagrange interpolation of \( \tanh \) using \( n=5 \) variable points

Interpolating points are Chebyshev nodes
Lagrange Interpolation

Lagrange interpolation of tanh using n=7 variable points

Interpolating points are Chebyshev nodes
Interpolating points are Chebyshev nodes
Lagrange Interpolation

Lagrange interpolation of \( \tanh \) using \( n=11 \) variable points

Interpolating points are Chebyshev nodes
Lagrange Interpolation

Lagrange interpolation of tanh using $n=13$ fixed points

Interpolating points are Chebyshev nodes
Lagrange Interpolation

Interpolating points are Chebyshev nodes
Lagrange Interpolation

Lagrange interpolation of $\tanh$ using $n=17$ variable points

Interpolating points are Chebyshev nodes
Lagrange Interpolation

Lagrange interpolation of \( \tanh \) using \( n=19 \) variable points

Interpolating points are Chebyshev nodes

---

PHY 604 Computational Methods in Physics and Astrophysics II
Example: Thermodynamic Consistency

- Stellar equations of state (EOS) can be expensive to evaluate
  - Solution: tabulate the results in terms of $\rho$, $T$, and some (maybe) composition
  - Interpolate to find $p$, $e$, ... everywhere in the $\rho$ – $T$ plane
- Problem: what about thermodynamic consistency?
  - Solution: interpolate a free energy, and derive needed thermodynamic quantities by differentiating the interpolant itself
- This is the approach of the popular helmeos by Timmes & Swesty (2000)
  - Interpolate Helmholtz free energy: $F = e + T s$
  - Via first law: $dF = -s dT + \frac{P}{\rho^2} d\rho$
  
  \[
  P = \rho^2 \left. \frac{\partial F}{\partial \rho} \right|_T, \quad s = - \left. \frac{\partial F}{\partial T} \right|_\rho, \quad e = F + T s
  \]
Example: Thermodynamic Consistency

- Thermodynamic consistency guaranteed
- Now: choice of interpolating function determines accuracy
  - Desire for smoothness of derivatives and second derivatives require higher-order polynomial
  - Biquintic Hermite polynomial gives smoothness, exactly reproduces first and second partial derivatives at grid points
    - requires storing $F$ and eight partial derivatives at each point
Conservative Interpolation

- Imagine that instead of having $f(x)$ at discrete points, we instead knew the average of $f$ in some interval
  - Finite-volume discretization

\[
\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) \, dx
\]

- We want an interpolant that respects these averages
  - Conservative interpolant
Conservative Interpolation

- **Quadratic interpolation**
  - We are given the average of $f$ in each zone
  - Constraints:
    \[
    \langle f \rangle_{i-1} = \frac{1}{\Delta x} \int_{x_{i-3/2}}^{x_{i-1/2}} f(x) \, dx
    \]
    \[
    \langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) \, dx
    \]
    \[
    \langle f \rangle_{i+1} = \frac{1}{\Delta x} \int_{x_{i+1/2}}^{x_{i+3/2}} f(x) \, dx
    \]
  - Reconstruction polynomial
    \[
    f(x) = a(x - x_i)^2 + b(x - x_i) + c
    \]
  - Three equations and three unknowns
Conservative Interpolation

- **Solve for unknowns:**

\[
f(x) = \frac{\langle f \rangle_{i-1} - 2\langle f \rangle_i + \langle f \rangle_{i+1}}{2\Delta x^2} (x - x_i)^2 + \frac{\langle f \rangle_{i+1} - \langle f \rangle_{i-1}}{2\Delta x} (x - x_i) + \frac{-\langle f \rangle_{i-1} + 26\langle f \rangle_i - \langle f \rangle_{i+1}}{24}
\]

- This recovers the proper averages in each interval
- Usually termed *reconstruction*
- We'll see this when we talk about finite-volume methods for advection.
Splines

- So far, we've only worried about going through the specified points
- Large number of points → two distinct options:
  - Use a single high-order polynomial that passes through them all
  - Fit a (somewhat) high order polynomial to each interval and match all derivatives at each point—this is a spline
- Splines match the derivatives at end points of intervals
  - Piecewise splines can give a high-degree of accuracy
- Cubic spline is the most popular
  - Matches first and second derivative at each data point
  - Results in a smooth appearance
  - Avoids severe oscillations of higher-order polynomial
Splines

- We have a set of discrete data: \( f_i = f(x_i) \) at \( x_0, x_1, x_2, \ldots, x_n \)
  - We'll assume regular spacing here

- \( m \)-th order polynomial in \([x_i, x_{i+1}]\)

\[
p_i(x) = \sum_{k=0}^{m} c_{i,k} x^k
\]

- Smoothness requirement: all derivatives match at endpoints

\[
p_i^{(l)}(x_{i+1}) = p_{i+1}^{(l)}(x_{i+1}) \quad l = 0, 1, \ldots, m - 1
\]
Splines

- There are $n$ intervals ([0,1], [1,2], ... [n-1,n]) and in each interval, there are $m+1$ coefficients for the polynomial
  - We have $(m+1)n$ total coefficients
    - Smoothness operates on the $n - 1$ interior points
    - End point values provide 2 more constraints
    - Remaining constraints come from imposing conditions on the second derivatives at end points
- You are solving for the coefficients of all piecewise polynomial interpolants together, in a coupled fashion.
Splines Example

- Here are 3 intervals / 4 points fit with 3 cubic splines
  - Cubic: $m = 3$
  - Intervals (# of cubics): $n = 3$
  - We need $(m+1)n = 12$ constraints

- Interior points
  - At 1: $p_0(x_1) = f_1$
    $p_1(x_1) = f_1$
    $p'_0(x_1) = p'_1(x_1)$
    $p''_0(x_1) = p''_1(x_1)$
  - At 2: $p_1(x_2) = f_2$
    $p_2(x_2) = f_2$
    $p'_1(x_2) = p'_2(x_2)$
    $p''_1(x_2) = p''_2(x_2)$
Splines Example

- At the boundaries:
  \[ p_0(x_0) = f_0 \]
  \[ p_2(x_3) = f_3 \]
- We need 2 more constraints
  - Natural boundary conditions—set the second derivative to 0 at the ends
    \[ p''_0(x_0) = 0 \]
    \[ p''_2(x_3) = 0 \]
  - Other choices are possible
- We now have all the constraints necessary to solve for the set of cubic splines
Cubic Splines

- **Cubic splines:** 3\(^{rd}\) order polynomial in \([x_i, x_{i+1}]\)
  
  1. Start by linearly interpolating second derivatives

  \[ p''_i(x) = \frac{1}{\Delta x} \left[ (x - x_i)p''_{i+1} - (x - x_{i+1})p''_i \right] \]

  2. Integrate twice:

  \[ p_i(x) = \frac{1}{6\Delta x} \left[ (x - x_i)^3 p''_{i+1} - (x - x_{i+1})^3 p''_i \right] + A(x - x_i) + B(x - x_{i+1}) \]

  (note we wrote the integration constants in a convenient form)

  3. Impose constraints: \( p(x_i) = f_i, \ p(x_{i+1}) = f_{i+1} \)

Note: different texts use different forms of the cubic—the ideas are all the same though. This form also seems to be what NR chooses.
Cubic Splines

- Result (after a bunch of algebra):

\[ p_i(x) = \alpha_i(x - x_i)^3 + \beta_i(x - x_{i+1})^3 + \gamma_i(x - x_i) + \eta_i(x - x_{i+1}) \]

\[ \alpha_i = \frac{p''_{i+1}}{6\Delta x} \quad \beta_i = -\frac{p''_i}{6\Delta x} \quad \gamma_i = -\frac{1}{6}p''_{i+1}\Delta x^2 + f_{i+1} \quad \eta_i = \frac{1}{6}p''_i\Delta x^2 - f_i \]

- Note that all the coefficients in the cubic are in terms of the second derivative at the data points.
  - We need to solve for all second derivatives

- Final continuity constraint:

\[ p'_{i-1}(x_i) = p'_i(x_i) \]
Cubic Splines

- After lots of algebra, we arrive at:

\[ p''_{i-1} \Delta x + 4p''_i \Delta x + p''_{i+1} \Delta x = \frac{6}{\Delta x} (f_{i-1} - 2f_i + f_{i+1}) \]

- This is a linear system
- Applies to all interior points, \( i = 1, \ldots, n-1 \) to give \( p''_i \)
- Natural boundary conditions:

\[ p''_0 = p''_n = 0 \]

This is our first example of a linear system—we'll see that these pop up all over the place.
Cubic Splines

- Matrix form:

\[
\begin{pmatrix}
4\Delta x & \Delta x \\
\Delta x & 4\Delta x & \Delta x \\
& \ddots & \ddots & \ddots \\
& \Delta x & 4\Delta x & \Delta x \\
\Delta x & 4\Delta x
\end{pmatrix}
\begin{pmatrix}
p''_1 \\
p''_2 \\
p''_3 \\
\vdots \\
p''_{n-2} \\
p''_{n-1}
\end{pmatrix}
= \frac{6}{\Delta x}
\begin{pmatrix}
f_0 - 2f_1 + f_2 \\
f_1 - 2f_2 + f_3 \\
f_2 - 2f_3 + f_4 \\
\vdots \\
f_{n-3} - 2f_{n-2} + f_{n-1} \\
f_{n-2} - 2f_{n-1} + f_n
\end{pmatrix}
\]

- This is a tridiagonal matrix
- We'll look at linear algebra later—for now we can use a “canned” solver
- Example code...
Cubic Splines
Cubic Splines

Note that the splines can overshoot the original data values
Cubic Splines

- Note: cubic splines are not necessarily the most accurate interpolation scheme (and sometimes far from...)
- But, for plotting/graphics applications, they look right