Interpolation

CS3220 - Summer 2008
Jonathan Kaldor
Interpolation

- We’ve looked at the problem of model fitting using least squares: given some sample points, determine linear parameters for a model that “best” fits the data.
- Note: no guarantees whether or not computed function will pass through any of our data points.
Interpolation

• Suppose instead we know that our data points are correct, and we want to find some function to approximate the points inbetween

• Function should pass through all of our data points, behave reasonably inbetween
Examples

• Have position of an object at set times, want to have reasonable method of determining position *between* times

• Have complex function that is expensive to evaluate, known values at set points, find an approximate function that is cheaper to evaluate
Examples

• When to use linear fitting versus interpolation?
  • Linear fitting: have good guess for correct model, parameters of model are linear, dont mind function not passing through known data points
  • Interpolation: want to pass through known data points, difficult to find good model fit
Interpolant, Formally

- Given data points \((x_i, y_i)\), find function \(F(x)\) such that \(F(x_i) = y_i\) for all \(i\) and \(F(x)\) behaves “reasonably” between data points.
- Reasonably is of course open for interpretation.
- Assume WLOG that \(x_i < x_{i+1}\) for all \(i\).
Interpolant Example
Interpolating Polynomial

- This is the familiar polynomial from earlier
- Exactly fits $n$ data points: degree $n-1$
- Can determine coefficients using monomial basis (a.k.a. 1, $x$, $x^2$, $x^3$, ...)
- Write out equations, solve linear system (matrix is known as a Vandermonde matrix)
- Can be difficult to solve for high degrees
Lagrange Basis Polynomials

- Idea: we want to find some polynomial $P_i(x)$ of degree $n-1$ which is 1 at $x_i$ and 0 at all other $x_j$

- Can scale polynomials by $y_i$ and sum them to form interpolating polynomial
Lagrange Basis Polynomials

- Take 2-data-point case: \((x_1, y_1)\) and \((x_2, y_2)\). We want to find a polynomial \(P_1(x)\) such that \(P_1(x_1) = 1\) and \(P_1(x_2) = 0\).

- Note that \((x - x_2)\) is zero at \(x_2\). Now we just need to make it equal to 1 at \(x_1\). We can do this by multiplying by \(1/(x_1-x_2)\). So \(P_1(x) = (x - x_2)/(x_1 - x_2)\)

- Confirm that this satisfies our conditions
Lagrange Basis Polynomials

- Similarly, $P_2(x) = \frac{(x - x_1)}{(x_2 - x_1)}$

- Our interpolating polynomial is then $F(x) = y_1 P_1(x) + y_2 P_2(x)$

- Confirm: this is degree $n-1$, and it is the same as the equation for the line through the two points
Lagrange Basis Polynomials

• Suppose we had three data points. What would $P_1(x)$ look like?

• Observation: $(x-x_2)(x-x_3)$ is zero at both $x_2$ and $x_3$. Just need to scale so it is 1 at $x_1$. End up with

$$P_1(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}$$

• Degree 2, so we still satisfy the requirements
Generalizing to $N$

- In the general case, we want $P_i(x)$ to be a polynomial that is zero at all $x_j$ where $j \neq i$ and 1 at $x_i$

- $(x-x_1)(x-x_2)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)$ is zero for all of the appropriate $x_j$

- Scale by $\frac{1}{((x_i-x_1)(x_i-x_2)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)$
Generalizing to $N$

- Interpolating polynomial is then
  \[ F(x) = P_1(x)y_1 + P_2(x)y_2 + \ldots + P_n(x)y_n \]

- Can also express this as
  \[ \sum \left( \prod \frac{x-x_j}{x_k-x_j} \right) y_k \]
  \[ k \neq j \]

- Note: easier to find polynomial, but harder to evaluate
Lagrange Basis Example

• Interpolate (1, 3), (2, 1), (4, 2)
Newton Polynomial Basis

- Monomial basis: difficult to find interpolant, easy to evaluate
- Lagrange basis: easy to find interpolant, annoying to evaluate
- Newton basis: compromise between the two (also allows for interpolant to be built up over time as new points are added)
Newton Polynomial Basis

• Idea: find polynomials $\Pi_i(x)$ that are zero for all $x_j, j < i$

• $\Pi_i(x) = (x - x_1)(x - x_2) \ldots (x - x_{i-1})$

• $= \prod_{j=1}^{i-1} (x - x_j)$

• We can then build up interpolating polynomial
Newton Polynomial Basis

- Find $F_1(x)$ that is degree 0 and interpolates $(x_1, y_1)$: simply $F_1(x) = y_1$
- Now find $F_2(x)$ that interpolates both $(x_1, y_1)$ and $(x_2, y_2)$ using $F_1(x)$:
  \[
  F_2(x) = F_1(x) + a_2 \pi_2(x)
  = y_1 + a_2 (x - x_1)
  \]
- Need to choose $a_2$ so that $F_2(x_2) = y_2$ (note: $F_2(x_1) = y_1$ still by our use of $\pi_2(x)$)
Newton Polynomial Basis

- $F_2(x) = y_1 + a_2 (x - x_1)$
- So $a_2 = \frac{(y_2 - y_1)}{(x_2 - x_1)} = \frac{(y_2 - F_1(x_2))}{\pi_2(x_2)}$
- Can compute similar formula for adding third point
Newton Polynomial Basis

- In general, $a_i = \frac{y_i - F_{i-1}(x_i)}{\prod_{i}(x_i)}$

- Analogous to solving lower triangular system (if you write out the equations, it is lower triangular)
Newton Basis Example

- Interpolate (1, 3), (2, 1), (4, 2)
Uniqueness

- HW asks us to prove uniqueness, but our example polynomials don’t look similar...

- We are expressing polynomial in a different basis - they will end up looking different.

- Expanding them and converting them to $a_1 + a_2x + a_3x^2$ should show you they are the same polynomial.

- Not a proof! (but see HW)
Evaluating in Newton Form

• We saw that polynomials in Lagrange Basis were comparatively expensive to evaluate. At first, Newton Basis seems to have the same problem.

• However, observe:

\[ P(x) = a_1 + (x-x_1)a_2 + (x-x_1)(x-x_2)a_3 + ... \]

\[ = a_1 + (x-x_1)(a_2 + (x-x_2)a_3 + ...) \]

\[ = a_1 + (x-x_1)(a_2 + (x-x_2)(a_3 +...)) \]
Evaluating in Newton Form

• This is known as Horner’s method, or nested evaluation
• Can be applied to $a_1 + a_2x + a_3x^2 + ...$ as well
• Ends up being more efficient ($\sim n$ multiplications, $\sim 2n$ additions in Newton Basis) and better in floating point arithmetic
Problems with the Interpolating Polynomial

- Interpolates each of the data points exactly - no gripes here
- But is it “reasonable” in between the data points?
Interpolant Example
Problems with the Interpolating Polynomial

- Seems to behave reasonably around interior data points, but behaves very poorly on edge data points

- Arguably produces unreasonable results on boundary
Alternate Interpolants

- So far, we have considered a single polynomial as an interpolant.
- Suppose instead that we consider piecewise functions.
- Interpolant consists of a set of functions, each defined and used on a disjoint subset of domain.
Piecewise Linear
Piecewise Interpolants

- If we have data points \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\) then it is natural to define a piecewise function for each range \([x_1, x_2], [x_2, x_3], \ldots, [x_{n-1}, x_n]\).

- At each data point there is a knot (where the interpolant changes to a different polynomial).

- Simplest such piecewise function: linear interpolant.
Linear Interpolant
Linear Interpolant

- Piecewise functions are straightforward. On interval \([x_i, x_{i+1}]\):
  \[ F_i(x) = (1-w) y_i + w y_{i+1} \]
  where \( w = \frac{x - x_i}{x_{i+1} - x_i} \)

- Note \( w = 0 \) at \( x = x_i \), and \( w = 1 \) at \( x = x_{i+1} \)
Linear Interpolant

- Easiest piecewise interpolant
- Oftentimes “good enough”
- Dense set of data points
- Downsides: Sharp corners at data points where piecewise functions meet
Continuity of Curves

• Mathematical notion of continuity

• A function is $C^0$ if it is continuous. It is $C^1$ if both it and its first derivative is continuous. In general, it is $C^k$ if its $k$-th derivative is continuous.

• Note: we are assured of being $C^0$ by requirements of interpolant (assuming each piecewise function is continuous)
Linear Interpolant

- Our linear interpolant is continuous but derivative is not continuous (changes discontinuously at data points)
- Would like to specify smoother curve that is $C^1$ or even $C^2$
- Note: this requires us to move away from piecewise linear interpolants (not enough degrees of freedom)
Cubic Interpolant

- We move from linear to cubic interpolants, skipping quadratic (see HW!)
- A Hermite cubic interpolant satisfies interpolation conditions on derivatives as well as data points
Cubic Interpolant
Cubic Interpolant

• How many parameters do we have?
Cubic Interpolant

- How many parameters do we have?
- Each cubic has 4 parameters
Cubic Interpolant

- How many parameters do we have?
- Each cubic has 4 parameters
- We have \( n-1 \) cubics
Cubic Interpolant
Cubic Interpolant

- How many equations do we have?
Cubic Interpolant

- How many equations do we have?
- $2(n-1)$ equations to specify cubic behavior at endpoints
Cubic Interpolant

• How many equations do we have?

• $2(n-1)$ equations to specify cubic behavior at endpoints

• Continuous first derivative: $n-2$ additional equations
Cubic Interpolant

- How many equations do we have?
- $2(n-1)$ equations to specify cubic behavior at endpoints
- Continuous first derivative: $n-2$ additional equations
- $n$ free parameters: can choose them by specifying derivative at each data point, or satisfying other / aesthetic constraints
Catmull-Rom

- Specify derivative at $x_i : (y_{i+1} - y_{i-1})/(x_{i+1} - x_{i-1})$
- Gives $n-2$ additional equations, need 2 more to uniquely determine spline
Cubic Splines

- Want even more continuity than just $C^1$
- Splines are piecewise polynomial curves of degree $k$ which are continuously differentiable $k-1$ times
- Our interest: cubic splines
- Note: Linear interpolation is $k=1$ case
Cubic Splines

- Can derive cubic splines directly from system of linear equations
- Need to enforce interpolation conditions
- Need to enforce continuity of first and second derivative
Cubic Splines

• Have two remaining degrees of freedom

• Can use “natural spline” conditions - second derivative at \( P(x_1) \) and \( P(x_n) \) should be 0

• Also can use not-a-knot

  • Instead of using one cubic for \([x_1, x_2]\) and another for \([x_2, x_3]\), use one cubic for \([x_1, x_3]\). Same for \([x_{n-2}, x_n]\)
Cubic Splines

- Much like polynomial interpolant, can rewrite equations to make finding cubic spline easier
- Ends up becoming a tridiagonal system
- Slightly beyond scope of this course (but try it for yourselves at home)
Cubic Splines in MATLAB

- Computing cubic splines in MATLAB:
  \[ PP = \text{spline}(x, y) \]

- Returns special data structure consisting of piecewise cubic interpolants

- Can compute value of interpolant using
  \[ yy = \text{ppval}(PP, xx) \]
  which evaluates the piecewise functions at the points in \( xx \)
Cubic Splines in MATLAB

- Can also use $yy = \text{spline}(x, y, xx)$, which is shorthand for $yy = \text{ppval}(\text{spline}(x, y), xx)$

- Note: default behavior is to use not-a-knot conditions at endpoints
Cubic Spline Example
Interpolation in Two Dimensions

• Suppose instead of data points \((x_i, y_i)\), we have data points \((x_i, y_k, z_{i,k})\) where \(z_{i,k}\) is the dependent variable.

• Our independent variables \(x_i\) and \(y_k\) can be placed on a grid (compare to 1D case where they were on a line).
Interpolation in Two Dimensions

\[ x_{i-1}, y_{k+1} \quad x_i, y_{k+1} \quad x_{i+1}, y_{k+1} \]

\[ x_{i-1}, y_k \quad x_i, y_k \quad x_{i+1}, y_k \]

\[ x_{i-1}, y_{k-1} \quad x_i, y_{k-1} \quad x_{i+1}, y_{k-1} \]
Interpolation in Two Dimensions

\[ (x, y) \]

The figure illustrates a grid with points \( x_{i-1}, y_{k+1} \), \( x_i, y_{k+1} \), \( x_{i+1}, y_{k+1} \), \( x_{i-1}, y_k \), \( x_i, y_k \), \( x_{i+1}, y_k \), \( x_{i-1}, y_{k-1} \), \( x_i, y_{k-1} \), and \( x_{i+1}, y_{k-1} \). The point \( (x, y) \) is an interpolation point between these grid points.
Interpolation in Two Dimensions

- Recall that our linear interpolation in one dimension between $x_i$ and $x_{i+1}$ looked like
  $$(1-w) v_i + w v_{i+1}$$
  where $w = (x - x_i) / (x_{i+1} - x_i)$ represents the “weight” (how far we are between $x_i$ and $x_{i+1}$)

- Idea: interpolate linearly in one dimension, then in other dimension
Interpolation in Two Dimensions

\[ (x, y) \]
Bilinear Interpolation

- Linearly interpolate between \((x_i, y_k)\) and \((x_i + 1, y_k)\) to get \((x, y_k)\)
- Repeat to get \((x, y_{k+1})\)
- Interpolate between these two values in \(y\) to get interpolant for \((x, y)\)
Bilinear Interpolation

- Substituting everything into linear interpolants gives:
  \[
  (1-a)(1-b) z_{i,k} + a (1-b) z_{i+1,k} + \\
  (1-a)b z_{i,k+1} + a b z_{i+1,k+1}
  \]
  where \( a = (x-x_i)/(x_{i+1}-x_i) \) and \( b = (y-y_i)/(y_{i+1}-y_i) \)

- Note that we can interpolate first in \( y \) and then in \( x \) and still get the same answer.
Other Types of Interpolation

- Our higher degree interpolants have analogs in higher dimensions
- Beyond scope of this course
Other Advantages of Interpolation

- What happens when you have an outlier in interpolation?
  - In least squares?
  - Outliers have a local effect in interpolation
    - Doesn’t affect all of interpolant
    - Somewhat reduces impact of them on result
Outliers