Interpolation

Lagrange Interpolation and Divided Differences

Sachin Shanbhag

Department of Scientific Computing
Florida State University,
Tallahassee, FL 32306.

References

- S. Chapra and R. Canale, Numerical Methods for Engineers
- A. Greenbaum and T. Chartier, Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms
- Carnahan and Wilkes, Applied Numerical Methods, University of Michigan, Class Notes, 1996.
- Burden and Faires, Numerical Analysis, 1993
- Pal, Numerical Analysis for Scientists and Engineers, 2007.
- M. Heath, Scientific Computing: An Introductory Survey
- wikipedia.org

Contents

Why interpolate?

- Replace a continuous function $f(x)$ with something more amenable
  - easier/cheaper to differentiate, integrate
- Sometimes function $f(x)$ known only at discrete points, perhaps, the result of a long simulation or experimental data
  - trajectory of a rocket
  - boiling temperature at a few different pressures
- Interpolation, not regression (approximation) actually passes through the points, not “near”

Prototypical Problem

- Given a discrete set of $n + 1$ points

$$\{x_0, ..., x_i, ..., x_n\},$$

and the function values at those points

$$\{f_0, ..., f_i, ..., f_n\}.$$

- We seek an interpolating *function* $p(x)$ which allows us to compute the value of the function for $x \in [x_0, x_n]$.
- The approximating function can be
  - polynomial
  - Fourier
  - exponential, etc.

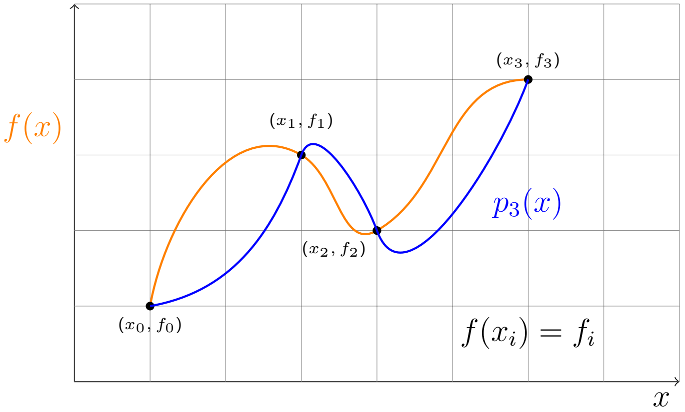
Polynomial Interpolation

- Important and popular

$$p_n(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

- One and only one polynomial of degree n passes through the $n+1$ points (proof at the end of the lecture)
- straight line ($n = 1$) passes through 2 points
- Lagrange and Newton's divided differences methods to get this polynomial
- Note that points may or may not be equally spaced

Picture



Polynomial Interpolation

- Want polynomial to pass through the $n+1$ points

$$\begin{aligned}
a_0 + a_1 x_0 + \cdots + a_n x_0^n &= f_0 \\
a_0 + a_1 x_1 + \cdots + a_n x_1^n &= f_1 \\
&\vdots \\
a_0 + a_1 x_n + \cdots + a_n x_n^n &= f_n
\end{aligned}$$

- In all, $n+1$ equations, $n+1$ unknowns (the a_i)
- Can't I just solve for a_i? Let's see what happens...
- Need to solve the linear system $\mathbf{Xa} = \mathbf{f}$, where $\mathbf{X}$, $\mathbf{a}$, and $\mathbf{f}$ are given by:

Polynomial Interpolation

$$\begin{bmatrix} 1 & x_0 & x_0^2 & ... & x_0^n \\ 1 & x_1 & x_0^2 & ... & x_1^n \\ ... & ... & ... & ... & ... \\ 1 & x_n & x_n^2 & ... & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ ... \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ ... \\ f_n \end{bmatrix}$$

- The matrix $\mathbf{X}$ is called Vandermonde matrix. If x_i are distinct then the determinant of the matrix $|\mathbf{X}| \neq 0$, and the matrix is invertible in principle.
- In practice, it can be very poorly conditioned

```
x = [0:1:10];
v = vander(x);
cond(v)
ans =  4.4628e+12
```

Example

Problem: Consider the function $f(x) = \exp(-0.2x)\cos(\pi x/4)$ between $x \in (1, 10)$. Let us interpolate the function using three points:

x_i	f_i
1.0352	0.5588
4.5967	-0.3558
10.0099	-0.0011

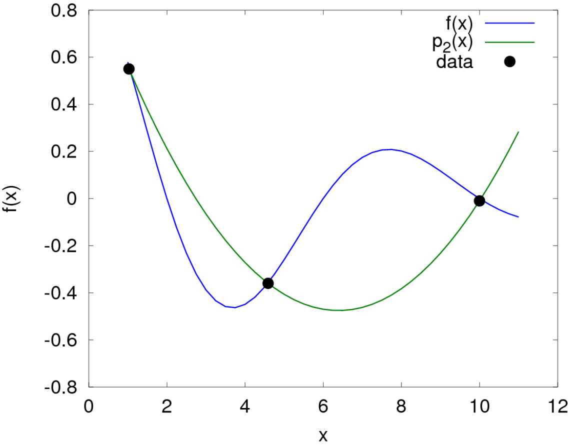
Solution:

The condition number of the corresponding Vandermonde matrix is 172.3. We can solve the system to get

$$\mathbf{a} = \begin{bmatrix} 0.0359 \\ -0.4591 \\ 0.9955 \end{bmatrix}$$

And it seems to fit the data "as expected".

Plot



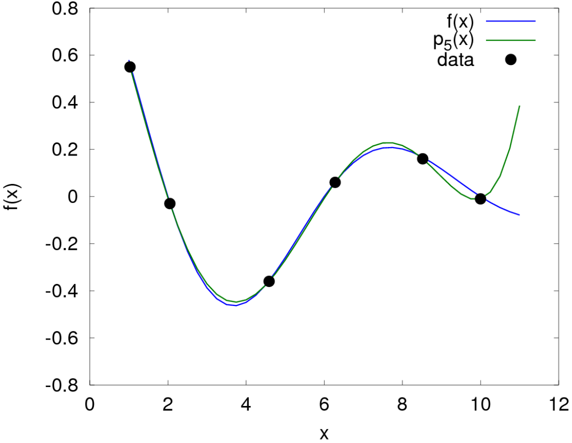
Example

- Let us interpolate the function using 6 points:

x_i	f_i
1.0352	0.5588
2.0568	-0.0295
4.5967	-0.3558
6.2893	0.0640
8.5268	0.1664
10.0099	-0.0011

- The condition number of the corresponding Vandermonde matrix is 1.7 $\times 10^6$.
- We can solve the system to get

Plot



$$\mathbf{a} = \begin{bmatrix} 0.0005 \\ -0.0143 \\ 0.1209 \\ -0.3104 \\ -0.2996 \\ 1.0832 \end{bmatrix}$$

Can we avoid having to directly solving the Vandermonde linear system?

Lagrange Interpolation

- Lagrange's form for the interpolating polynomial

$$p_n(x) = \sum_{i=0}^{n} L_i(x) f_i, \qquad 0 \leq i \leq n,$$

where the Lagrange polynomial $L_i(x)$ is given by,

$$\begin{aligned} L_i(x) &= \prod_{j \neq i}^{n} \frac{x - x_j}{x_i - x_j} \\ &= \frac{(x - x_0)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)} \\ &= \frac{\text{a polynomial of degree } n}{\text{a number}} \end{aligned}$$

Lagrange Interpolation

- For $n+1=2$ points, (x_0, f_0) and (x_1, f_1)

$$\begin{aligned} p_1(x) &= L_0(x)f_0 + L_1(x)f_1 \\ p_1(x) &= \frac{x-x_1}{x_0-x_1}f_0 + \frac{x-x_0}{x_1-x_0}f_1 \end{aligned}$$

- $L_i(x)$ is a polynomial of order n because numerator has n terms involving $(x-a)$
- Thus, $p_n(x)$ is the sum of a bunch of n-order $L_i(x)$
- Note that $L_i(x_k) = \delta_{ik}$ (Kroneker Delta Function), where,

$$\delta_{ik} = \begin{cases} 0, \text{if } i \neq k \\ 1, \text{if } i = k \end{cases}$$

Quadratic Interpolation

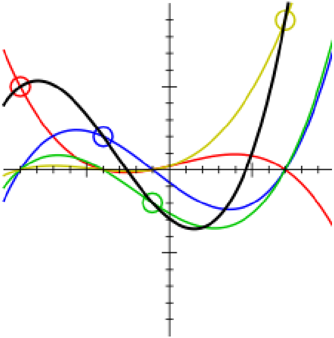
- For $n+1=3$ points, (x_0, f_0), (x_1, f_1) and (x_2, f_2)

$$
\begin{aligned}
f(x) \quad = \quad & \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f_1 \\
& \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f_2
\end{aligned}
$$

- Since $L_i(x_k) = \delta_{ik}$,

$$
p_n(x_k) = \sum_{i=0}^{n} L_i(x_k) f_i = \sum_{i=0}^{n} \delta_{ik} f_i = f_k .
$$

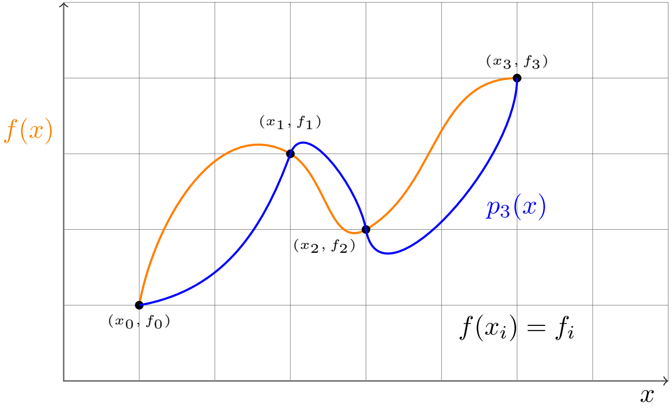
Example



- Four points $(-9, 5), (-4, 2), (-1, -2), (7, 9)$*
- Cubic interpolant, because $(x_0, ..., x_3)$
- $f_0 L_0(x), ..., f_3 L_3(x)$ pass through "their" point
- Zero at the "other" points
- Black curve is the sum of the colored curves $p_3(x)$

*wikipedia.org

Error



How do we quantify the difference between the “true” function $f(x)$ through the points and the interpolating function $p_3(x)$?

Error

- We start by writing:

$$\begin{aligned} f(x) &= p_n(x) + E(x) \\ &= \sum_{i=0}^{n} L_i(x) f_i + E(x) \end{aligned}$$

- It can be shown that the error looks like a truncated Taylor series

$$E(x) = \left[\prod_{i=0}^{n} (x - x_i) \right] \frac{f^{(n+1)}(\xi)}{(n+1)!}$$

where $\xi \in (x_0, x_n)$

Error

- Recall, that a Taylor series allows you to “expand” or approximate a function $f(x)$ around the point a using a polynomial series:

$$f(x) = \underbrace{\sum_{i=0}^{n} \frac{f^{(i)}(a)}{i!}(x-a)^i}_{\text{polynomial}} + \underbrace{\frac{f^{(n+1)}(\xi)}{n+1!}(x-a)^{n+1}}_{\text{error}}$$

where $\xi \in (a, x)$.

Example

- Find a 2nd degree interpolating polynomial passing through the three points (0,-5), (1,1), (3,25).
- Solution

$$
\begin{aligned}
L_0 &= \frac{(x-1)(x-3)}{(0-1)(0-3)} = \frac{x^2-4x+3}{3} \\
L_1 &= \frac{(x-0)(x-3)}{(1-0)(1-3)} = \frac{-x^2+3x}{2} \\
L_2 &= \frac{(x-0)(x-1)}{(3-0)(3-1)} = \frac{x^2-x}{6}
\end{aligned}
$$

- Therefore,

$$p_2(x) = L_0(-5) + L_1(1) + L_2(25) = 2x^2 + 4x - 5^{\dagger}$$

---

†can we find the error bound?

Complexity

- Let us first consider the complexity of the direct method (solving the Vandermonde matrix) to get $p_n(x)$
- Since we have to solve a linear system $(n + 1 \times n + 1)$, the complexity is $\mathcal{O}(n^3)$ to determine the coefficients
- Once the polynomial is detemined, computing $p_n(x)$ at a particular x is $\mathcal{O}(n)$ (Horner's rule - look it up)
- This method is not recommended because the problem is ill-conditioned for large n
- In Lagrange interpolation, we typically don't explicitly compute $p_n(x)$. Instead we simply evaluate $p_n(x)$ at a particular x directly using the formulas described previously

Algorithm

Given the data (xdata, fdata) $= (x_i, f_i)$ for $0 \leq i \leq n$, and a point x, we can naively implement the algorithm as

```
function fx = LagrangeInterpolation(xdata, fdata, n, x)

  sum = 0

  % loop runs over Li * fdata(i)
  for i = 0:n

    product = fdata(i)

    % compute Li
    for j = 0, n

      if(i != j) then
        product = product * (x-xdata(j))/(xdata(i) - xdata(j))
      endif

    endfor

    sum = sum + product

  endfor

  fx = sum

endfunction
```

Complexity

- The number of multiply/divide operations in the inner (j loop) is $2n$ (one mult and one div each turn, skip when $i = j$)
- The outer loop (i loop) runs $(n + 1)$ times.
- Total mult/div is $2n(n + 1) = \mathcal{O}(n^2)$ operations
- If we want to increase the degree of polynomial by adding a new point (x_{n+1}, f_{n+1}), then all the $L_i(x)$ must be recalculated.
- Newton's divided differences addresses these two issues. It solves "reuse" problem, and reduces operations by a constant factor (although still $\mathcal{O}(n^2)$).

Newton's Divided Differences

- The basic idea is to write:

$$f(x) \approx p_n(x) = f_0 + b_0(x - x_0) + b_1(x - x_0)(x - x_1) + \ldots + b_{n-1}\prod_{i=0}^{n-1}(x - x_i)$$

- Note $f(x_0) = p_n(x_0) = f_0$ is automatically satisfied.
- Evaluate coefficients using data (x_i, f_i)

$$f(x_1) = f_1 = p_n(x_1) = f_0 + b_0(x_1 - x_0)$$

$$b_0 = \underbrace{f[x_0, x_1]}_{\text{DD order 1}} = \frac{f_1 - f_0}{x_1 - x_0}$$

Divided Differences

- Similarly

$$f_2 = p_n(x_2) = f_0 + b_0(x_2 - x_0) + b_1(x_2 - x_0)(x_2 - x_1)$$

- With algebra, it can be shown

$$b_1 = \underbrace{f[x_0, x_1, x_2]}_{\text{DD order 2}} = \frac{f[x_0, x_1] - f[x_1, x_2]}{x_0 - x_2}$$

- That is,

$$\begin{aligned} p_n(x) =& f[x_0] + f[x_0, x_1](x - x_0) + \\ & f[x_0, x_1, x_2](x - x_0)(x - x_1) + \ldots \text{ } (n + 1 \text{ terms}) \end{aligned}$$

- Look like Taylor series/derivatives?

Divided Differences

- Ordering does not matter

$$f[x_0, x_1, x_2, ...x_n] = f[x_{i_0}, x_{i_1}, ...x_{i_n}]$$

in particular,

$$f[x_0, x_1, x_2] = f[x_1, x_2, x_0]$$

- General recursion formula

$$f[x_0, x_1, x_2, ...x_n] = \frac{f[x_0, x_1, ..., x_{n-1}] - f[x_1, x_2, ..., x_n]}{x_0 - x_n}$$

in particular, recall

$$f[x_0, x_1, x_2] = \frac{f[x_0, x_1] - f[x_1, x_2]}{x_0 - x_2}$$

Table

x_0	f_0	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$	$f[x_0, x_1, x_2, x_3, x_4]$
x_1	f_1	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	$f[x_1, x_2, x_3, x_4]$	
x_2	f_2	$f[x_2, x_3]$	$f[x_2, x_3, x_4]$		
x_3	f_3	$f[x_3, x_4]$			
x_4	f_4				

- to compute the shaded divided difference, need $\mathcal{O}(n^2)$ operations

$$n + n - 1 + n - 2 + \cdots + 1 = n(n+1)/2$$

- Although only the top row of the table is used, all the terms in the table have to be evaluated
- adding new x_i costs $\mathcal{O}(n)$

Illustration

- Find a degree 2 interpolating polynomial passing through the three points (0,-5), (1,1), (3,25).

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$
0	-5	6	2
1	1	12	
3	25		

- Therefore,

$$p_2(x) = f_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1)$$

$$p_2(x) = -5 + (6)(x) + 2(x)(x-1) = 2x^2 + 4x - 5$$

Code

```
% Newton's Divided Differences: Need to modify the algorithm to account
% for Matlab's inability to handle zero indices.

xdata = [0;1;3];
fdata = [-5;1;25];
x     = 2;

% Compute the Divided Differences

n       = length(xdata) - 1;
DD      = zeros(n+1, n+1);
DD(:,1) = fdata;

for j = 2:n+1
  for i = 1: n + 2 - j
    DD(i,j) = (DD(i+1,j-1) - DD(i,j-1))/(xdata(i+j-1)-xdata(i));
  endfor
endfor

% Evaluate the polynomial at "x"

prodx = 1; fx    = DD(1,1);

for order = 2 : n+1
  prodx = prodx * (x - xdata(order-1));
  fx    = fx + DD(1,order) * prodx;
endfor
```

Historical Perspective

- Newton (1675) was interested in computing the square and cube roots of numbers between 1 and 10,000 to 8 decimal places.
- He found that could set up a nonlinear system, say:

  $$f(x) = x^3 - a = 0$$

  where $a \in (1, 10000)$, which could be solved by his method for solving nonlinear equations.
- Alternatively, he could compute the "easy" cube-roots (1, 8, 27, 64 etc.); and interpolate through them.

Historical Perspective

- For global polynomial interpolation, the so-called Neville's algorithm, which is based on Newton's divided differences, is a default choice
- E. H. Neville was the British mathematician who "discovered" S. Ramanujan, and introduced him to G. H. Hardy.
- Lagrange presented his namesake polynomials in 1795, in the context of a surveying problem he was interested in.
- He was unaware that the same formula had been published by Waring in 1779, and Euler in 1783.

Lagrange versus Divided Differences

- "Lagrangian interpolation is praised for analytic utility and beauty but deplored for numerical practice." (Acton, 1990)
- Generally assumed to be useful for proving theorems, but not a good computational algorithm.
- Lagrange's form for the interpolating polynomial

$$p_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i), \qquad L_i(x) = \prod_{j \neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

- Shortcomings include:
  1. Each evaluation of $p(x)$ requires $\mathcal{O}(n^2)$ operations.
  2. Adding a new data pair (x_{n+1}, f_{n+1}) requires a new computation from scratch.
  3. The computation can become numerically unstable if the x_i are too close.

Lagrange versus Divided Differences

- With divided differences
  1. The asymptotic cost of constructing the polynomial is $\mathcal{O}(n^2/2)$ operations.
  2. Evaluation of $p(x)$ at a particular x requires $\mathcal{O}(n)$ operations.
  3. Adding a new data pair $(x_{n+1},\ f_{n+1})$ requires $\mathcal{O}(n)$ operations.
  4. Generalizability to incorporate derivative data (Hermite interpolation)
- Often (in Newton's cube-root problem for instance), we don't know the order of the polynomial to use. Divided differences allows us to increase the order of the polynomial systematically.
- Lagrange interpolation needs to know the order of the polynomial before hand, which can be problematic at times

Appendix: Proof of Uniqueness

Theorem

If two polynomials of order n pass through the same n+1 points, then the polynomials are identical

We try to prove the theorem by contradiction.

- Assume $P(x)$ and $Q(x)$ are two distinct n order polynomials which pass through the $(n+1)$ points (x_k, f_k), with $k = 0, 1, ...n$
- Let $R(x) = P(x) - Q(x)$. $R(x)$ is also a n^{th} order polynomial which passes through the $(n+1)$ points $(x_k, 0)$.
- This implies that $R(x)$, a n^{th} order polynomial, has $n+1$ zeros, when it can have at most n zeros. Contradiction!
- Hence, P(x) and Q(x) are not distinct