Newton Interpolation

- Consider our data set of n+1 points $y_i = f(x_i)$ at $x_0, x_1, \ldots, x_n$.
- Since $p_n(x)$ is the unique polynomial $p_n(x)$ of order $n$, write it:

  \[ p_n(x) = b_0 + b_1(x-x_0) + b_2(x-x_0)(x-x_1) + \cdots + b_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}) \]

  \[ b_0 = f(x_0) \]
  \[ b_1 = f[x_1, x_0] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \]
  \[ b_2 = f[x_2, x_1, x_0] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0} \]
  \[ \vdots \]
  \[ b_n = f[x_n, \ldots, x_0] = \frac{f[x_n, \ldots, x_1] - f[x_{n-1}, \ldots, x_0]}{x_n - x_0} \]

- $f[x_p, x_q]$ is a first divided difference
- $f[x_p, x_q, x_0]$ is a second divided difference, etc.
- Efficient way of adding points to the interpolation!
- Used to fit data to a table
Error

- Define the error term as:

\[ \varepsilon_n(x) = f(x) - p_n(x) \]

- If \( f(x) \) is an \( n \)th order polynomial \( p_n(x) \) is of course exact.
- Otherwise, since there is a perfect match at \( x_0, x_1, \ldots, x_n \)
- This function has at least \( n+1 \) roots at the interpolation points.

\[ \therefore \varepsilon_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n)h(x) \]

Interpolation Errors

- Suppose we want to measure error at a point \( x \)
- To make polynomial go through \( x \), add to existing polynomial divided difference term.
- This is the error we make using existing polynomial

\[ x \notin \{x_0, x_1, \ldots, x_n\} \]

\[ \varepsilon_n(x) = f(x) - p_n(x) = f[x_0, x_1, \ldots, x_n, x] \prod_{i=0}^{n} (x - x_i) \]

- Comparing with Taylor series

\[ f[x_0, x_1, \ldots, x_n] = \frac{1}{n!} f^{(n)}(\xi) \]
Interpolation Errors

\[ \varepsilon_n(x) = f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i) \]
\( x \in [a, b], \xi \in (a, b) \)

- Looks a bit like Taylor series remainder
- Recall, first \( n+1 \) terms of the Taylor Series is also an \( n^{th} \) degree polynomial.

Interpolation: the story so far

- Given a function at \( N \) points, find its value at other point(s)
- So far: polynomial interpolation
  - Polynomials are guaranteed to approximate any given function in an interval as accurately as we want
- Different polynomial bases
  - Monomial or Power basis
  - Newton and Lagrange basis
- For a given set of points and function values
  - interpolating polynomial is unique
- Interpolation problem requires solution of a linear system
  - System is dense for Monomial/Power basis
  - Newton and Lagrange forms allow the direct solution of the polynomial interpolation form
  - Newton form particularly convenient to add new values
- Error for interpolation with \( n \) points is related to the value of the \( (n+1)^{th} \) derivative of the underlying function
Polyinterp

• Lagrange interpolation code
  – x,y are points and function values
  – u are points where value is needed

  function v = polyinterp(x,y,u)
  n = length(x);
  v = zeros(size(u));
  for k = 1:n
    %Lagrange function k at u
    w = ones(size(u));
    for j = [1:k-1 k+1:n]
      w = (u-x(j))./(x(k)-x(j)).*w;
    end
    v = v + w*y(k);
  end

• Cost: 2 nested loops, so the cost is \( n^2 \).
  • \( k = 5, \ n = 9 \)
  • \( j = [1:k-1 \ k+1:n] \)
  • \( j = 1 \ 2 \ 3 \ 4 \ 6 \ 7 \ 8 \ 9 \)

Examples of polynomial interpolation

• Go to MATLAB demo
  – Vandermonde
  – Polynomial interpolation for small set
  – For larger set

• See that even for six points we have a problem
  – In between the data points, (especially in first and last subintervals), function shows excessive variation.
  – overshoots changes in the data values.
  – As a result, full-degree polynomial interpolation is hardly ever used for data and curve fitting.

• However we saw polynomial interpolation works well when degree is low