Thursday, 10/29/13 Math 471, Fall 2013, Section 001 Topic 6

1 Interpolation and approximation

The problem of data fitting (i.e., interpolation or approximation) arises in all aspects of science and engineering. The most obvious example of this problem arising in an application occurs with measured data: suppose you have a set of discrete measurements of a function, but you require that function's value at a point that is not one of the measurement locations.

But you can find the interpolation/approximation problem even in contexts where there are no measurements. Suppose you have a model that uses a mesh (like a finite difference method), but you (or somebody else) needs information from your model at locations that don't coincide with your mesh's gridpoints – you will have to interpolate or approximate.
In general, we face the choice of the following two problems in almost everything we do:

The interpolation problem. Given a set of locations $\{x_j\}$, $j = 1, \dots, n$ in $\mathbb{R}^d$ and a corresponding set of function values f_j, we seek a (continuous) function P_f such that

$$P_f(x_j) = f_j, \qquad j = 1, \dots, n.$$

The approximation problem. Given a set of locations $\{x_j\}$, $j = 1, \dots, n$ in $\mathbb{R}^d$ and a corresponding set of function values f_j, we seek a (continuous) function P_f such that

$$P_f(x_j) \approx f_j, \qquad j = 1, \dots, n.$$

Question 1: Why would approximation be sometimes more desirable than interpolation?

There are more conditions we may demand of the interpolating or approximating function; for example, we may wish that it have a minimum number of derivatives, in addition to being continuous.
It is convenient to pose this problem as needing to find coefficients that represent P_f as a linear combination of a set of basis functions,

$$P_f(x) = \sum_{k=1}^{m} c_k B_k(x)$$

where $B_k(x)$ are the basis functions.
Solving the interpolation problem then reduces to solving the linear system of equations

$$Ac = f,$$

for the vector c, the vector of coefficients. The entries of the interpolation matrix A are given by $A_{jk} = B_k(x_j)$ and the right-hand side vector f are the function values $f_j = f(x_j)$, which are given from the data.

Question 2: What about other applications, such as approximating integrals and derivatives? In other words, if $f(x) \approx P_f(x)$,

- Is $\int_a^b f(x)\,dx \approx \int_a^b P_f(x)\,dx$?
- Is $f'(x) \approx P_f'(x)$?

Let's start simple, and work with $B_k(x)$ that are polynomials.

2 Polynomial interpolation

Recall: a polynomial of degree n can be written in the form $p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$.
Note: there are other equivalent forms to write polynomials, as we shall see.

Theorem 1 (Weirstrauss approximation theorem)**.** *Let $f(x)$ be continuous on the closed interval $[a,b]$. Then for any $\epsilon > 0$ there exists a polynomial $p(x)$ such that $\max\limits_{a\le x\le b} |f(x) - p(x)| < \epsilon$.*

Proof. Math 451.
The implication of this theorem is that any continuous function can be approximated to arbitrary accuracy by polynomials.

2.1 Taylor polynomials

Definition 2. Given $f(x)$ and a point $x = a$, the Taylor polynomial of degree n about $x = a$ is

$$p_n(x) = f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2 + \cdots + \frac{1}{n!}f^{(n)}(a)(x-a)^n.$$

The Taylor polynomial satisfies the following conditions:

1. $p_n(a) = f(a)$, $p_n'(a) = f'(a)$, $\ldots$, $p_n^{(n)}(a) = f^{(n)}(a)$.
2. $f(x) = p_n(x) + r_n(x) \quad : \quad r_n(x) =$ remainder , error
3. $r_n(x) = \displaystyle\int_a^x \frac{(x-t)^n}{n!} f^{(n+1)}(t)\,dt = \frac{1}{(n+1)!} f^{(n+1)}(\xi)(x-a)^{n+1}$ for some point $\xi \in [a, a+x]$.

$$\Rightarrow |f(x) - p_n(x)| \le \frac{1}{(n+1)!} \max_{a\le t\le x} \left| f^{(n+1)}(t) \right| \cdot |x-a|^{n+1} \quad : \quad \text{error bound}$$

Taylor polynomials are may be written as $p_n(x) = \sum_{k=1}^n c_k (x-a)^k$, like the above form; this would imply an expansion using the basis $B_k(x) = (x-a)^k$, and the coefficients $c_k = \frac{1}{n!} f^{(k)}(a)$. Note that the Taylor polynomials are an approximating method; the do not satisfy the interpolation matrix.
Example 1: Find $p_n(x)$ about $a = 0$ for $f(x) = \dfrac{1}{1+25x^2}$.
Direct approach : find formulas for all n derivatives, plug in $x = a$.
Better idea : use geometric series to avoid lots of differentiation.
recall : $1 + r + r^2 + r^3 + \cdots = \begin{cases} \frac{1}{1-r} & |r| < 1 \\ \text{diverges} & |r| \ge 1 \end{cases}$

$$\frac{1}{1+25x^2} = \frac{1}{\underbrace{1-(-25x^2)}_{r=-25x^2}} = 1 + (-25x^2) + (-25x^2)^2 + (-25x^2)^3 + \cdots$$

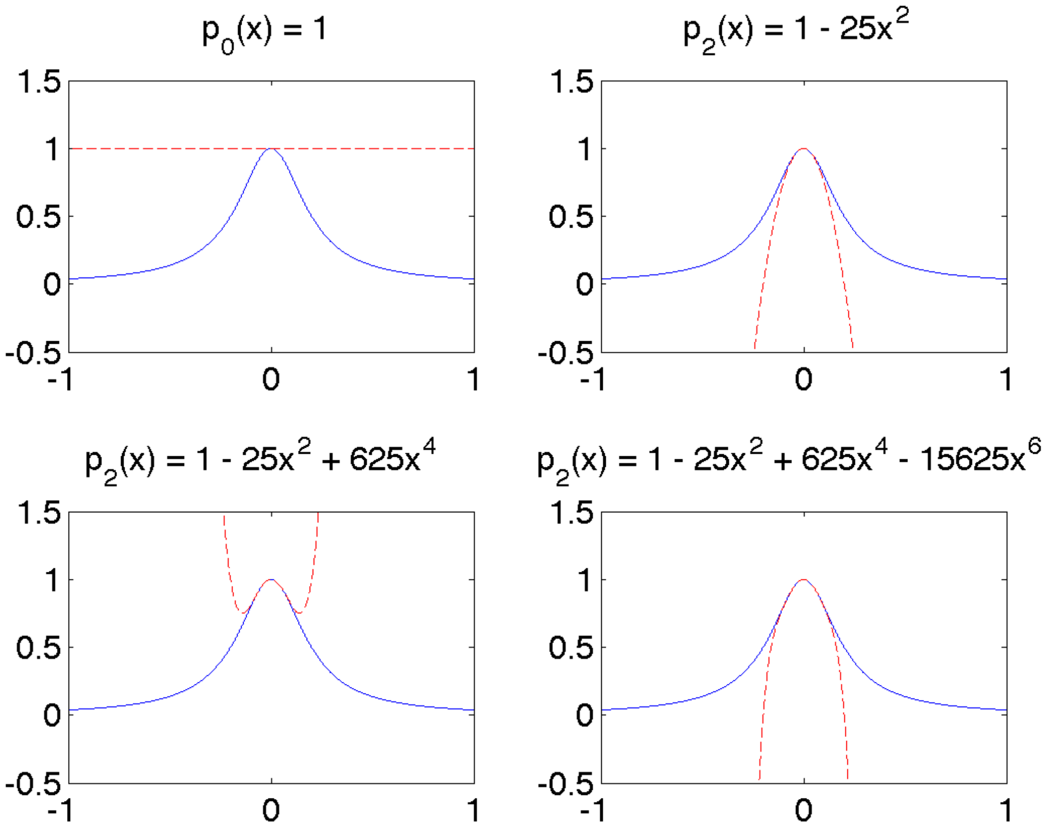
Note:
$\left|-25x^2\right| < 1 \Rightarrow |x| < \dfrac{1}{5} \Rightarrow \lim\limits_{n\to\infty} p_n(x) = f(x)$ for $-\frac{1}{5} < x < \frac{1}{5}$ for $|x| \geq \frac{1}{5}$, $\lim\limits_{n\to\infty} p_n(x)$ diverges

Explanation : $f(x)$ has singularities at $x = \pm\frac{1}{5}i$ in the complex plane (Math 555)

Take a look at p_n and $f(x)$:

```
clear;
%% Taylor polynomials

x = -1:0.001:1;

f = 1./(1 + 25*x.^2);

p0 = ones(size(x));

p2 = 1-25*x.^2;

p4 = p2 + 625*x.^4;

p6 = p4 - 15625*x.^6;

figure(1);clf;
subplot(2,2,1);
plot(x,f,'b-',x,p0,'r--');
set(gca,'FontSize',16);
title('p_0(x) = 1');
xlim([-1,1]);
ylim([-0.5,1.5]);

subplot(2,2,2);
plot(x,f,'b-',x,p2,'r--');
set(gca,'FontSize',16);
title('p_2(x) = 1 - 25x^2');
xlim([-1,1]);
ylim([-0.5,1.5]);

subplot(2,2,3);
plot(x,f,'b-',x,p4,'r--');
set(gca,'FontSize',16);
title('p_2(x) = 1 - 25x^2 + 625x^4');
xlim([-1,1]);
ylim([-0.5,1.5]);

subplot(2,2,4);
plot(x,f,'b-',x,p6,'r--');
set(gca,'FontSize',16);
title('p_2(x) = 1 - 25x^2 + 625x^4 - 15625x^6');
```

```
xlim([-1,1]);
ylim([-0.5,1.5]);

saveas(1,'taylorPolys.png');
```



△

Taylor polynomials are good approximations when f is sufficiently differentiable and x is close to a.
What about other methods of approximation?

2.2 General polynomial interpolation

Text : section 5.2

Theorem 3. *Let $x_0, x_1, \ldots, x_n$ be $n+1$ distinct points and $f_0, f_1, \ldots, f_n$ be $n+1$ corresponding data values of a function $f(x)$. Then there exists a unique polynomial $p_n(x)$ of degree $\leq n$ such that $p_n(x)$ interpolates f, i.e., $p_n(x_i) = f(x_i)$ for each $i = 0, \ldots, n+1$.*

Proof. 1. We'll prove the existence of this polynomial soon.

2. Uniqueness is guaranteed by the Fundamental Theorem of Algebra (Math 412).

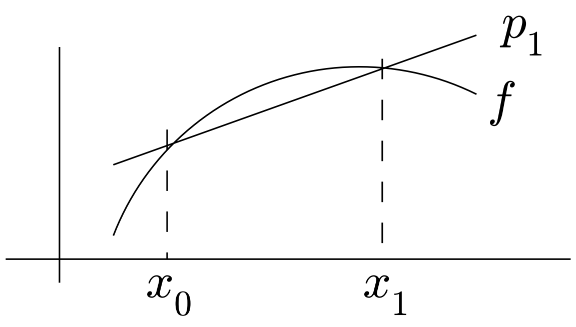
□

Example 2:

Case $n = 2 \Rightarrow x_0, x_1$.

Goal : Find a linear polynomial $p_1(x) = a_0 + a_1 x$ such that $p_1(x_0) = f_0$ and $p_1(x_1) = f_1$. Easy to show that $a_0 = \dfrac{x_1 f_0 - x_0 f_1}{x_1 - x_0}$ and $a_1 = \dfrac{f_1 - f_0}{x_1 - x_0}$.

Then $p_n(x) = \dfrac{x_1 f_0 - x_0 f_1}{x_1 - x_0} + \dfrac{f_1 - f_0}{x_1 - x_0} x$



△

Question 3:

1. How can we construct $p_n(x)$ for larger n?
2. Is there an efficient way to evaluate $p_n(x)$ for $x \neq x_i$?
3. How large is the error $|p_n(x) - f(x)|$ for $x \neq x_i$?

Definition 4. The kth Lagrange polynomial is

$$L_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \left(\frac{x - x_i}{x_k - x_i} \right), \quad k = 0, \dots, n$$

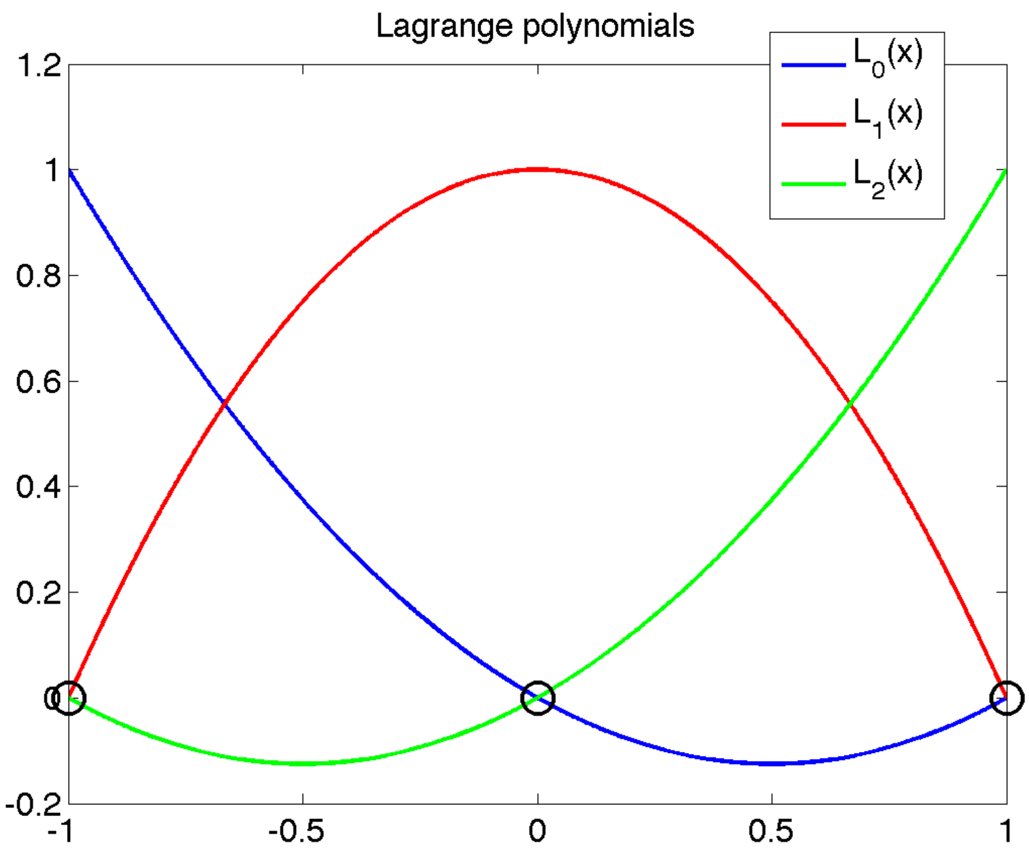
is a polynomial of degree n associated with data point k.

Example 3: $\quad n = 2, \quad x_0 = -1, \quad x_1 = 0, \quad x_2 = 1$

$$L_0(x) = \prod_{i=1}^{2} \left(\frac{x - x_i}{x_k - x_i} \right) = \left(\frac{x - x_1}{x_0 - x_1} \right) \cdot \left(\frac{x - x_2}{x_0 - x_2} \right) = \left(\frac{x - 0}{-1 - 0} \right) \cdot \left(\frac{x - 1}{-1 - 1} \right) = \frac{1}{2}x^2 - \frac{1}{2}x$$

$$L_1(x) = \prod_{\substack{i=0 \\ i \neq 1}}^{2} \left(\frac{x - x_i}{x_k - x_i} \right) = \left(\frac{x - x_0}{x_1 - x_0} \right) \cdot \left(\frac{x - x_2}{x_1 - x_2} \right) = \left(\frac{x - (-1)}{0 - (-1)} \right) \cdot \left(\frac{x - 1}{0 - 1} \right) = -x^2 + 1$$

$$L_1(x) = \prod_{i=0}^{1} \left(\frac{x - x_i}{x_k - x_i} \right) = \left(\frac{x - x_0}{x_2 - x_0} \right) \cdot \left(\frac{x - x_1}{x_2 - x_1} \right) = \left(\frac{x - (-1)}{1 - (-1)} \right) \cdot \left(\frac{x - 0}{1 - 0} \right) = \frac{1}{2}x^2 + \frac{1}{2}x$$



△

Properties of Lagrange polynomials:

1. The degree of each L_k is n
2. $L_k(x_i) = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases}$

Interpolating using the Lagrange polynomials:

Let $f(x)$ be given. The <u>Lagrange form</u> of the interpolating polynomial is

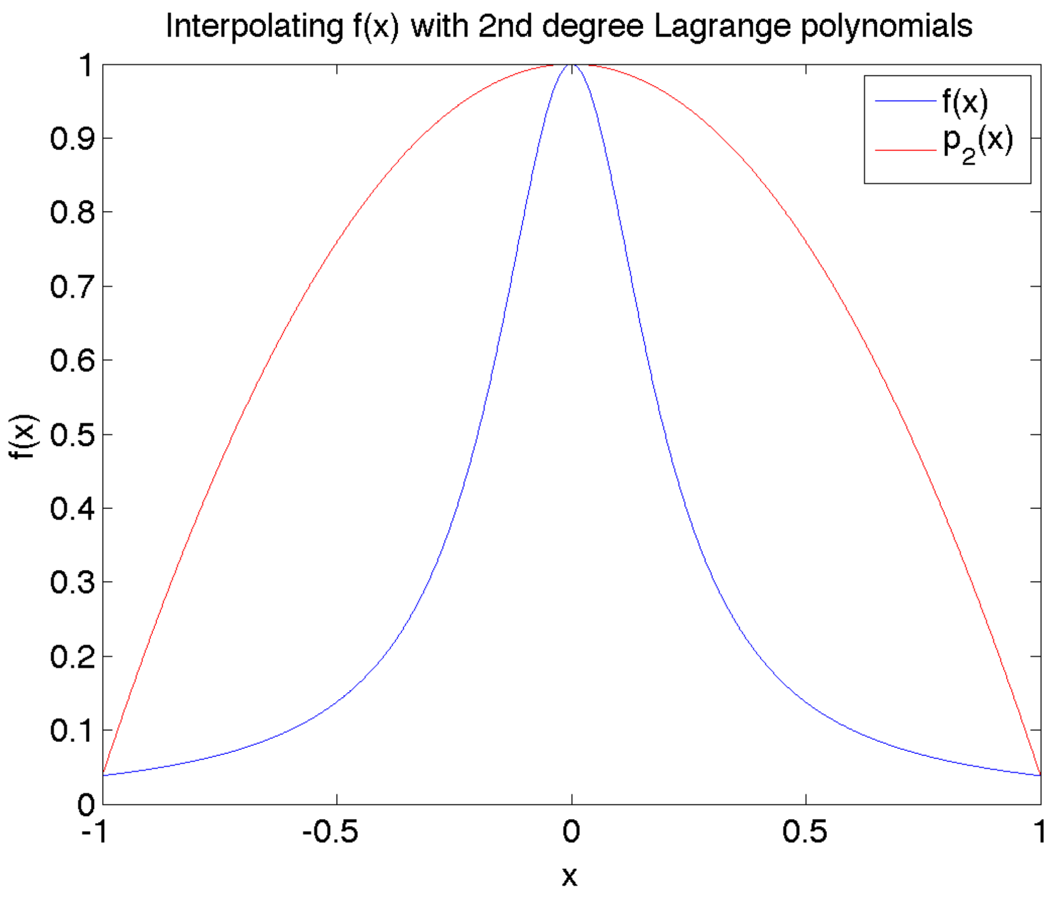
$$p_n(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x) + \cdots + f(x_n)L_n(x) = \sum_{k=0}^{n} f(x_k)L_k(x)$$

$\Rightarrow$ Using the basis of Lagrange polynomials, the coefficients are the data values.

<u>Example 4:</u> $\quad f(x) = \dfrac{1}{1+25x^2}, \quad x_0 = -1, \quad x_1 = 0 \quad x_2 = 1$

$p_2(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x)$

$p_2(x) = \dfrac{1}{26} \cdot \left(\dfrac{1}{2}x^2 - \dfrac{1}{2}x\right) + 1 \cdot (x^2 - 1) + \dfrac{1}{26}\left(\dfrac{1}{2}x^2 + \dfrac{1}{2}x\right) = -\dfrac{25}{26}x^2 + 1$



Notes:

- The 2nd degree interpolating polynomial here is different than the 2nd degree Taylor polynomial centered at $x = 0$.
- This polynomial interpolates the data at the given points x_i.
- The approximation near $x = 0$ is much worse than the Taylor polynomial, but elsewhere in the domain this approximation is much better.

△

Advantages of the Lagrange form are :

- They are helpful in theory, for example, to establish the existence of $p_n(x)$.
- They are helpful conceptually: the data f_j are separated from the locations x_i.

Disadvantages of the Lagrange form are :

- It is expensive to evaluate $p_n(x)$ for $x \neq x_i$ using this form (for small n, this is not a problem)
- Changing the mesh, for example, by adding an additional point x_{n+1} means that all $L_k(x)$ must be recalculated.

2.3 Polynomial interpolation : Newton's form

Recall: A unique polynomial of degree n exists that interpolates data at $n+1$ locations,

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$$

We have seen that this form of the polynomial is subject to cancellation error (hw 1). Alternatively, we have proposed the Lagrange form of $p_n(x)$, given by the basis of Lagrange polynomials.

We saw that Lagrange polynomials are useful theoretically (for proving existence of p_n, for example), but can be costly to use for computing when n is large. In this section we look for a more efficient means of constructing and evaluating $p_n(x)$.

Text : section 5.3

Definition 5. The Newton form of an interpolating polynomial $p_n(x)$ that interpolates a function f is

$$p_n(x) = c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + \cdots c_n(x-x_0)\cdots(x-x_{n-1}).$$

In this case the basis functions $B_k(x)$, $k = 0, 1, \ldots, n$ are

$$B_k(x) = \prod_{i=0}^{k-1}(x - x_i).$$

Example 5: $n = 1, \quad x_0, \quad x_1$

$$\begin{aligned} p_1(x) &= f(x_0)\left(\frac{x-x_1}{x_0-x_1}\right) + f(x_1)\left(\frac{x-x_0}{x_1-x_0}\right) : \text{ Lagrange form} \\ &= f(x_0) + \left(\frac{f(x_1)-f(x_0)}{x_1-x_0}\right)(x-x_0) : \text{ Newton form} \end{aligned}$$

△

For $n \geq 2$, we need a method to compute $a_0, a_1, \ldots, a_n$.

1. Solve the interpolating matrix. We may write $p_n(x) = \sum_{k=0}^{n} c_k B_k(x)$ and impose the interpolating condition $p_n(x_i) = f(x_i)$ for $i = 0, 1 \ldots, n$ to arrive at the linear system

$$\begin{pmatrix} B_0(x_0) & B_1(x_0) & \cdots & B_n(x_0) \\ B_0(x_1) & B_1(x_1) & \ldots & B_n(x_1) \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ B_0(x_n) & \ldots & \ldots & B_n(x_n) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ \vdots \\ f_n \end{pmatrix}$$

   The definition of the Newton form of $p_n(x)$ makes this matrix lower-triangular, which can be solved by forward substitution.

   $i = 0: \quad p_0(x_0) = c_0 = f_0$

$$i = 1: \quad p_1(x_1) = \underbrace{c_0}_{p_0(x_1)} + c_1(x_1 - x_0) = f_1 \quad \Rightarrow \quad c_1 = \frac{f_1 - p_0(x_1)}{x_1 - x_0}$$

$$i = 2: \quad p_2(x_1) = \underbrace{c_0 + c_1(x - x_0)}_{p_1(x)} + c_2(x - x_0)(x - x_1) = f_2 \quad \Rightarrow \quad c_2 = \frac{f_2 - p_1(x_2)}{(x_2 - x_0)(x_2 - x_1)}$$

$$\vdots$$

$$i = n: \quad p_n(x_n) = p_{n-1}(x_n) + c_n(x - x_0)\cdots(x - x_{n-1}) = f_n \;\Rightarrow\; c_n = \frac{f_n - p_{n-1}(x_n)}{(x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-1})}$$

Here we have defined polynomials $p_0(x)$ as the polynomial that interpolates f at $x = x_0$, and $p_1(x)$ as the polynomial that interpolates f at $x = 0$ and $x = x_1$... Then $p_i(x)$ is the polynomial that interpolates f at all $x = x_k$ for $k = 0, 1, \ldots, i$.

2. Divided differences.

Definition 6. The Newton form of the interpolating polynomial may be given by divided differences, where the kth divided difference between points x_k and x_j with $k + j \leq n$ is

$$f[x_j, x_{j+1}, x_{j+2}, \ldots, x_{j+k}] = \frac{f[x_{j+1}, \ldots, x_{j+k}] - f[x_j, \ldots, x_{j+k-1}]}{x_{j+k} - x_j}$$

Thus, each divided difference is computed from the divided differences that precede it, starting with $f[x_0] = f_0$.

Theorem 7. *The coefficients of the Newton form of the interpolating polynomial are given by the divided difference formulas*

$$c_0 = f[x_0], \; c_1 = f[x_0, x_1], c_2 = f[x_0, x_1, x_2], \ldots$$

Proof. Let $p_{n-1}(x)$ interpolate $f(x)$ at the first $n - 1$ points, i.e., at $x = x_0, x_1, x_2, \ldots, x_{n-1}$.

Let $q_{n-1}(x)$ interpolate $f(x)$ at the next $n - 1$ points, i.e., at $x = x_1, x_2, \ldots, x_n$.

Then both $p_{n-1}(x)$ and $q_{n-1}(x)$ have degree $\leq n - 1$.

Define $g(x) = \left(\dfrac{x - x_0}{x_n - x_0}\right) q_{n-1}(x) + \left(\dfrac{x_n - x}{x_n - x_0}\right) p_{n-1}(x)$.

Then $\deg g \leq n$ and $g(x)$ interpolates f at $x = x_0$ and $x = x_n$:

$g(x_0) = p_{n-1}(x_0) = f_0$

$g(x_n) = q_{n-1}(x_n) = f_n$

For $i = 1 : n - 1$, $g(x_i) = \left(\dfrac{x_i - x_0}{x_n - x_0}\right) q_{n-1}(x_i) + \left(\dfrac{x_n - x_i}{x_n - x_0}\right) p_{n-1}(x_i)$

$\Rightarrow g(x_i) = f_i \left(\dfrac{x_i - x_0}{x_n - x_0}\right) + f_i \left(\dfrac{x_n - x_i}{x_n - x_0}\right) = f_i$

Thus, $g(x)$ is a degree $\leq n$ polynomial that interpolates f at points x_i for $i = 0, \ldots, n \Rightarrow g(x) = p_n(x)$.

Note that $p_{n-1}(x)$ and $q_{n-1}(x)$ are equal to each other at x_i for $i = 1 : n-1$. Starting from $n = 2$, we can set the coefficients of x^n from these two polynomials equal to each other to find that

$$\underbrace{\frac{f[x_1, \dots, x_n]}{x_n - x_0}}_{} - \underbrace{\frac{f[x_0, \dots, x_{n-1}]}{x_n - x_0}}_{} = f[x_0, x_1, \dots, x_n]$$

$\square$

Hence, we can write the Newton form of $p_n(x)$ as

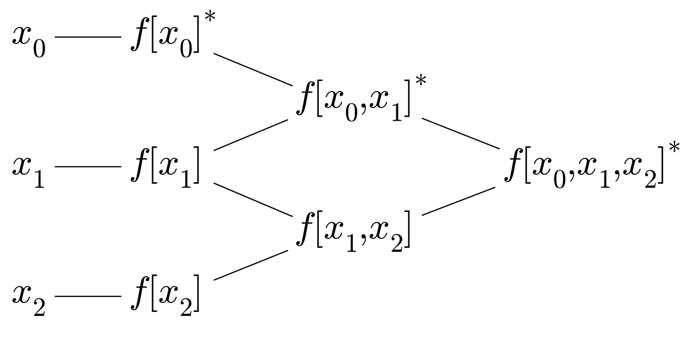
$$\begin{aligned} p_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + \\ f[x_0, x_1, x_2](x - x_0)(x - x_1) + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) + \cdots \\ + f[x_0, \dots, x_n](x - x_0) \cdots (x - x_{n-1}) \end{aligned} \tag{1}$$

We can organize the computation of divided differences into a table:

$$\begin{array}{llllll}
x_0 & f[x_0] & & & \\
 & & f[x_0, x_1] & & \\
x_1 & f[x_1] & & f[x_0, x_1, x_2] & \dots \\
 & & f[x_1, x_2] & & \\
x_2 & f[x_2] & & f[x_1, x_2, x_3] & \dots \\
 & & f[x_2, x_3] & \vdots & \\
x_3 & f[x_3] & \vdots & & \\
\vdots & \vdots & & &
\end{array}$$

This table builds the interpolating polynomial $p(x)$ one term at a time and interpolates f at one additional point. Each intermediate step $k \leq n$ interpolates the first $k+1$ points. Note that adding a point does not change the previous basis functions (unlike the Lagrange form of the interpolating polynomial).

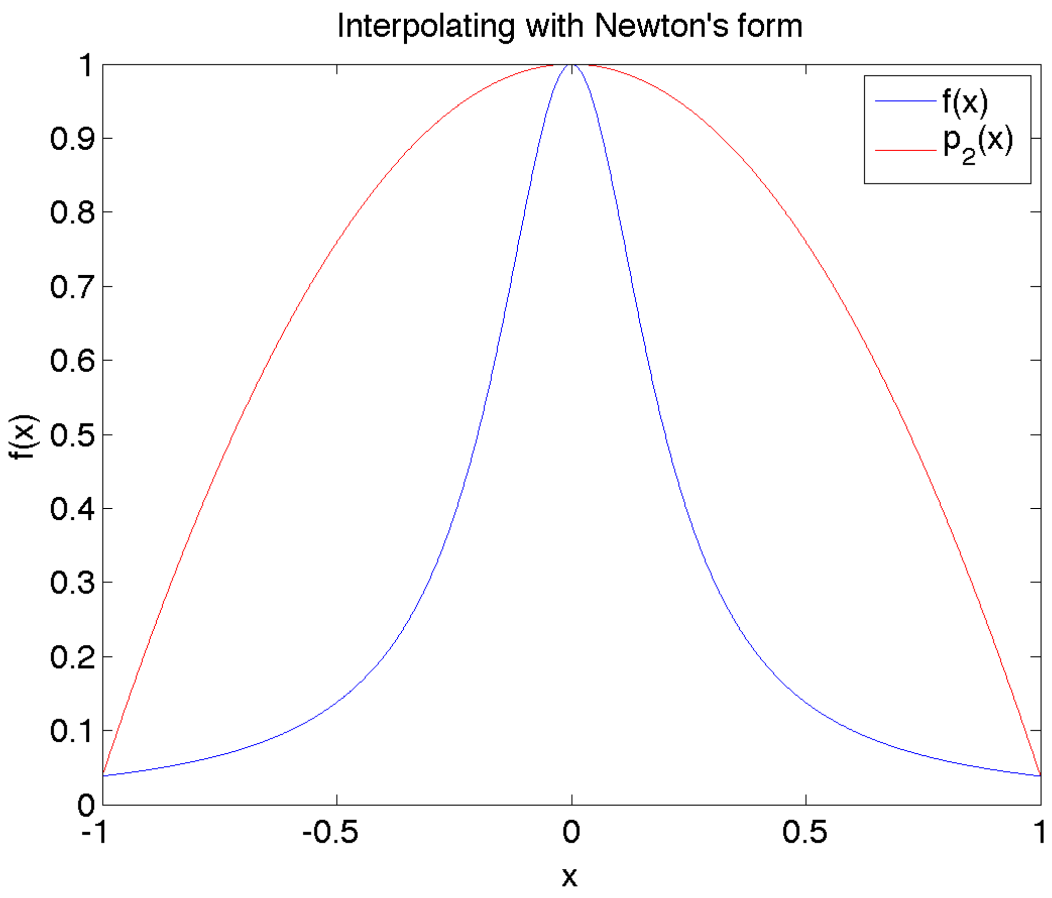
Example 6: $\quad n = 2,\ x_0 = -1,\ x_1 = 0,\ x_2 = 1,\ f(x) = \dfrac{1}{1 + 25x^2}$



x_i	$f[\,\cdot\,]$	$f[\,\cdot\ \ \cdot\,]$	$f[\,\cdot\ \ \cdot\ \ \cdot\,]$
-1	$\frac{1}{26}$		
		$\frac{25}{26}$	
0	1		$-\frac{25}{26}$
		$-\frac{25}{26}$	
1	$\frac{1}{26}$		

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

$$p_2(x) = \frac{1}{26} + \frac{25}{26}(x + 1) - \frac{25}{26}x(x + 1)$$



$\triangle$

2.3.1 Operation counts

evaluation of $p_n(x)$

$$\begin{aligned} p_2(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) : \text{ Newton form, 3 mults.} \\ &= a_0 + (x - x_0)\,(a_1 + a_2(x - x_1)) : \text{ Nested form, 2 mults.} \end{aligned}$$

general case

$$p_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x1) + \cdots + a_n(x-x_0)\cdots(x-x_{n-1}) : \text{ Newton form}$$
$$p_n(x) = a_0 + (x-x_0)\left(a_1 + (x-x_1)(a_2 + (x-x_2)(a_2 + \cdots + a_n(x-x_n)))\cdots\right) : \text{ nested form}$$

Same number of additions. Multiplication count: Newton form $= \frac{n(n-1)}{2}$, nested form $= n$ (cheap!).

Error in polynomial interpolation

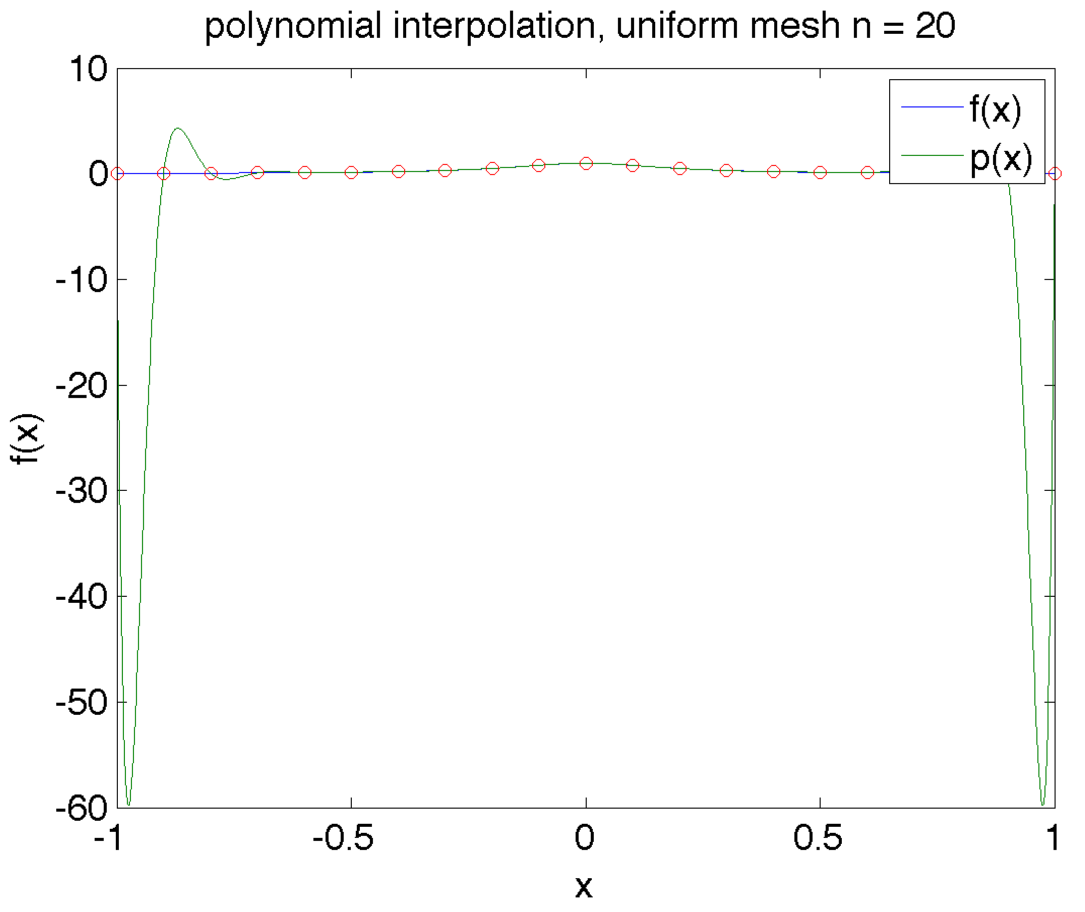
Theorem 8. *For a differentiable function $f(x)$ and interpolation points $x_0 < x_1 < \cdots < x_n$, the interpolating polynomial $p_n(x)$ satisfies, on the interval $x_0 \le x \le x_n$,*

$$f(x) = p_n(x) + \frac{1}{(n+1)!} f^{(n+1)}(\xi)(x-x_0)(x-x_1)\cdots(x-x_n)$$

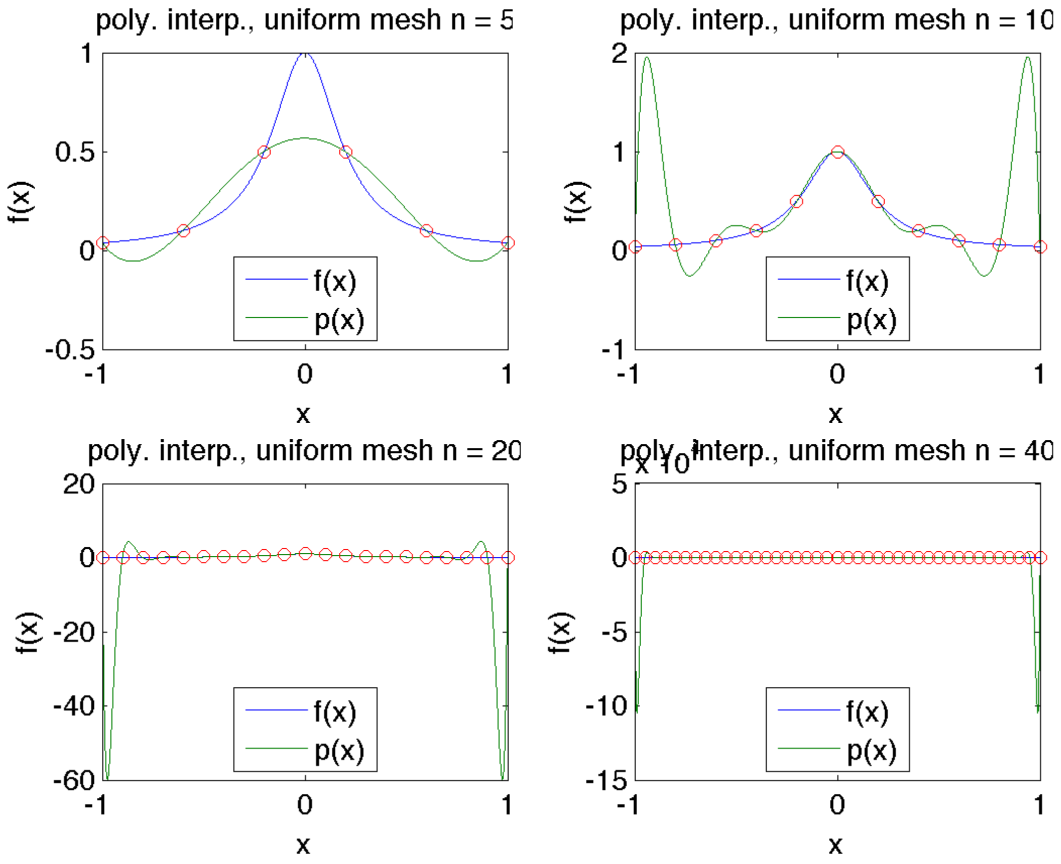
Proof. Omitted. Note that this resembles the error in Taylor series approximation.

Example 7: Error bound, $n=1$. $a \le x \le b \Rightarrow |f(x) - p_1(x)| \le \frac{1}{8} M \left|b-a\right|^2$, where $M = \max\limits_{a\le x\le b} \left|f''(x)\right|$. (hw 6)

Return to example :



We see that the polynomial does well near the center of the domain, but has very high error near the boundaries. This problem does not get better with a more refined mesh (in fact, it gets worse).



This is known as Runge's phenomenon, and it is due to the fact that on the outer parts of an interval, when using equidistant points, the polynomial interpolation problem is sometimes ill-conditioned. Note, this ill-conditioning is a feature of the problem, not the numerical method. Lookup Matlab's `polyfit` and `polyval` tools.

Thursday, 11/7/13

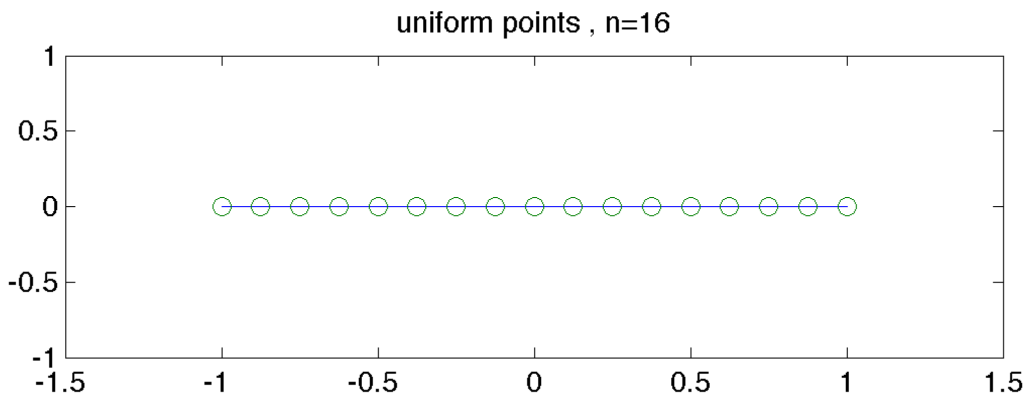
2.4 Optimal points for interpolation

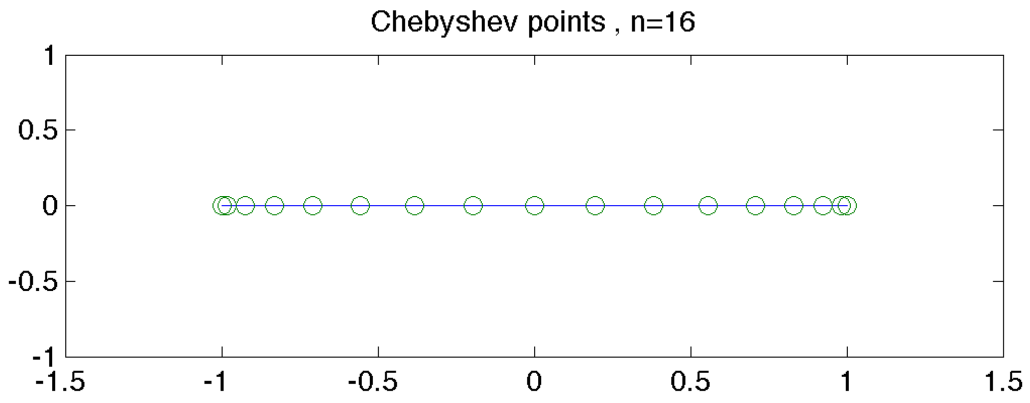
Text : section 5.4

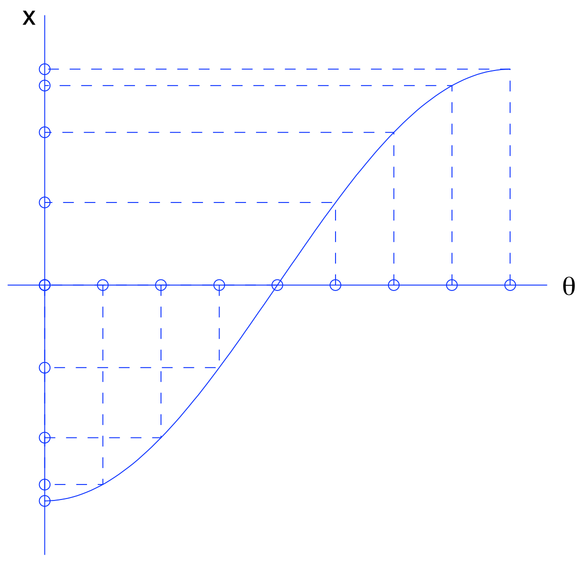
Interpolation problem, part b: Suppose you know that you will have to use interpolation, perhaps because the function you need to evaluate is not known or expensive to compute; where should you place your mesh points?
Equivalently, suppose your lab only has enough funding to measure n data points over a domain $D \subseteq \mathbb{R}^3$, where should you place your measurement devices to minimize interpolation error?
Consider two options for the interval $x \in [-1, 1]$:

1. uniform mesh : $x_i = -1 + ih, \quad h = \frac{2}{n}, \quad i = 0 : n.$

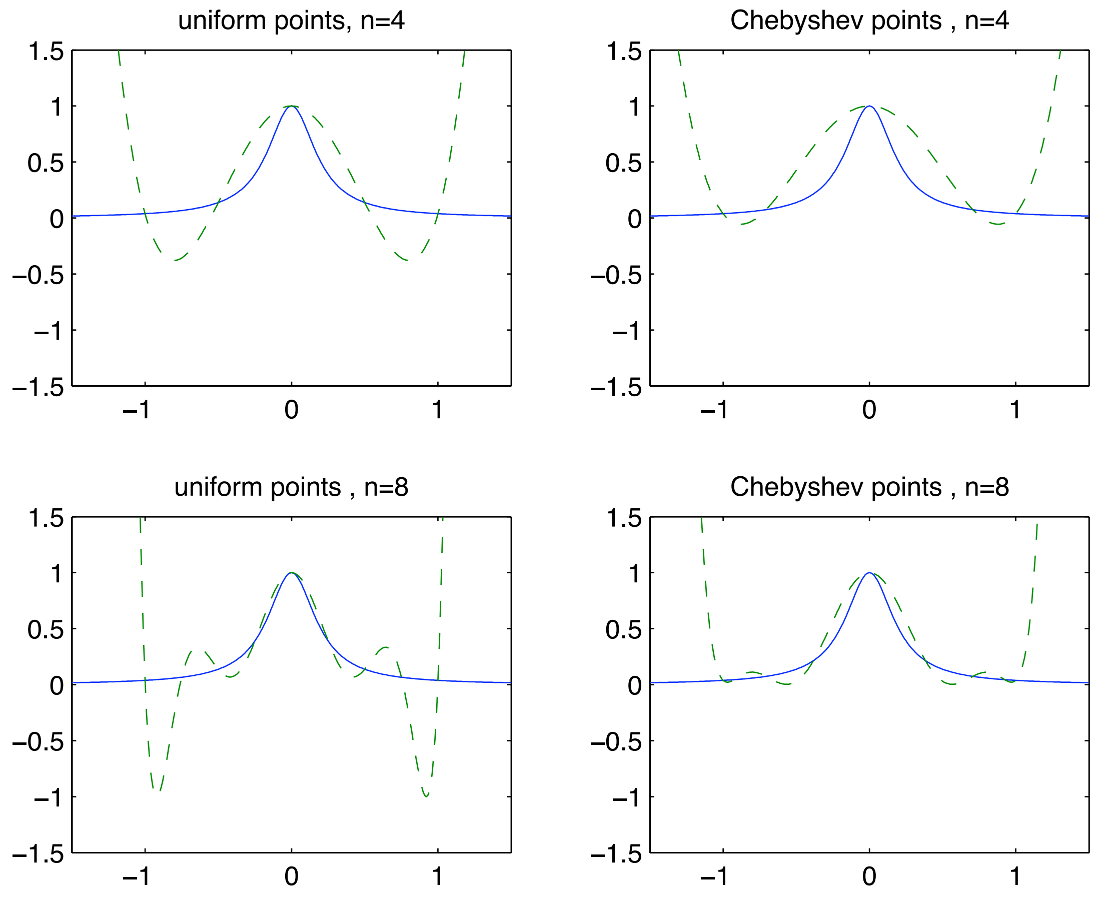
2. Chebyshev points : $x_i = -\cos\theta_i, \quad \theta_i = i\frac{\pi}{n}, \quad i = 0 : n.$

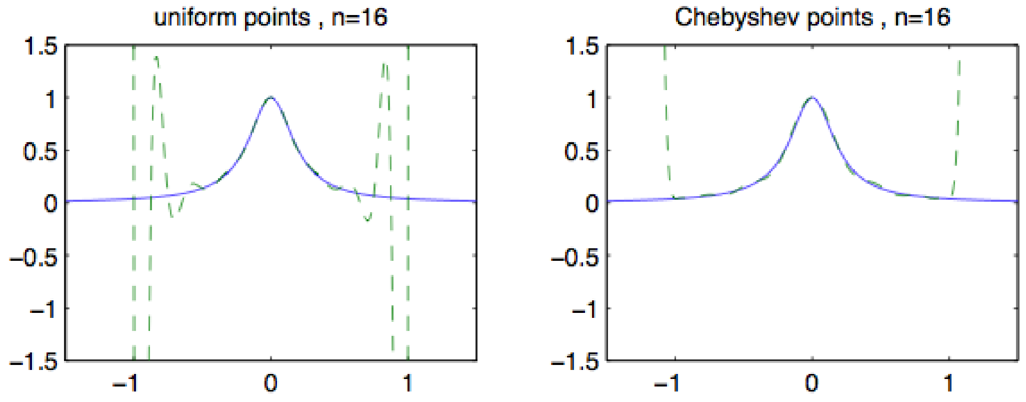






The cosine function concentrates grid points toward the endpoints of the interval.





Notes:

- Interpolating on a set of Chebyshev points gives a good approximation over the whole interval; uniform meshes only achieve good accuracy near the center of the interval.
- The Chebyshev points are roots of the Chebyshev polynomials, an important set of orthogonal polynomials. The points define the optimal locations for minimizing the L_∞ error of the interpolating polynomial over the interval $x \in [a, b]$.
- They are a set of orthogonal polynomials. Another set are the Legendre polynomials,

whose roots give the optimal points for minimizing the L_2 norm of the polynomial $p_n(x)$.

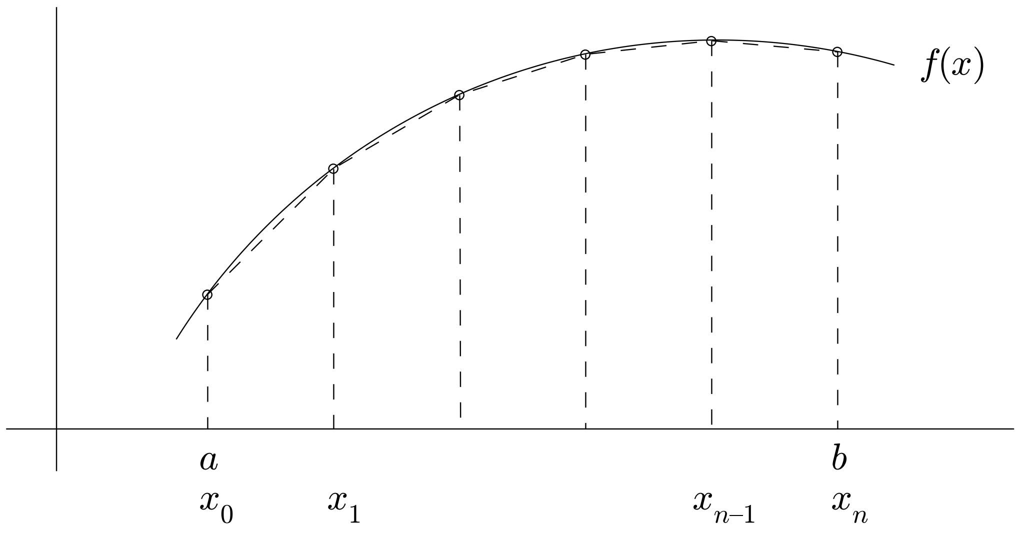
3 Piecewise polynomial interpolation

We have seen that although unique, the polynomial $p_n(x)$ that interpolates a function f on $n+1$ data points may not be well conditioned. In situtations where we can control the interpolation locations, we can use Chebyshev points or other special points to minimize interpolation error. For cases where we do not have this flexibility, it is often more useful to use lower-degree polynomials, perhaps even lots of them.
The interpolating polynomial $p_n(x)$ may not be a good approximation of $f(x)$ on the whole interval, so instead we may consider piecewise linear interpolation, denoted by $q(x)$.

Text : section 5.5

Given $f(x),\ a \le x \le b,\quad a = x_0 < x_1 < x_2 \cdots < x_{n-1} < x_n = b$

$$q(x) = \begin{cases} f[x_0] + f[x_0,x_1](x-x_0), & x_0 \le x < x_1 \\ \vdots & \vdots \\ f[x_i] + f[x_i,x_{i+1}](x-x_i), & x_i \le x < x_{i+1} \\ \vdots & \vdots \\ f[x_{n-1}] + f[x_{n-1},x_n](x-x_{n-1}), & x_{n-1} \le x \le x_n \end{cases}$$



Notes:

- $q(x)$ is continuous, but not differentiable, at $x = x_i$
- error : $|f(x) - q(x)| \le \dfrac{1}{8} \max\limits_{a \le x \le b} \left|f''(x)\right| \cdot \max\limits_i |x_{i+1} - x_i|^2$: 2nd order accurate
- Piecewise interpolation is often referred to as a local method, as the function is locally approximated by a line at each grid point.

- Local polynomials of higher degree may also be used; a famous technique in fluid dynamics and finite volume methods is known as the "Piecewise Parabolic Method," or PPM.

3.1 cubic spline interpolation

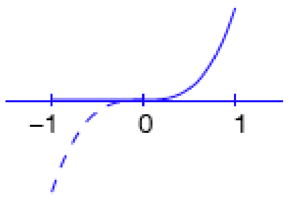
Text : section 5.6

Definition 9. A cubic spline is a function $s(x)$ that satisfies the following conditions.

1. For each subinterval, $x_i \le x \le x_{i+1}$, $s(x)$ is a cubic polynomial.
2. $s(x)$, $s'(x)$, and $s''(x)$ are continuous at the interior points $x_1, \dots, x_{n-1}$.

Example 8: $n = 2,\ x_0 = -1, x_1 = 0, x_2 = 1.$

$$s(x) = \begin{cases} 0, & -1 \le x \le 0 \\ x^3, & 0 \le x \le 1 \end{cases}$$



$$s'(x) = \begin{cases} 0, & -1 \le x \le 0 \\ 3x^2, & 0 \le x \le 1 \end{cases}$$

$$s''(x) = \begin{cases} 0, & -1 \le x \le 0 \\ 6x, & 0 \le x \le 1 \end{cases}$$

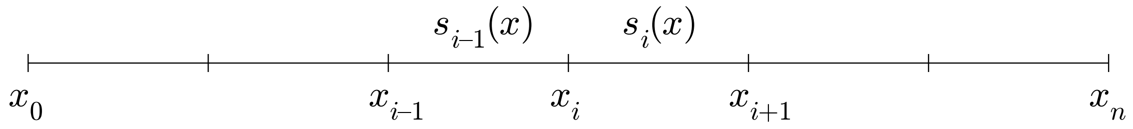
$s(x)$ satisfies all conditions, hence s is a cubic spline.

△

cubic spline interpolation problem

Given $f(x)$ and $a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$, find a cubic spline $s(x)$ which interpolates $f(x)$ at the given points, i.e., $s(x_i) = f(x_i)$ for $i = 0 : n$.

On each subinterval $x_i \le x \le x_{i+1}$, $s(x) = s_i(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$: 4 coefficients.



$n+1$ points $\Rightarrow$ n intervals $\Rightarrow$ $4n$ unknown coefficients

Interpolation conditions : $\Rightarrow 2n$ equations : $s_{i-1}(x_i) = f(x_i), \quad s_i(x_i) = f(x_i)$

Continuity of $s'(x), s''(x)$ at interior points $\Rightarrow 2(n-1)$ equations : $s'_{i-1}(x_i) = s'_i(x_i), \quad s''_{i-1}(x_i) = s''_i(x_i)$

Remaining 2 equations come from a choice of how to handle the boundary conditions. A popular choice is to set $s''(x_0) = s''(x_n) = 0$, which gives rise to the natural cubic spline interpolant.

Example 9: Finding $s(x)$ on a uniform mesh.

Divide the interval $x \in [-1, 1]$ into n subintervals $\Rightarrow n+1$ points.

$x_i = -1 + ih,\ h = \dfrac{2}{n},\ i = 0, \dots, n$: uniform mesh

step 1: 2nd derivative conditions

$s_i''(x)$ is a linear polynomial $\Rightarrow s_i''(x) = a_i \left(\dfrac{x_{i+1} - x}{h}\right) + a_{i+1}\left(\dfrac{x - x_i}{h}\right)$

Note that $s_i''(x_i) = a_i$ and $s_i''(x_{i+1}) = a_{i+1} \Rightarrow s_{i-1}''(x_i) = a_i = s_i''(x_i)$

Hence $s''(x)$ is continuous at the interior points.

step 2: interpolation

Integrate twice:

$s_i'(x) = \displaystyle\int s_i''(x)\,dx = -\frac{a_i}{2h}(x - x_{i+1})^2 + \frac{a_{i+1}}{2h}(x - x_i)^2 + \frac{a_i}{2h}x_{i+1}^2 - \frac{a_{i+1}}{2h}x_i^2 + C_0,\quad C_0 =$ constant of integration

Define $b_i = \dfrac{a_{i+1}}{2}x_i^2 - \dfrac{1}{2}C_0$ and $c_i = \dfrac{a_i}{2}x_i^2 + \dfrac{1}{2}C_0$ so that

$$s_i'(x) = -\frac{a_i}{2h}(x - x_{i+1})^2 + \frac{a_{i+1}}{2h}(x - x_i)^2 - \frac{b_i}{h} + \frac{c_i}{h}.$$

$$s_i(x) = \int s_i'(x)\,dx = \frac{a_i}{6h}(x_{i+1} - x)^3 + \frac{a_{i+1}}{6h}(x - x_i)^3 - \frac{b_i}{h}x + \frac{c_i}{h}x + C_1$$

Define $C_1 = b_i \dfrac{x_{i+1}}{h} - c_i \dfrac{x_i}{h}$; then

$$s_i(x) = \frac{a_i}{6h}(x_{i+1} - x)^3 + \frac{a_{i+1}}{6h}(x - x_i)^3 + b_i\left(\frac{x_{i+1} - x}{h}\right) + c_i\left(\frac{x - x_i}{h}\right)$$

Interpolation conditions: $s(x_i) = f(x_i)$

$$s_i(x_i) = \frac{a_i h^2}{6} + b_i = f_i \Rightarrow b_i = f_i - \frac{a_i h^2}{6}$$

$$s_i(x_{i+1}) = \frac{a_{i+1}h^2}{6} + c_i = f_{i+1} \Rightarrow c_i = f_{i+1} - \frac{a_{i+1}h^2}{6}$$

$$s_i(x) = \frac{a_i}{6h}(x_{i+1} - x)^3 + \frac{a_{i+1}}{6h}(x - x_i)^3 + \left(f_i - \frac{a_i h^2}{6}\right)\left(\frac{x_{i+1} - x}{h}\right) + \left(f_{i+1} - \frac{a_{i+1}h^2}{6}\right)\left(\frac{x - x_i}{h}\right)$$

step 3: 1st derivative conditions

$$s_i'(x) = -\frac{a_i}{2h}(x - x_{i+1})^2 + \frac{a_{i+1}}{2h}(x - x_i)^2 - \frac{1}{h}\left(f_i - \frac{a_i h^2}{6}\right) + \frac{1}{h}\left(f_{i+1} - \frac{a_{i+1}h^2}{6}\right)$$

$$s_i'(x_i) = -\frac{a_i h}{2} - \frac{f_i}{h} + \frac{a_i h}{6} + \frac{f_{i+1}}{h} - \frac{a_{i+1}h}{6}$$

$$s_i'(x_{i+1}) = \frac{a_{i+1}h}{2} - \frac{f_i}{h} + \frac{a_i h}{6} + \frac{f_{i+1}}{h} - \frac{a_{i+1}h}{6}$$

Continuity : $s'_{i-1}(x_i) = s'_i(x_i)$

$$\frac{a_i h}{2} - \frac{f_{i-1}}{h} + \frac{a_{i-1}h}{6} + \frac{f_i}{h} - \frac{a_i h}{6} = -\frac{a_i h}{2} - \frac{f_i}{h} + \frac{a_i h}{6} + \frac{f_{i+1}}{h} - \frac{a_{i+1}h}{6}$$

$$\frac{a_{i-1}h}{6} + a_i\left(\frac{h}{2} - \frac{h}{6} + \frac{h}{2} - \frac{h}{6}\right) + \frac{a_{i+1}h}{6} = \frac{f_{i-1}}{h} - \frac{f_i}{h} - \frac{f_i}{h} + \frac{f_{i+1}}{h}$$

$$a_{i-1} + 4a_i + a_{i+1} = \frac{6}{h^2}\left(f_{i-1} - 2f_i + f_{i+1}\right),\ i = 1 : n-1$$

step 4 : boundary conditions

$s''_0(x_0) = 0 \Rightarrow a_0 = 0, \quad s''_{n-1}(x_n) = 0 \Rightarrow a_n = 0$

$$\begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 4 \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ \vdots \\ \vdots \\ a_{n-1} \end{pmatrix} = \frac{6}{h^2} \begin{pmatrix} f_0 - 2f_1 + f_2 \\ \vdots \\ \vdots \\ \vdots \\ f_{n-2} - 2f_{n-1} + f_n \end{pmatrix}$$

Matrix A is symmetric, tridiagonal, and positive definite; it is also strictly diagonally dominant, which implies that a unique solution vector exists.

Tuesday, 11/12/13

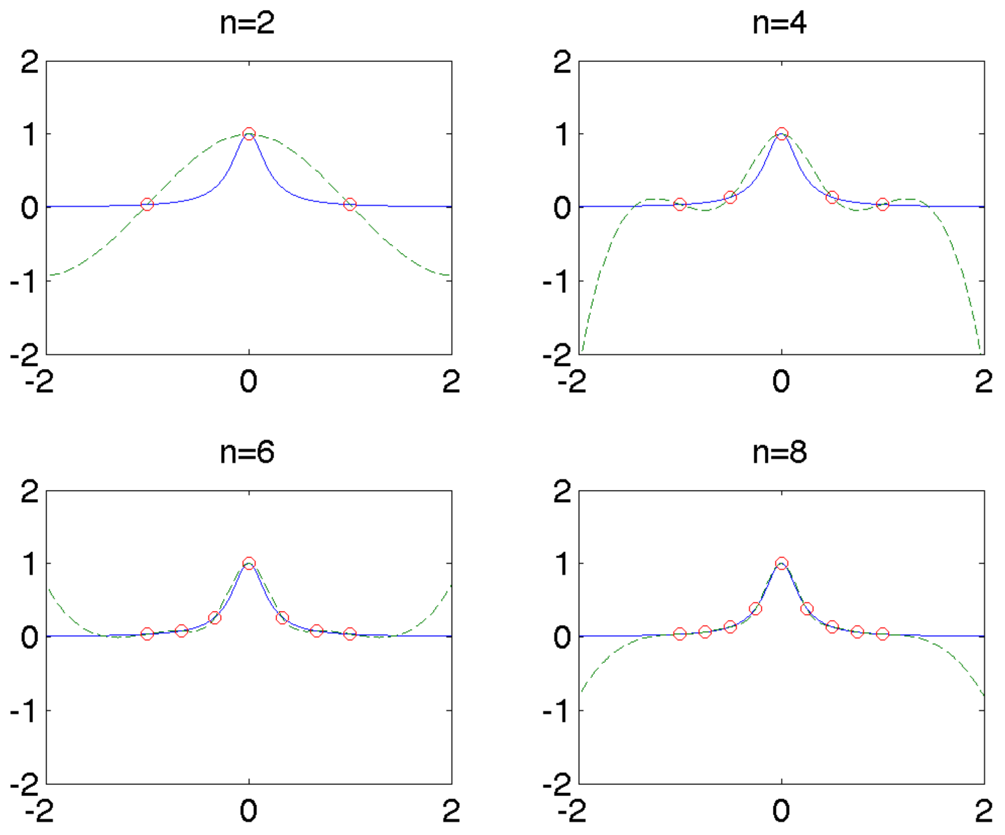
Notes:

- If $f''(x) = 0$ at $x = x_0$ and $x = x_n$, then the natural cubic spline is 4th order accurate.
- Clamped boundary conditions : If $f'(a)$ and $f'(b)$ are known, apply them to $s'(x_0)$ and $s'(x_n)$; then the clamped cubic spline is 4th order accurate.

$\triangle$

Example 10: natural cubic spline interpolation

$$f(x) = \frac{1}{1+25x^2}, \quad -1 \le x \le 1,\ x_i = -1 + ih, \quad h = \frac{2}{n}, i = 0 : n$$



$\triangle$

4 Hermite interpolation

Text : section 5.7

Another way of handling Runge's phenomenon is to notice that although the interpolating polynomial $p_n(x)$ interpolates the data $p_n(x_i) = f(x_i)$ at each point x_i, the polynomial may intersect the graph of f at very steep angles. It would be better if the interpolating function $p(x)$ was parallel to $f(x)$ at the interpolating points; that its, we would like $p'(x_i) = f'(x_i)$. Cubic splines come close to achieving this, but don't quite get there.

Suppose we have values for $f(x_i)$ *and* $f'(x_i)$ on an interval $a \le x \le b$ with n subdivisions and $n+1$ points,

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b, \quad h = \frac{b-a}{n}, \quad x_i = a + ih, \quad i = 0, \ldots, n$$

we want to determine a polynomial $p(x)$ that interpolates the data and goes through each point (x_i, f_i) with slope $f'(x_i)$, that is, the polynomial p should satisfy

$$p(x_i) = f(x_i), \quad p'(x_i) = f'(x_i), \quad i = 0, \ldots, n$$

Note that the data include the function values and the function derivatives, each at $n+1$ points, for a total of $2n+2$ input values to the interpolation problem; hence we know p can have degree of at most $2n+1$.

Theorem 10. *Let the interval $a \le x \le b$ be divided into n subintervals by $n+1$ distinct points, and the function f and its first derivative f' be defined at each of these points. Then there exists a unique polynomial $p(x)$ of degree less than or equal to $2n+1$ such that*

$$p(x_i) = f(x_i), \qquad p'(x_i) = f'(x_i), \quad i = 0, \dots, n$$

Proof. Let $L_k(x)$ denote the Lagrange polynomial of degree $\le n$ centered about the kth data location,

$$L_k(x) = \prod_{\substack{i=0\\ i\neq k}}^{n} \left(\frac{x - x_i}{x_k - x_i} \right).$$

Define the polynomials

$$\begin{aligned} H_k(x) &= \left(1 - 2L_k'(x_k)(x - x_k)\right) L_k^2(x) \\ \hat{H}_k(x) &= (x - x_k) L_k^2(x) \end{aligned}$$

Note that both of these polynomials have degree $\le 2n+1$. Also

$$\begin{aligned} H_k(x_i) &= \begin{cases} 1, & i = k \\ 0, & i \neq k \end{cases} \\ H_k'(x) &= -2L_k'(x_k)L_k^2(x) + 2\left(1 - 2L_k'(x_k)(x - x_k)\right) L_k(x) L_k'(x) \\ \Rightarrow H_k'(x_i) &= 0 \\ \hat{H}_k(x_i) &= 0 \\ \hat{H}_k'(x) &= L_k^2(x) + 2(x - x_k) L_k L_k'(x) \\ \Rightarrow \hat{H}_k'(x_i) &= \begin{cases} 1, & i = k \\ 0, & i \neq k \end{cases} \end{aligned}$$

These definitions and properties make it clear that H_k is associated with the function, and $\hat{H}_k$ is associated with the derivative at $x = x_k$.
Define the polynomial

$$p(x) = \sum_{k=0}^{n} f(x_k) H_k(x) + \sum_{k=0}^{n} f'(x_k) \hat{H}_k(x).$$

From the properties of H_k above, we have

$$p(x_i) = \sum_{k=0}^{n} f(x_k) H_k(x_i) + \sum_{k=0}^{n} f'(x_k) \hat{H}_k(x_i) = f(x_i)$$

$$p'(x_i) = \sum_{k=0}^{n} f(x_k) H_k'(x_i) + \sum_{k=0}^{n} f'(x_k) \hat{H}_k'(x_i) = f'(x_i)$$

for all $i = 0, \dots, n$. The polynomial p therefore interpolates all function and derivative values. Uniqueness follows from the fundamental theorem of algebra. □

As with our previous work with global interpolating polynomials, the Lagrange form is useful to establish existence, but cumbersome to use in practice. The Hermite interpolating polynomial is more efficiently computed and evaluated in Newton's form using a divided difference table. Before we start the table, we need to add another item to our understanding of divided differences:

Definition 11. Divided differences of the form $f[x_i, x_i]$ are understood to be $f[x_i, x_i] = f'(x_i)$. We justify this definition by considering the usual divided difference

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

in the limit $x_{i+1} \to x_i$. The divided difference in this limit is simply the definition of $f'(x_i)$,

$$\lim_{x_{i+1} \to x_i} f[x_i, x_{i+1}] = f[x_i, x_i] = \lim_{x_{i+1} \to x_i} \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}.$$

The divided difference table for Hermite interpolation is augmented from the previous version to account for the addition of the derivative data :

x	$f(x)$		
x_0	$f[x_0]$		
		f$[x_0, x_0]$	
x_0	$f[x_0]$		$f[x_0, x_0, x_1]$
		f$[x_0, x_1]$	
x_1	$f[x_1]$		$f[x_0, x_1, x_1]$
		f$[x_1, x_1]$	
x_1	$f[x_1]$		$f[x_1, x_1, x_2]$
		f$[x_1, x_2]$	
x_2	$f[x_2]$		$f[x_1, x_2, x_2]$
		f$[x_2, x_2]$	
x_2	$f[x_2]$		$f[x_2, x_2, x_3]$
		f$[x_2, x_3]$	
x_3	$f[x_3]$		$f[x_2, x_3, x_3]$
		f$[x_3, x_3]$	
x_3	$f[x_3]$		$f[x_3, x_3, x_4]$
		f$[x_3, x_4]$	
x_4	$f[x_4]$		$f[x_3, x_4, x_4]$
		f$[x_4, x_4]$	$\vdots$
x_4	$f[x_4]$	$\vdots$	
$\vdots$	$\vdots$		

Example 11: Hermite interpolation of $f(x) = xe^{-x}$ on $0 \leq x \leq 4$ with $n = 2$

$\Rightarrow x_0 = 0,\ x_1 = 2,\ x_2 = 4$

$f'(x) = (1 - x)e^{-x}$

x	$f(x)$					
0	0					
		$f'(0) = 1$				
0	0		$(e^{-2} - 1)/2$			
		e^{-2}		$(-3e^{-2} + 1)/4$		
2	$2e^{-2}$		$-e^{-2}$		$(e^{-4} + 4e^{-2} - 1)/16$	
		$f'(2) = -e^{-2}$		$(e^{-4} + e^{-2})/4$		$(-9e^{-4} - 4e^{-2} + 1)/64$
2	$2e^{-2}$		e^{-4}		$-e^{-4}/2$	
		$2e^{-4} - e^{-2}$		$(-7e^{-4} + e^{-2})/4$		
4	$4e^{-4}$		$(-5e^{-4} + e^{-2})/2$			
		$f'(4) = -3e^{-4}$				
4	$4e^{-4}$					

To write the polynomial, we introduce the sequence $\{z_k\}$ where $z_k = x_{\lfloor k/2 \rfloor}$, $k = 0, \ldots, 2n+1$:

$$\{z_k\} = \{x_0, x_0, x_1, x_1, x_2, x_2, \ldots, x_{n-1}x_{n-1}, x_n, x_n\}$$

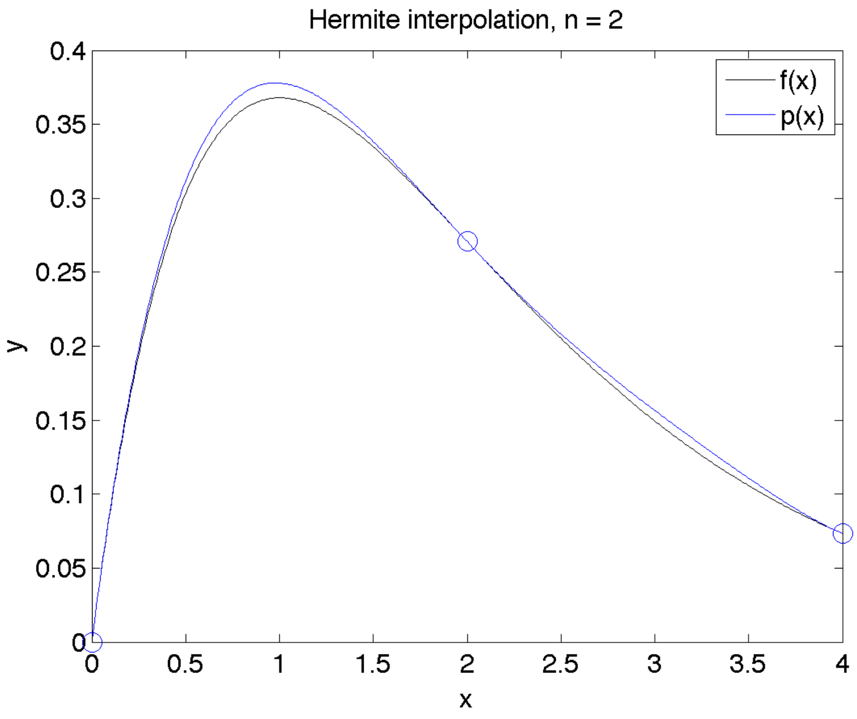
$$p(x) = \sum_{k=0}^{2n+1} f[z_0, z_1, \ldots, z_k] \left(\prod_{i=0}^{k-1} (x - z_i) \right)$$

The terms of this sum are

$$\begin{array}{lll}
k = 0: & f[z_0] & = f(x_0) \\
k = 1: & f[z_0, z_1](x - z_0) & = f'(x_0)(x - x_0) \\
k = 2: & f[z_0, z_1, z_2](x - z_0)(x - z_1) & = f[x_0, x_0, x_1](x - x_0)^2 \\
k = 3: & f[z_0, z_1, z_2, z_3](x - z_0)(x - z_1)(x - z_2) & = f[x_0, x_0, x_1, x_1](x - x_0)^2(x - x_1) \\
k = 4: & f[z_0, \ldots, z_4](x - z_0)(x - z_1)(x - z_2)(x - z_3) & = f[x_0, x_0, x_1, x_1, x_2](x - x_0)^2(x - x_1)^2 \\
k = 5: & f[z_0, \ldots, z_5](x - z_0)(x - z_1)(x - z_2)(x - z_3)(x - z_4) & = f[x_0, x_0, x_1, x_1, x_2, x_2](x - x_0)^2(x - x_1)^2(x - x_2)
\end{array}$$

Note that the coefficients are conveniently the top entry in each column of the divided difference table. Plugging those values in, along with $x_0 = 0,\ x_1 = 2,\ x_2 = 4$, we find

$$p(x) = x + \left(\frac{e^{-2} - 1}{2} \right) x^2 - \left(\frac{3e^{-2} - 1}{4} \right) x^2(x-2) + \left(\frac{e^{-4} + 4e^{-2} - 1}{16} \right) x^2(x-2)^2 - \left(\frac{9e^{-4} + 4e^{-2} - 1}{64} \right) x^2(x-2)^2(x-4)$$



△

Notes:

- Hermite interpolation can be very accurate, but the $2n + 1$ data points can be expensive to use if n is large.
- Can we use this idea more efficiently?

4.1 Piecewise cubic Hermite interpolation

Definition 12. The <u>Hermite cubic interpolant</u> of $f(x)$ on the interval $a \leq x \leq b$ with n subintervals

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$$

is a function s that satisfies

1. $s(x)$ coincides with a cubic polynomial $s_i(x)$ on each subinterval
2. s interpolates both $f(x)$ and $f'(x)$ at each data point
3. $s(x)$ and $s'(x)$ are continuous on $a \leq x \leq b$.

Note that there is no continuity condition imposed on $s''(x)$, hence cubic Hermite splines are not as smooth as natural cubic splines, but computing them requires less work.

Definition 13. The <u>shape functions</u> $\phi(\xi)$ and $\psi(\xi)$ associated with cubic Hermite interpolation are

$$\phi(\xi) = (1 + 2\xi)(1 - \xi)^2$$
$$\psi(\xi) = \xi(1 - \xi)^2$$

where the variable ξ is defined on each subinterval as $\xi_j = \dfrac{x - x_j}{x_{j+1} - x_j}$.

Definition 14. Given $f(x_i)$ and $f'(x_i)$ for $i = 0, \ldots, n$, the cubic Hermite interpolant is defined on each subinterval of $a \le x \le b$ to be

$$s_j(x) = f(x_{j+1}) + \phi(\xi_j)\left(f(x_j) - f(x_{j+1})\right) + (x_{j+1} - x_j)\left(\psi(\xi_j)f'(x_j) - \psi(1-\xi_j)f'(x_{j+1})\right)$$

so that

$$s(x) = s_j(x) \text{ for } x_j \le x \le x_{j+1}, \qquad j = 0, \ldots, n-1$$

Notes:

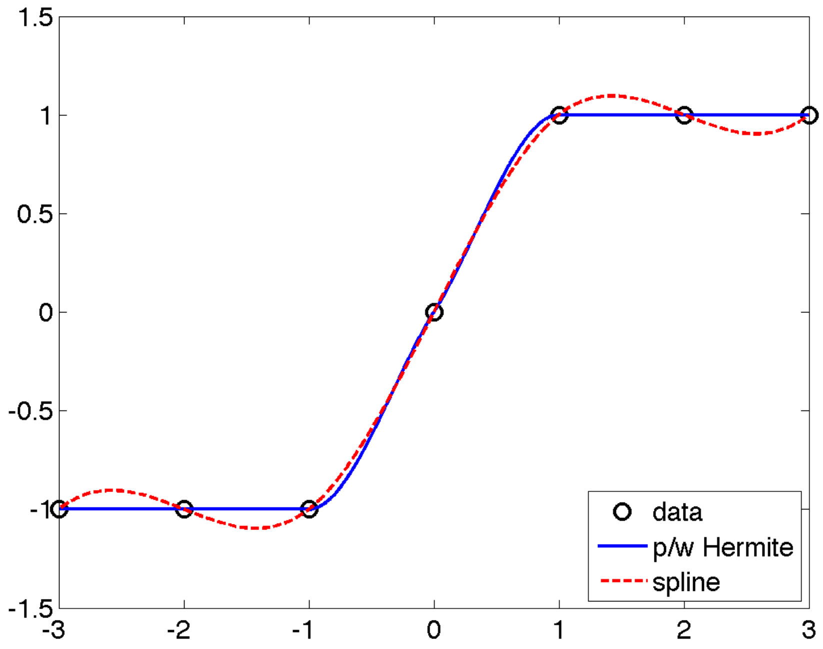
- Unlike cubic splines, no linear system of equations has to be solved
- Only three function evaluations : $\phi(\xi_j), \psi(\xi_j)$, and $\psi(1-\xi_j)$ are required to evaluate $s_j(x)$.
- Like any local method, still have to locate the interval j that contains the evaluation point to begin
- If the function itself is not differentiable, it is better to use the less-smooth cubic Hermite interpolation than cubic splines
- If you know the function you're interpolating is smooth, splines will provide better accuracy than cubic Hermite interpolation

Question 4: It's great that Hermite interpolation is accurate and that piecewise cubic Hermite interpolation is so fast, but what if we don't have data about $f'(x)$? Can we still use these ideas?

Example 12: Compare Matlab's piecewise cubic spline interpolation to its piecewise cubic Hermite interpolation for the data given in the following table.

x	y
-3	-1
-2	-1
-1	-1
0	0
1	1
2	1
3	1

△



```
clear;
%% Cubic Hermite interpolation vs. cubic spline interpolation

x = -3:3;                          % data locations
y = [-1,-1,-1, 0, 1,1,1];          % data values

xe = -3:0.01:3;                    % evaluation domain

sHermite = pchip(x,y,xe);          % cubic Hermite interpolation
sSpline = spline(x,y,xe);          % cubic spline interpolation
% Note: read Matlab's documentation to learn how the derivatives are
% estimated for the Hermite interpolation.  It's more involved than a
% simple finite difference scheme -- they use limiters (Math 572).

figure(1);clf;
plot(x,y,'ko',xe,sHermite,'b-',xe,sSpline,'r--','MarkerSize',10,'LineWidth',2);
set(gca,'FontSize',16);
legend('data','p/w␣Hermite','spline','Location','SouthEast');

saveas(1,'splineVsHermite.png')
```