THE UNIVERSITY OF UTAH

SCHOOL OF COMPUTING

CS 3200
Introduction to Scientific Computing

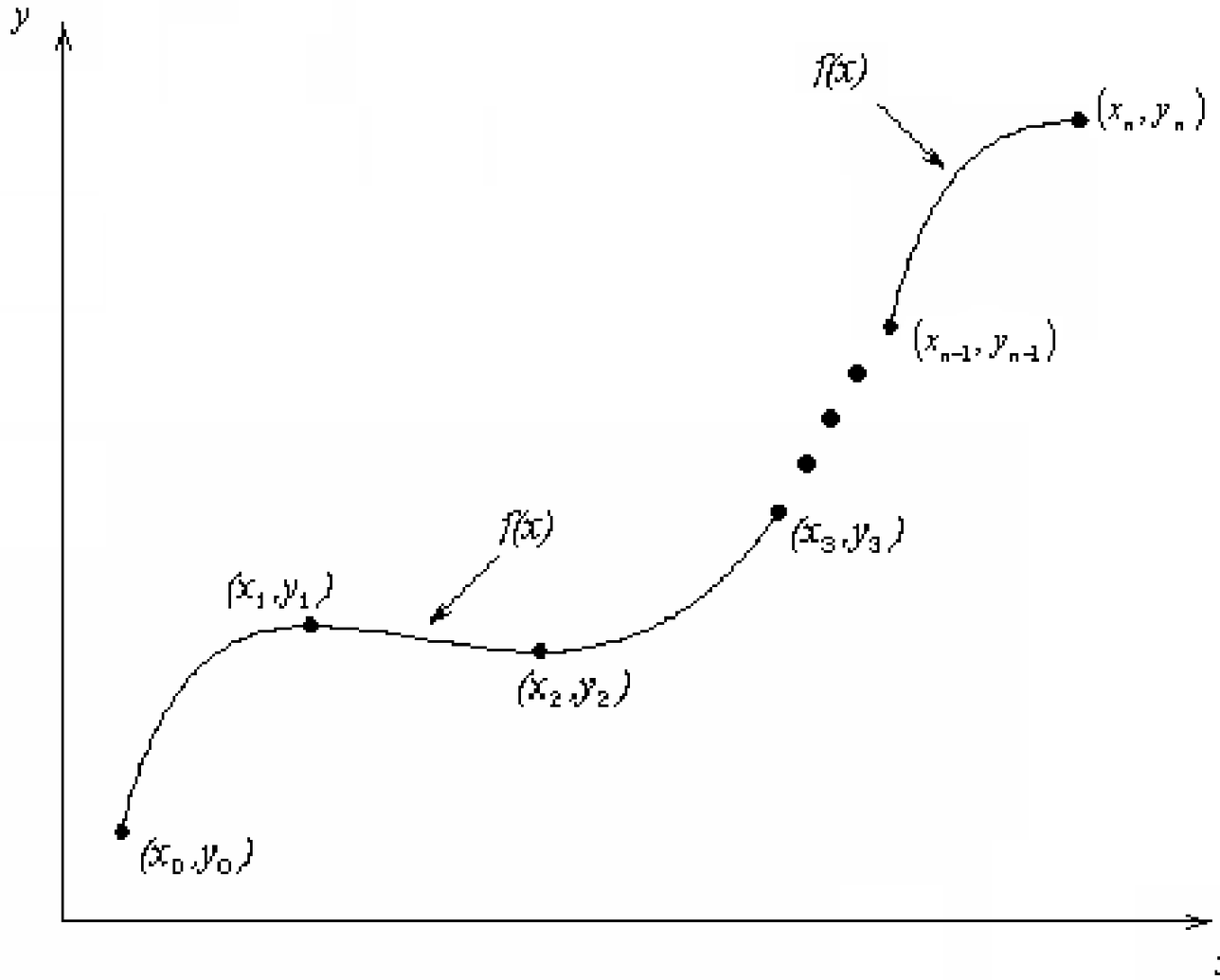
Instructor: Martin Berzins

Topic: Interpolation

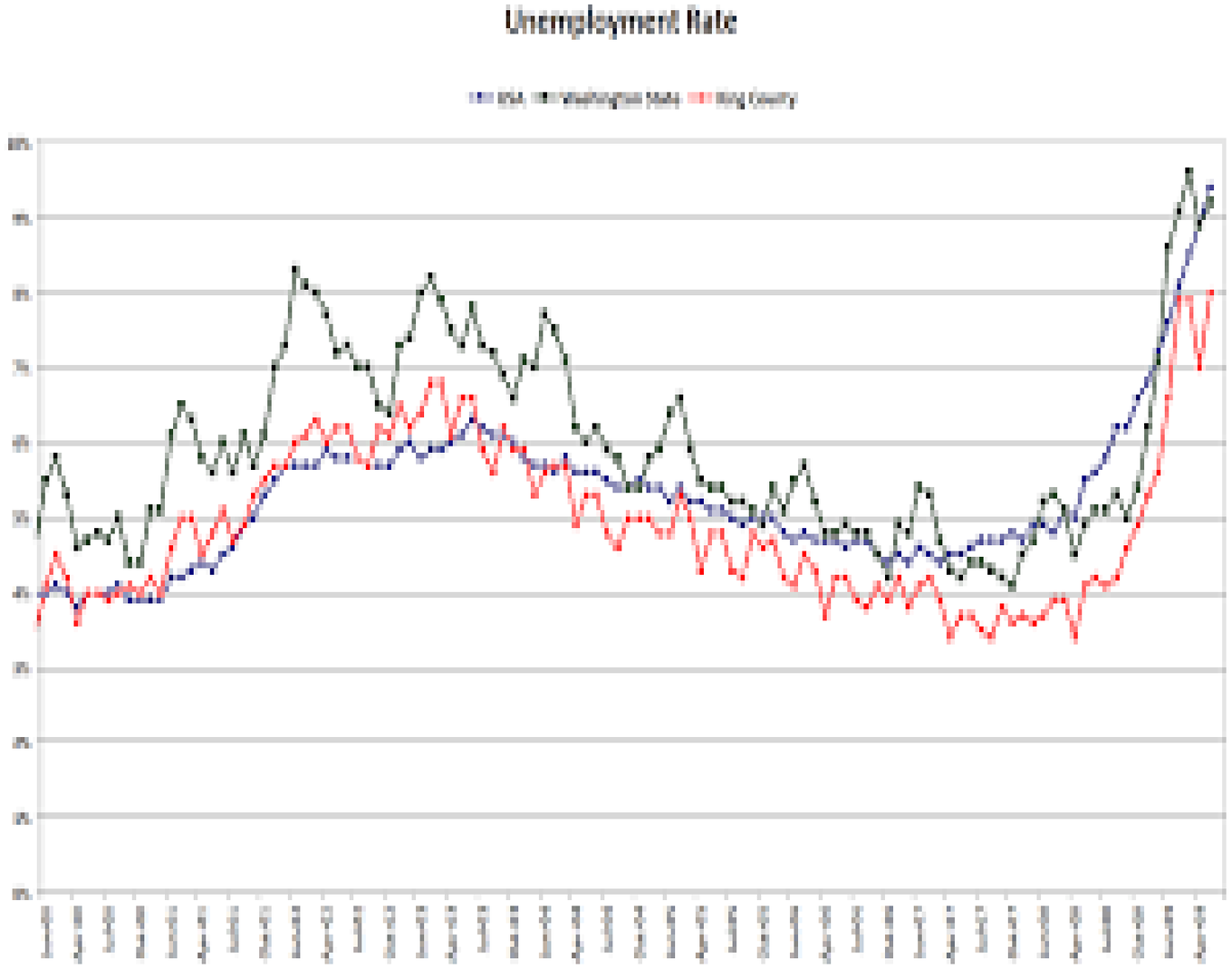
What is Interpolation?

Given $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$, find the value of y (or its gradient, derivative)

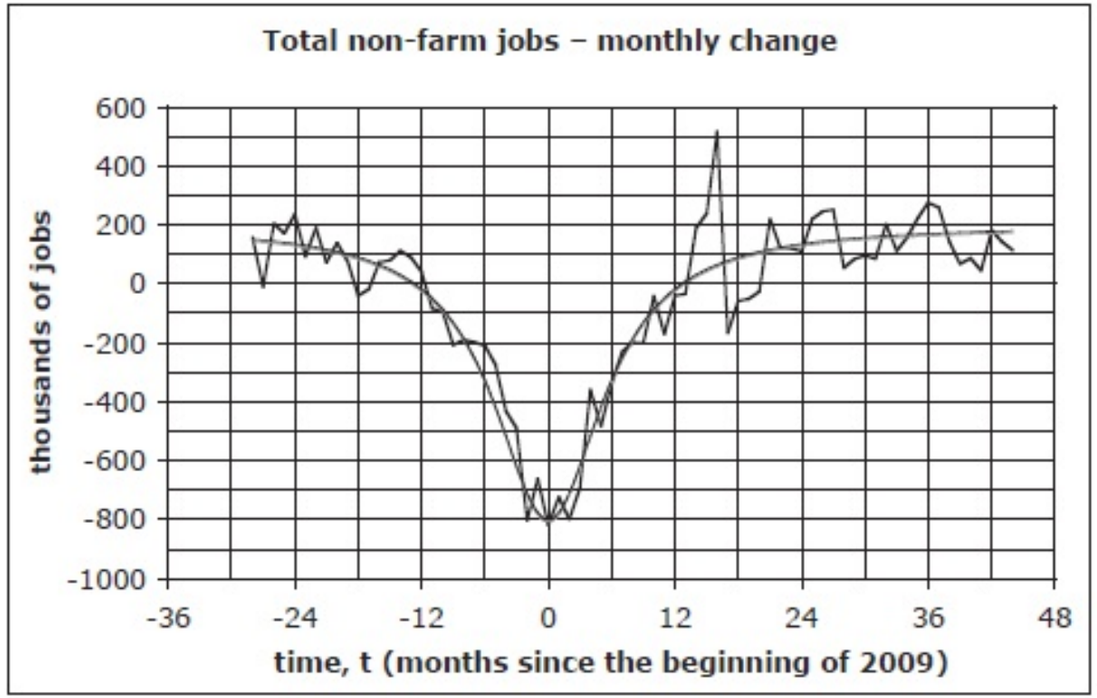
at a value of x that is not explicitly given



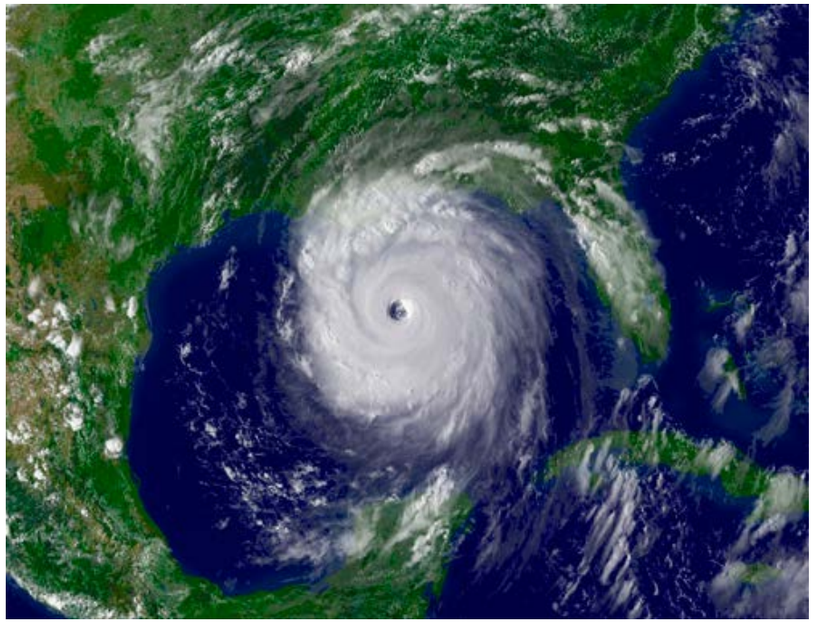
Real Data is often not smooth



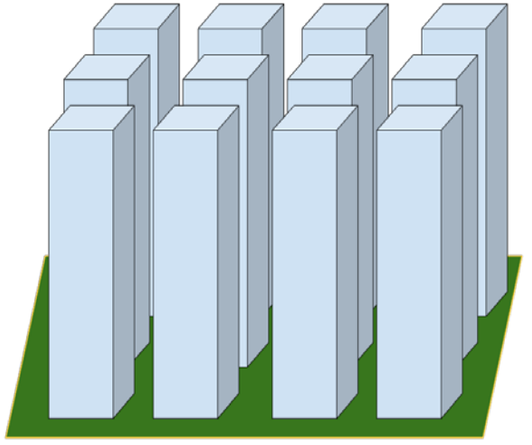
However we do use smooth fine-scale approximations at least to the level of floating point arithmetic



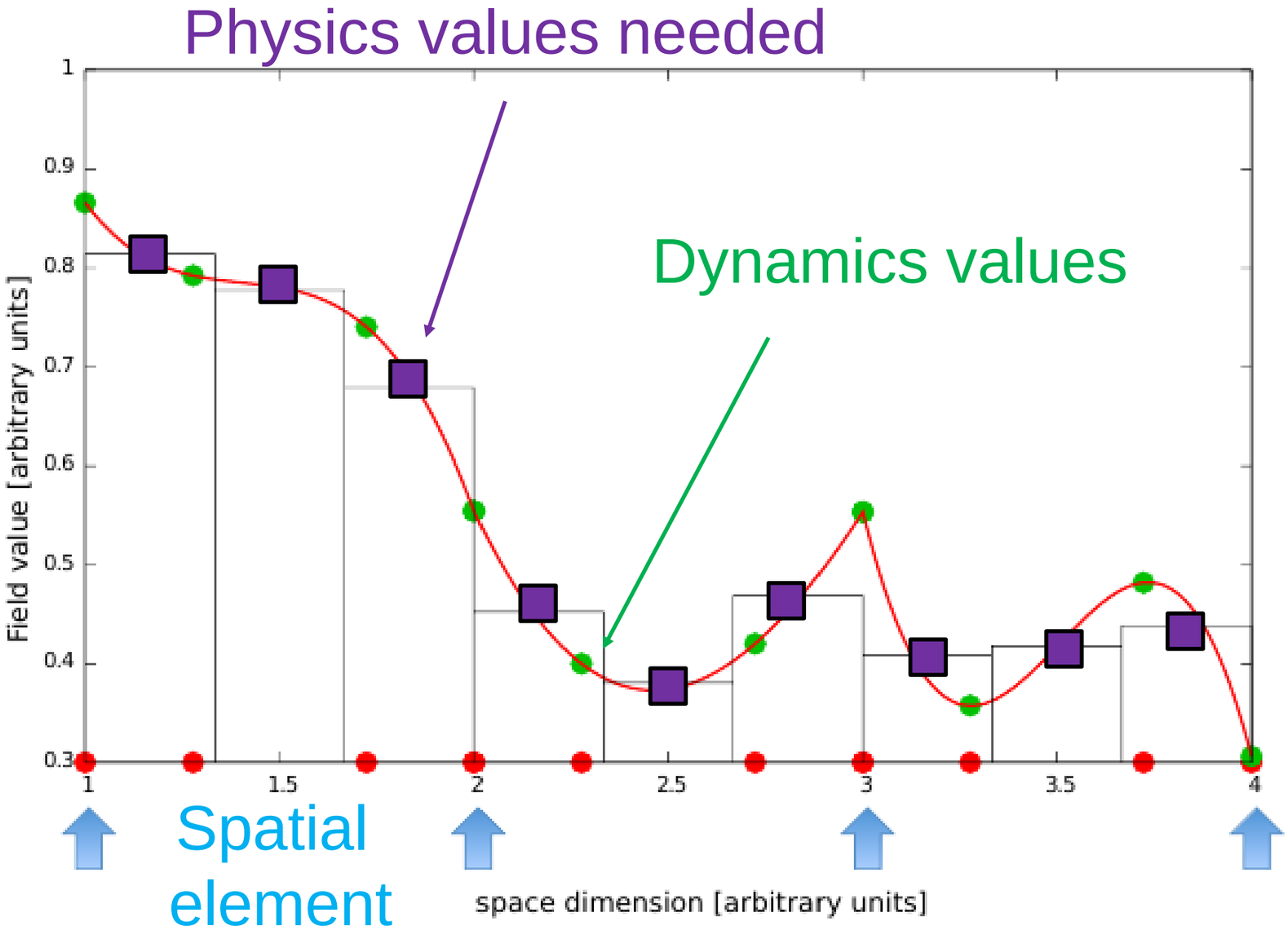
Interpolation in Weather Codes



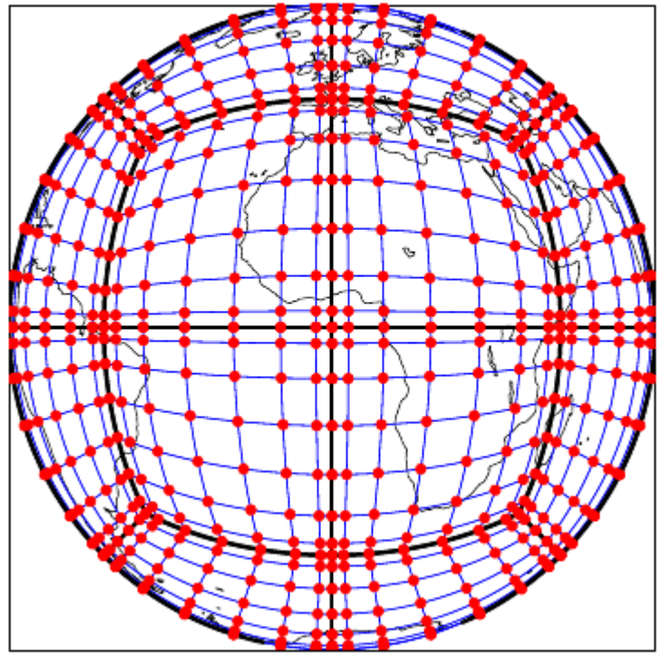
- Recent codes like Neptune split into dynamics and physics part
- Physics deals with rain snow grauple
- Physics considers vertical columns
- Physics routines need to be evenly spaced.
- Dynamics routines used a different mesh
- Need to map physical quantities between meshes
- Density velocities in x,y, and z and temperature are mapped back and forth



Weather Interpolation Problem



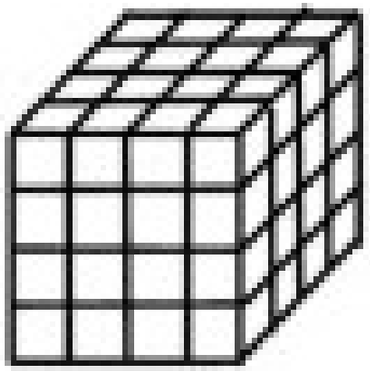
ne2np8 quadrature points on cubed-sphere

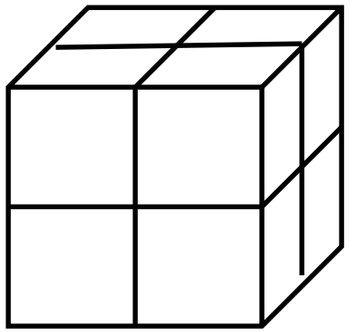


Need to use green values To find purple values

Algorithm-Based Fault Tolerance

- The largest machines today hve minor faults about once a day and may have a node crash once a week.
- Approaches to deal with this may involve hardware, system software or algorithmic resilience.
- In the last case we may store a coarse copy of a mesh patch on another node. In three dimensions a mesh patch with ½ the variables in each dimension require only 1/8 of the storage.
- In order to rebuild the fine mesh patch from the coarse one we need to use interpolation that gives physically meaningful values.
- We will explore this is the in class-activity on Monday.





Interpolation and approximation

1 Often interpolation of a function y(x) is done using a polynomial:

$$y_I(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

As there are n+1 degrees of freedom in there is one and only one n^{th}-order polynomial that fits n+1 points.

2 rational functions used in design

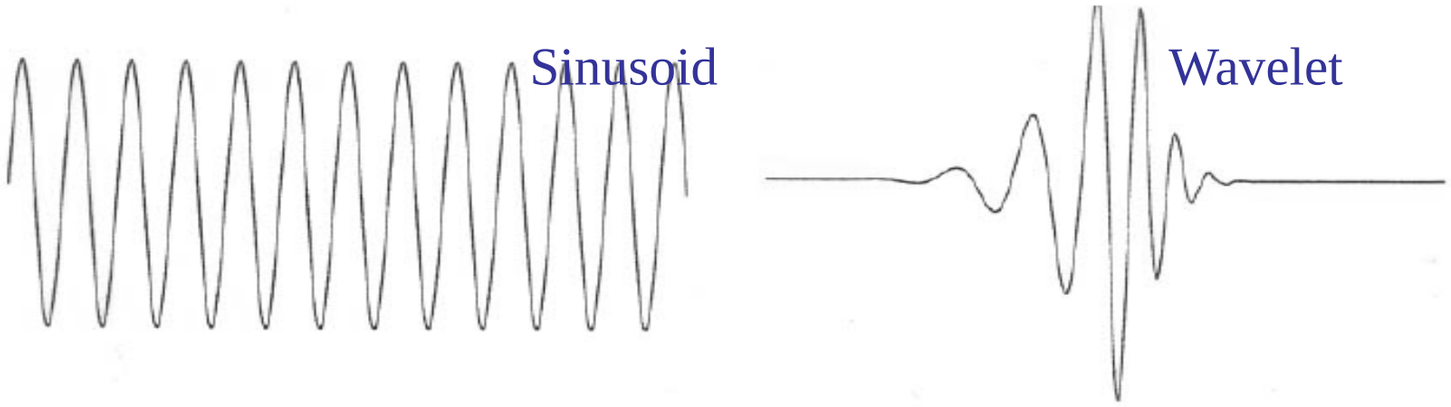
$$y_I(x) = \frac{a + bx + cx^2 + dx^3 + ex^4}{f + gx + hx^2 + kx^3 + ex^4}$$

3 Trigonometric interpolants, e.g Fourier series sometimes used in climate codes

$$y_I(x) = a_{-n} e^{-inx} + \ldots + a_0 + \cdots + a_n e^{inx}$$

4. An alternate approach to Fourier methods for image processing and even simulation - wavelets?

A function that “waves” above and below the x-axis with the following properties:
Varying frequency
Limited duration
Zero average value
This is in contrast to sinusoids, used by FT, which have infinite duration and constant frequency.



Why Polynomials?

- **Weierstrass Approximation Theorem (1885).** Suppose f is a continuous real-valued function defined on the real interval [a, b]. For every $\varepsilon > 0$, there exists a polynomial $p(x)$ such that for all x in [a, b], we have $|f(x) - p(x)| < \varepsilon$, or equivalently, the infinity or
- supremum norm i.e. for all x that the max value $|f(x) - p(x)| < \varepsilon$.
- More formally $\| f(x) - p(x) \|_\infty < \varepsilon$
- What is problematic about this theory?

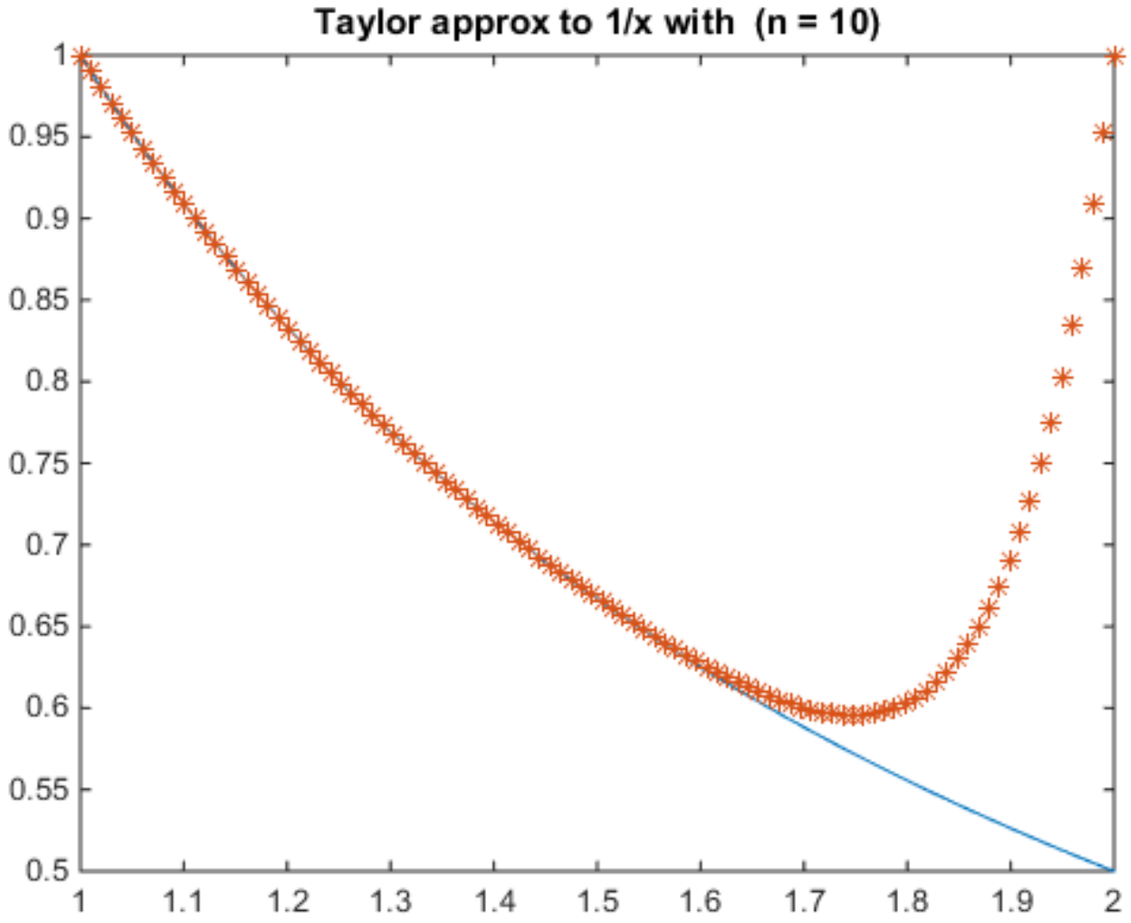
Why Polynomials?

- **Weierstrass Approximation Theorem (1885).** Suppose f is a continuous real-valued function defined on the real interval [a, b]. For every $\varepsilon > 0$, there exists a polynomial $p(x)$ such that for all x in [a, b], we have $|f(x) - p(x)| < \varepsilon$, or equivalently, the infinity or
- supremum norm i.e. for all x that the max value $|f(x) - p(x)| < \varepsilon$.
- More formally $\| f(x) - p(x) \|_\infty < \varepsilon$
- What is problematic about this theory?

Bernstein proved a constructive version of this theory in 1912

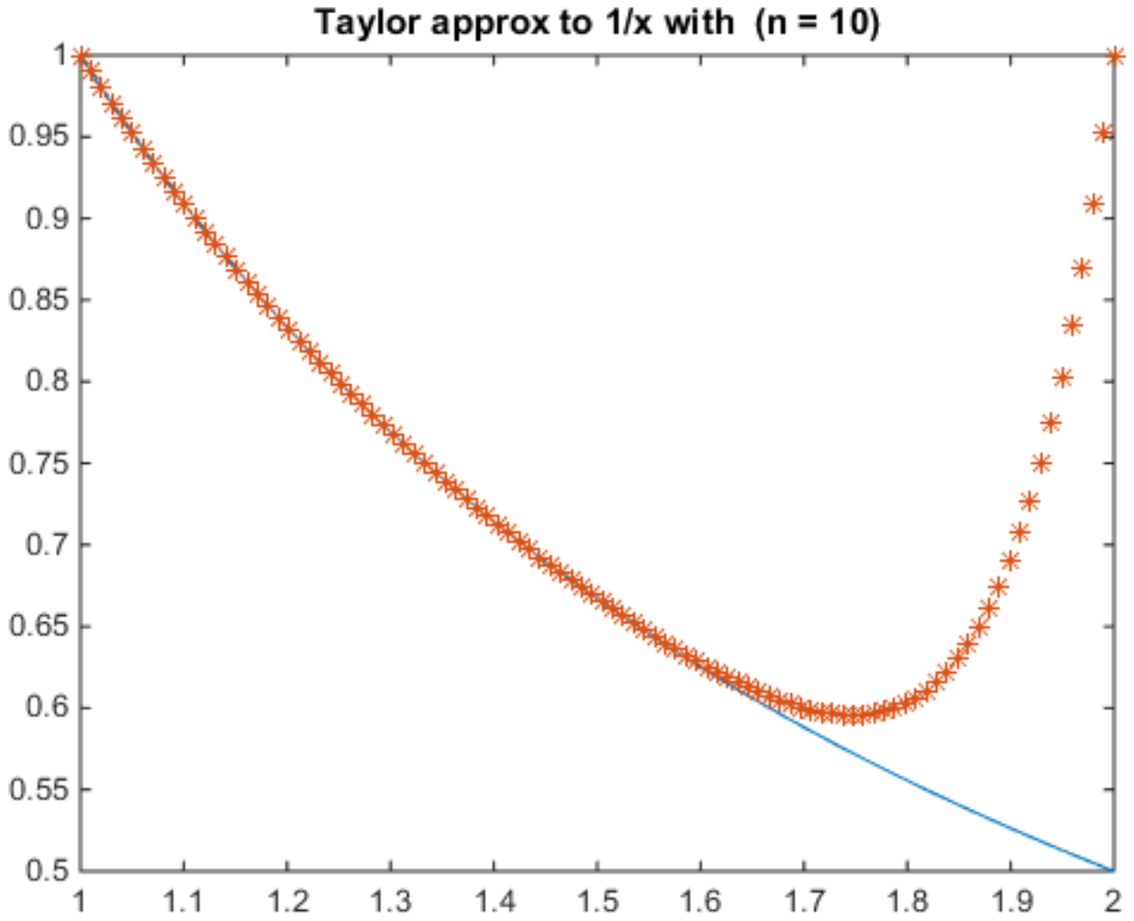
Using Taylors Series Directly

- Example – consider approximating the function 1/x close to x = 1 using Taylors series with n = 10.
- Even if we make n much larger there are still problems close to 2
- The answer is completely wrong
- Why?



Using Taylors Series Directly

- Example – consider approximating the function 1/x close to x = 1 using Taylors series with n = 10.
- Even if we make n much larger there are still problems close to 2
- The answer is completely wrong
- Why?



We saw that close to x=2 the terms in the Taylors series involve (x-1)^p and very many terms are needed for convergence

Coefficients of an Interpolating Polynomial

- Since $n+1$ data points are required to determine $n+1$ coefficients, simultaneous linear systems of equations can be used to calculate "a"s:

$$y(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 \cdots + a_n x_0^n$$

$$y(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 \cdots + a_n x_1^n$$

$$\vdots$$

$$y(x_n) = a_0 + a_1 x_n + a_2 x_n^2 \cdots + a_n x_n^n$$

where "x"s are the knowns and "a"s are the unknowns

Solving A Linear System

We can represent this system of equations as

$$\begin{pmatrix} & & & \\ 1, x_0, x_0^2, x_0^3, ..., & x_0^{n-1} & x_0^n \\ 1, x_1, x_1^2, x_1^3, ... & x_1^{n-1} & x_1^n \\ \vdots & \vdots & \vdots \\ 1, x_n, x_n^2, x_n^3, ... & x_n^{n-1} & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}$$

and our solution will be the polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + ... + a_n x^n$$

$O(n^3)$ operations to solve linear system

Example

Polynomial of degree 2 interpolating three data points

$(-2,-27),(0,-1),(1,0)$

Linear system is

$$\begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{pmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = y$$

In this case

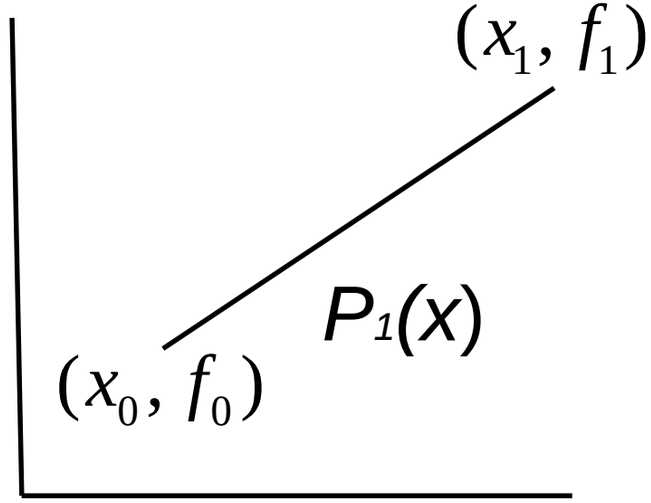
$$\begin{pmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -27 \\ -1 \\ 0 \end{bmatrix}$$

With solution

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 5 \\ 4 \end{bmatrix}, \text{giving polynomial} \quad -1+5x+4x^2$$

Linear Interpolation

- Given f($\boldsymbol{x_0}$) = $\mathbf{f_0}$ and f($\boldsymbol{x_1}$) = $\mathbf{f_1}$
  - Approximate the unknown y(x) with $p_1(x)$
  - Construct two linear interpolants each of which is zero at one point and one at the other point
  - $L_1(x) = \frac{(x-x_0)}{(x_1-x_0)}$, $L_0(x) = \frac{(x-x_1)}{(x_0-x_1)}$
  - Then $P_1(x) = L_0 f_0 + L_1 f_1$



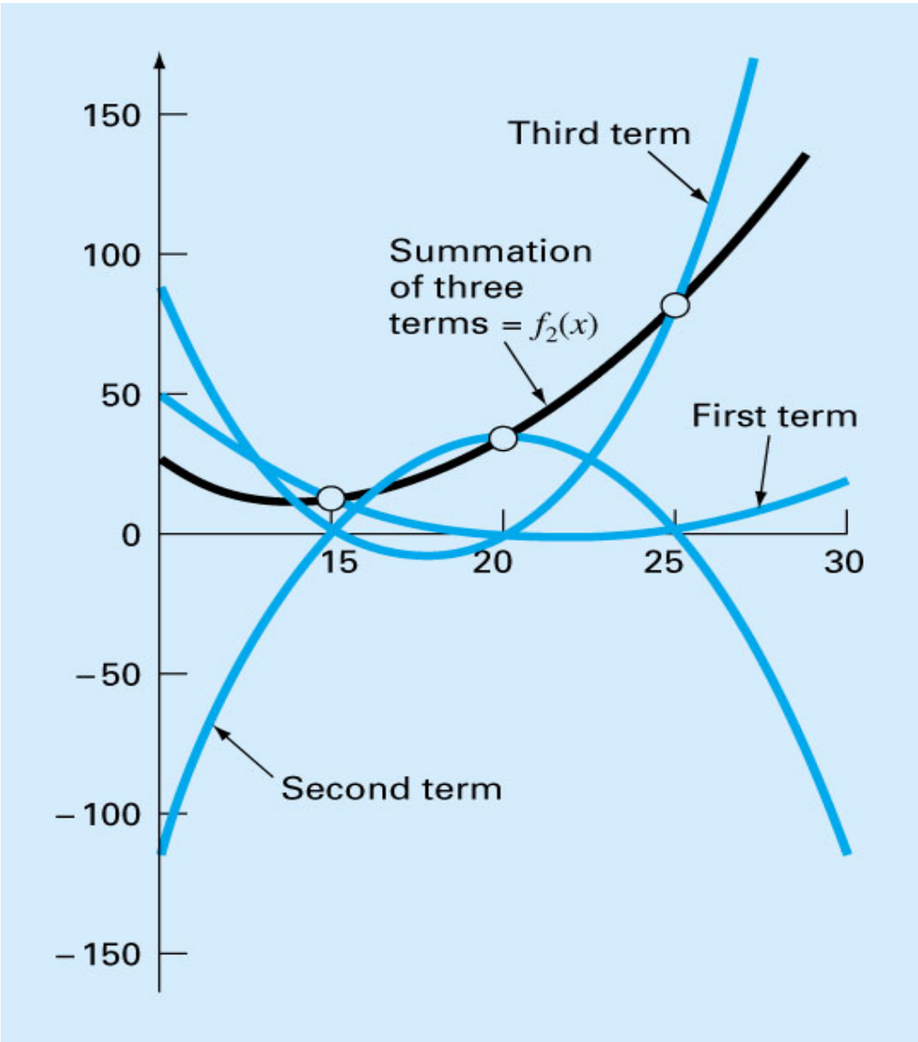
Linear and Quadratic Lagrange Interpolating Polynomials

The linear case

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

The quadratic case is similar

$$f_2(x) = \underbrace{\frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}}_{L_0} f(x_0) + \underbrace{\frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}}_{L_1} f(x_1) + \underbrace{\frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}}_{L_2} f(x_2)$$



Lagrange Interpolation

- Given multiple points $(x_i, f(x_i))$
  - Calculate a polynomial interpolant in the same way

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_0)(x - x_1)....(x - x_{i-1})(x - x_{i+1})...(x - x_n)}{(x_i - x_0)(x_i - x_1)....(x_i - x_{i-1})(x_i - x_{i+1})...(x_i - x_n)}$$

Error of Polynomial Interpolation

For an n^{th}-order interpolating polynomial, the error is given by the $(n+1)^{th}$ derivative of the unknown function

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

ξ is located in some interval containing the unknown and the data: $[x, x_0, x_1, ..., x_n]$

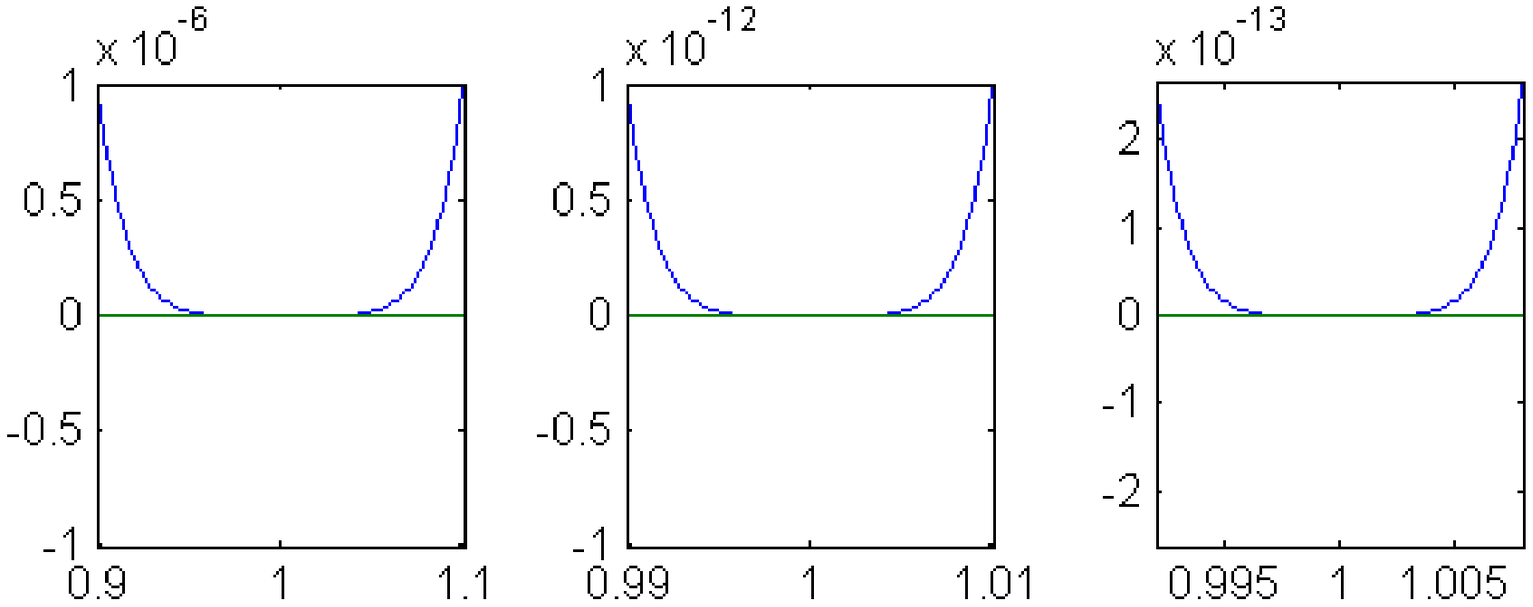
Summing a Polynomial

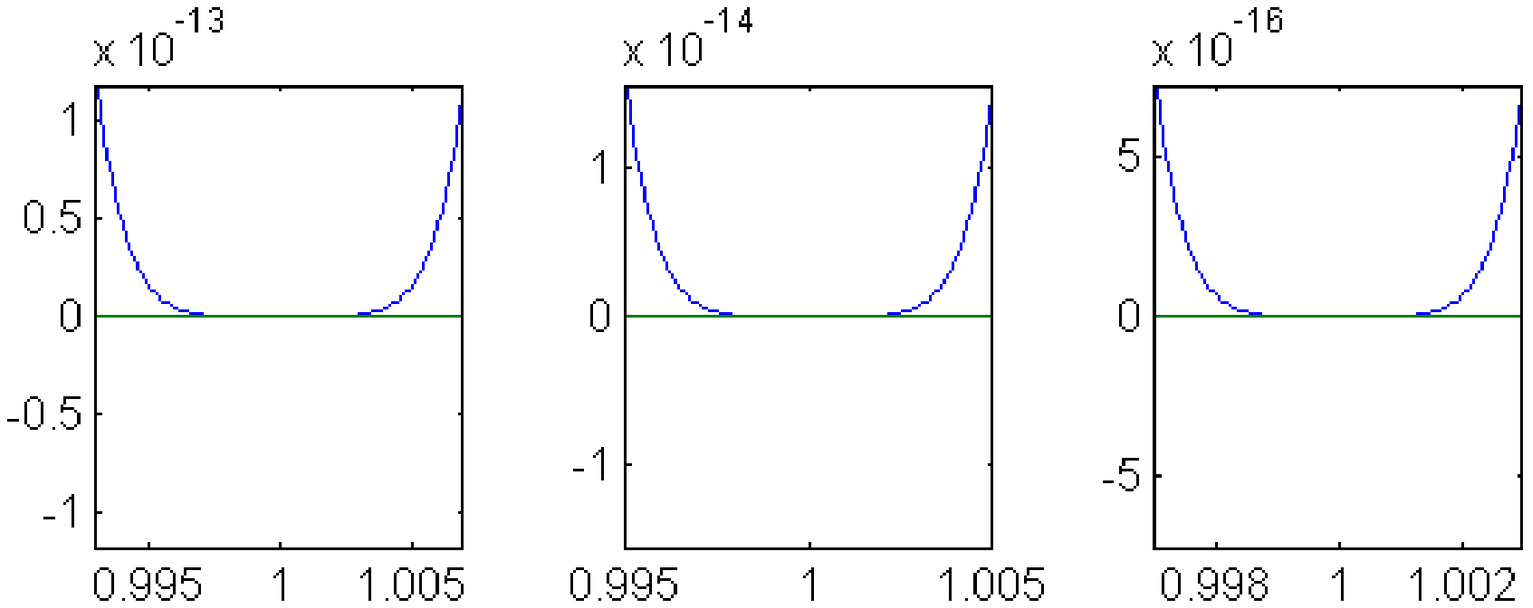
Method A calculate $(x-1)^6$ directly

Method B consider the expanded out form

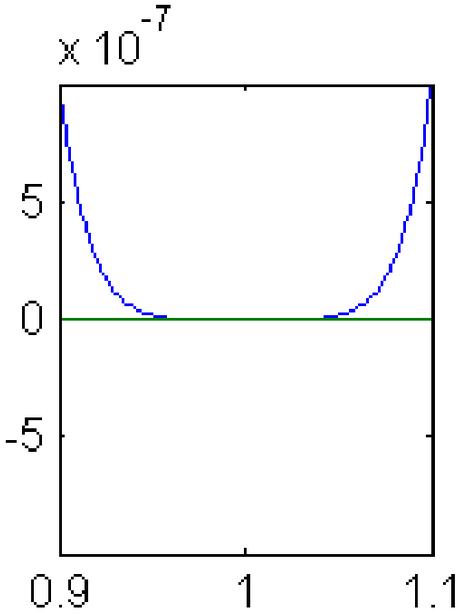
$(x-1)^6 = 1 -6x+15x^2-20x^3+15x^4 -6x^5 +x^6$

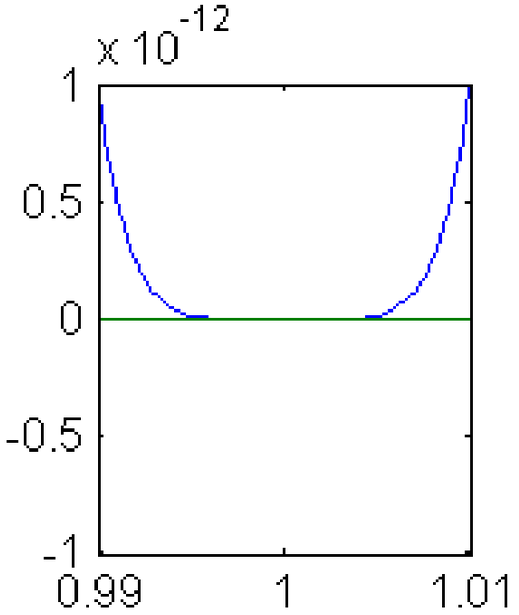
Why? Well we may not always know what the factors are of a polynomial.

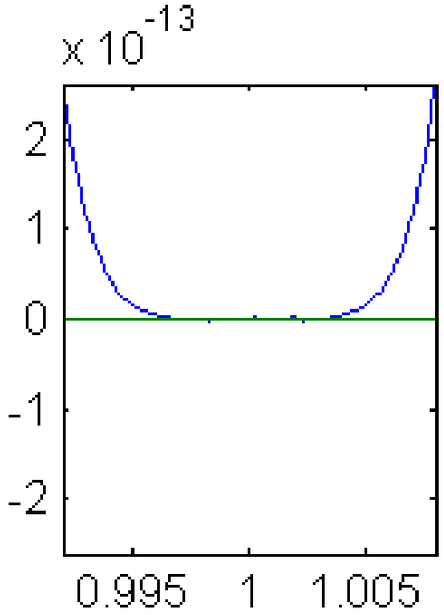

x 10^-6
1
0.5
0
-0.5
-1
0.9
1
1.1
x 10^-12
1
0.5
0
-0.5
-1
0.99
1
1.01
x 10^-13
2
1
0
-1
-2
0.995
1
1.005


x 10^-13
1
0.5
0
-0.5
-1
0.995
1
1.005
x 10^-14
1
0
-1
0.995
1
1.005
x 10^-16
5
0
-5
0.998
1
1.002

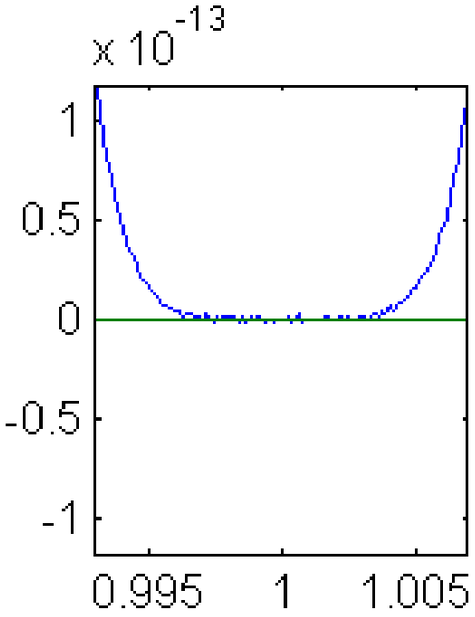
Method A
(x-1)^6

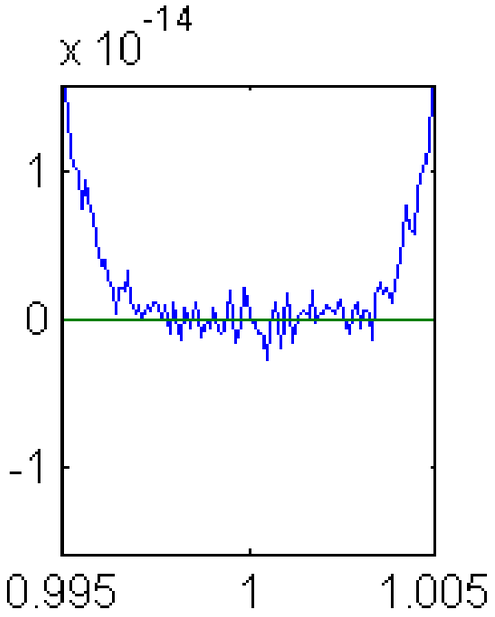


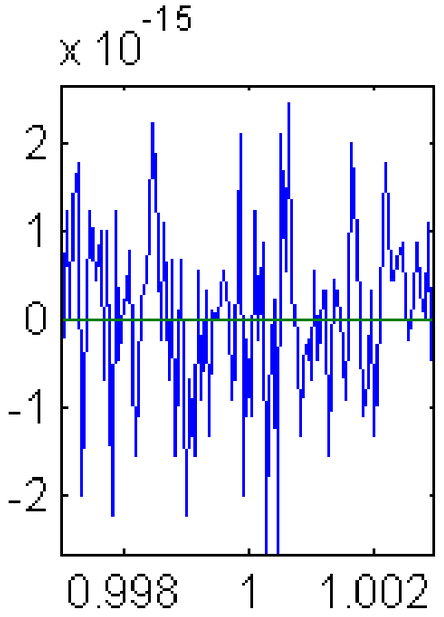




Method B
Expansion
of (x-1)^6
into its polynomial terms







Rounding Errors
give not one zero
but many close to1

General Application of Polynomials and Interpolation

- Decide on a set of functions (polynomials) that you will use to present the solution to the problem and a set of points at which these polynomials are defined
- Often these will be polynomials defined in terms of unknown values at spatial mesh points. $y(x_i)$
- Substitute these into the computational model
- Solve a set of equations, Linear nonlinear, time-dependent for these solution values and hence the polynomial.
- Use interpolation based on this polynomial to recover the solution everywhere

Application of Lagrange Interpolation

- Consider a tightly stretched string between two points x=a and x =b.

  The equation for the deformation of the string is given by

$$\frac{d^2y}{dx^2} - c = 0, y(a) = 0, y(b) = 0$$

This is known as a boundary value differential equation problem

We approximate the unknown solution by a Lagrange polynomial

$$y(x) \approx \sum_{i=0}^{n} L_i(x) y(x_i), x_i = a + i(b-a)/n$$

And then set up a system of equations for the unknown values $y(x_i)$

Substitute and then evaluate the first equation at the points x_j

$$\left.\frac{d^2y(x)}{dx^2}\right|_{x=x_j} - c = 0,\ j = 1,\ldots,n-1,\ y(x_0) = 0,\ y(x_n) = 0$$

or noting the zero solution values at the boundary

$$\left.\sum_{i=1}^{n-1}\frac{d^2L_i(x)}{dx^2}\right|_{x=x_i} y(x_i) = c,\ j = 1,\ldots,n-1,\ y(x_0) = 0,\ y(x_n) = 0$$

if $a_{j,i} = \left. \frac{d^2 L_i(x)}{dx^2} \right|_{x=x_j}$ then we have the system of equations

$$\begin{pmatrix} a_{11} & \cdots & a_{1n-1} \\ \vdots & \ddots & \vdots \\ a_{n-11} & \cdots & a_{n-1n-1} \end{pmatrix} \begin{pmatrix} y(x_1) \\ \vdots \\ y(x_{n-1}) \end{pmatrix} = \begin{pmatrix} c \\ \vdots \\ c \end{pmatrix}$$

for the vector of solution values $\begin{pmatrix} y(x_1) \\ \vdots \\ y(x_{n-1}) \end{pmatrix}$

The solution at any point in [a,b] may be found by interpolating from the Lagrange polynomial.

Example Using a Quadratic Polynomial

$$y(x) = L_2(x)\,y(x_i)$$

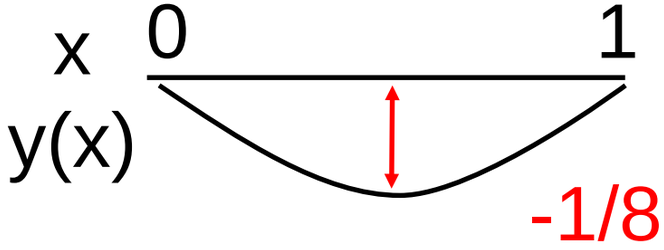
$$x_i = a + i(b-a)/2, a = 0, b = 1, c = 1$$

$$y(x) = \frac{(x-0)(x-1)}{-1/4}\,y(1/2)$$

$$\frac{d^2 L_2(x)}{dx^2} = -8 \Rightarrow -8y(1/2) = c = 1 \Rightarrow y(1/2) = -1/8$$

Hence the solution is

$$y(x) = \frac{(x-0)(x-1)}{-1/4}\left(-\frac{1}{8}\right) = \frac{x(x-1)}{2}$$



Limitation of interpolating polynomials

Runge's phenomenon.

When approximating the function f(x) on [a,b] by an interpolating polynomial, an error does not necessary decrease as increase the degree of polynomial. The interpolation oscillates to the end of the interval,

$$f(x) = \frac{1}{1 + 25x^2}, \text{ defined on an interval } [-1, 1],$$

$$\lim_{n \to \infty} \left(\max_{-1 \le x \le 1} |f(x) - P_n(x)| \right) = \infty.$$

https://en.wikipedia.org/wiki/Runge%27s_phenomenon

Example Code

```
% Script File: RungeEg
% For n=10:13, interpolants of f(x) = 1/(1+25x^2) on [-1,1]  are of plotted.
  close all
  x = linspace(-1,1,100);
  y = ones(100,1)./(1 + 25*x.^2);
  iplot = 0;
  for n=10:13
  xEqual = linspace(-1,1,n)';
     yEqual = ones(size(xEqual))./(1+25*xEqual.^2);;
     cEqual=              InterpN(xEqual,yEqual);
     pvalsEqual =       HornerN(cEqual,xEqual,x);
     iplot = iplot+1;
     subplot(2,2,iplot)
     plot(x,y,x,pvalsEqual,xEqual,yEqual,'*')
     title(sprintf('Equal Spacing (n = %2.0f)',n))
  end
```

Horner's Scheme for evaluating a Polynomial

Evaluating a polynomial

$$y_I(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

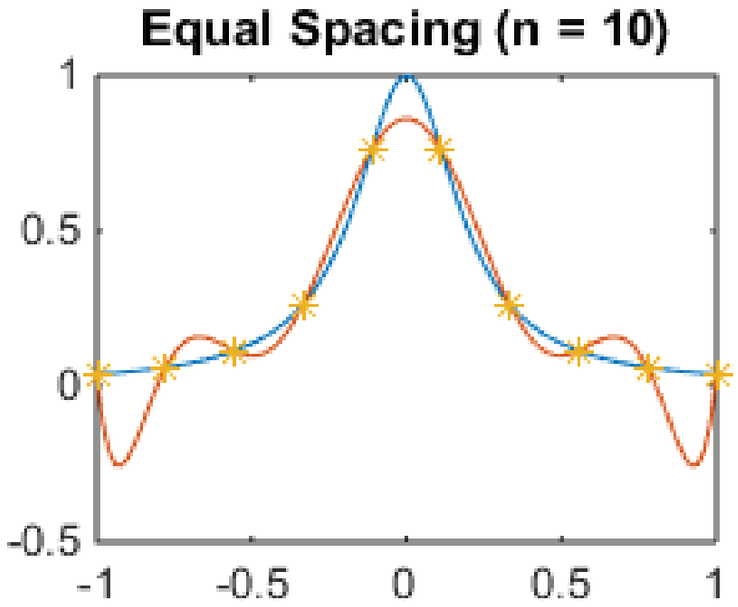
Optimal algorithm

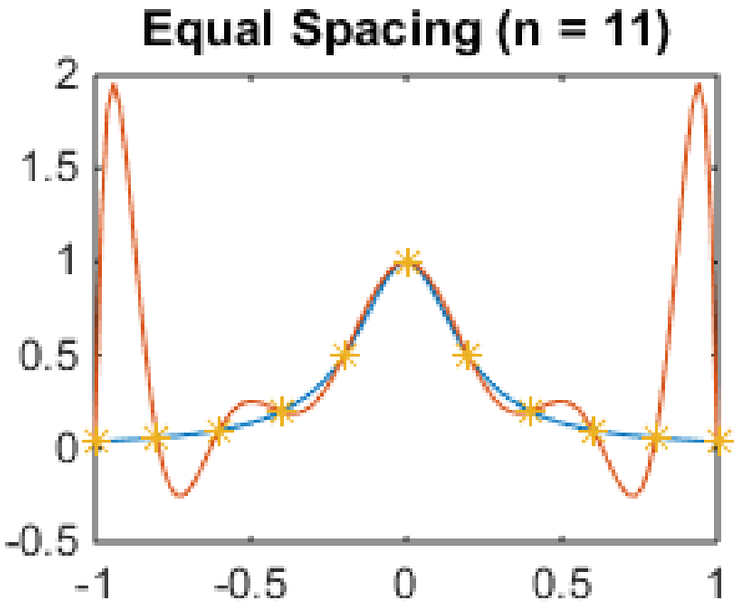
Sum = a_n

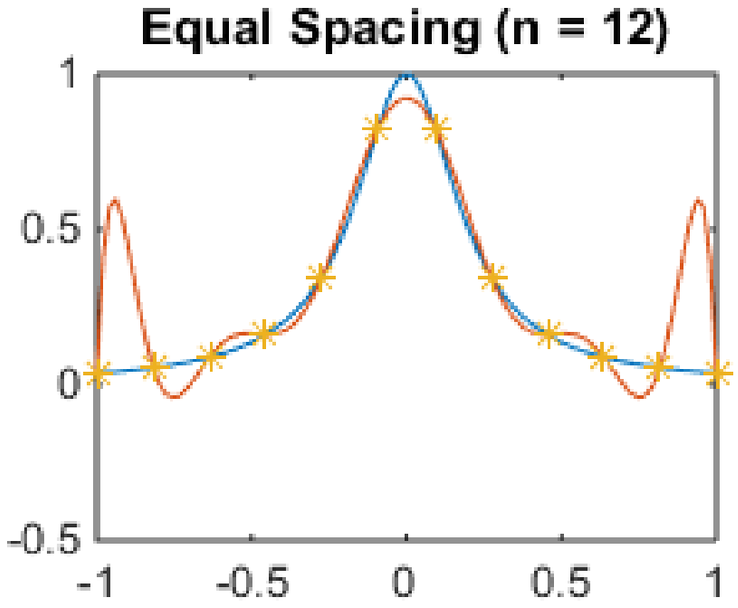
For i = n-1 to 0

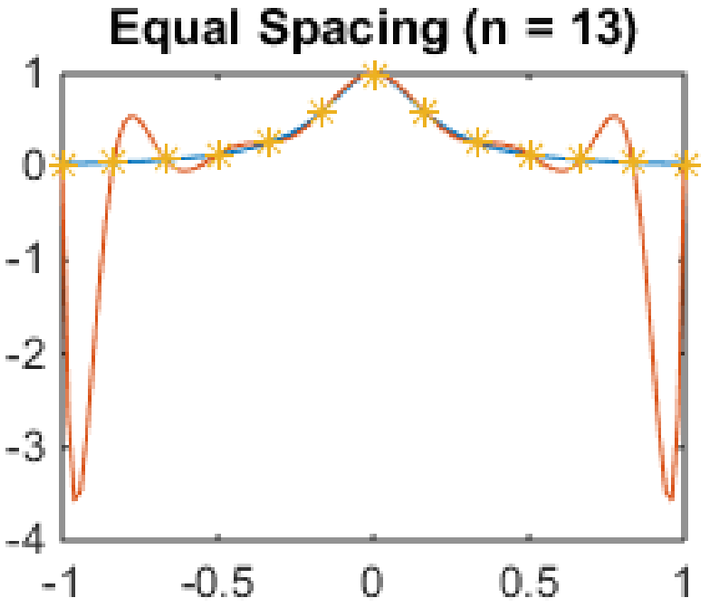
sum = x * sum + a_i

Runge’s Example Equally spaced polynomial interpolation points true solution blue polynomial approximation



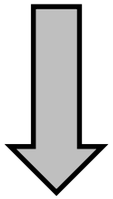






Why?

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



Grows very quickly for evenly spaced points

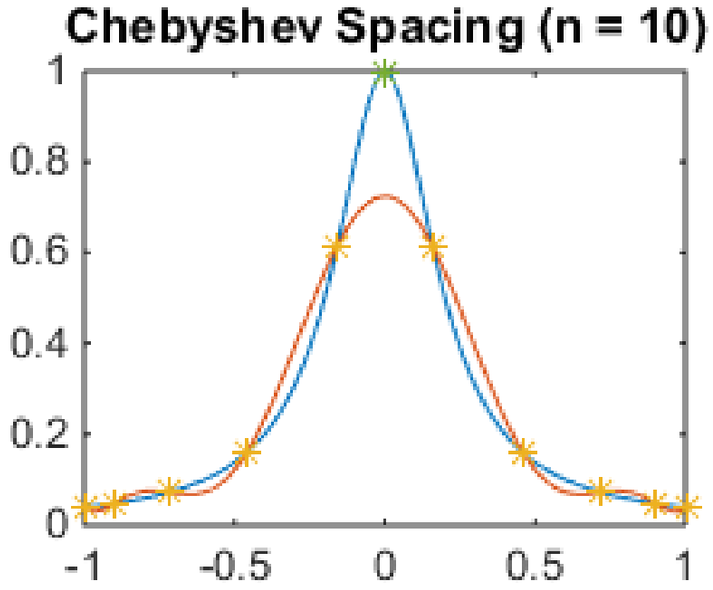
$$h^n \frac{(n-1)!}{4}$$

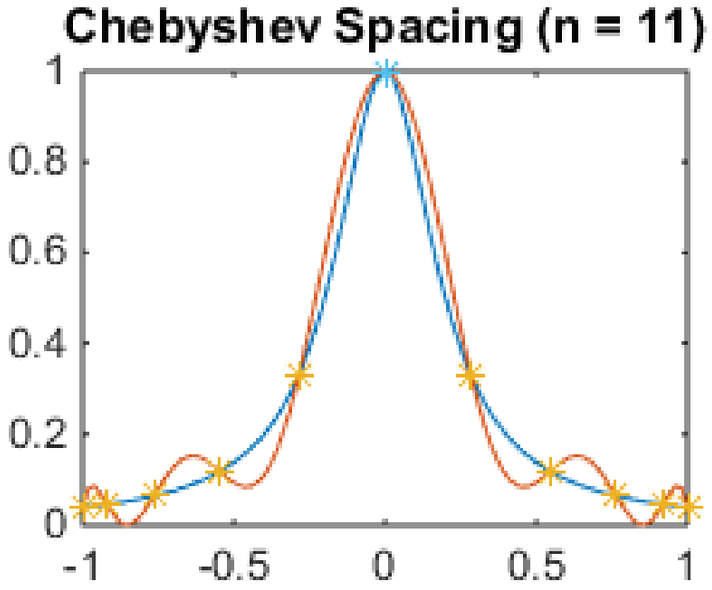
Runge Example Using Chebyshev Points

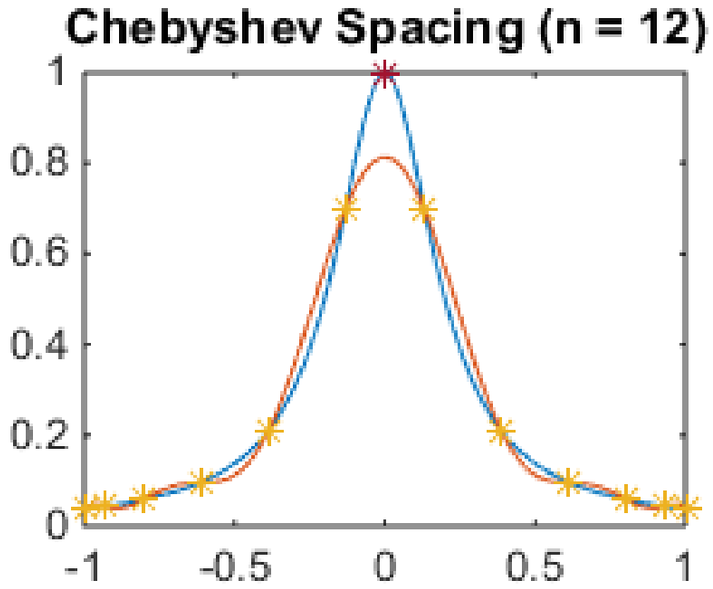
```
%  Script File: RungeEg
% For n=10:13, interpolants of f(x) = 1/(1+25x^2) on [-1,1]' are of plotted.
  close all
  x = linspace(-1,1,100);
  y = ones(1,100)./(1 + 25*x.^2);
  iplot = 0;
  for n=10:13
     xEqual = zeros(n);
     yEqual = zeros(n);
     for kk = 1:n
      xEqual(kk) = -cos( (2.0*kk-1.0)*pi/(2.0*n)) / cos (pi/(2.0*n));
     end
     yEqual = ones(n)./(1 +25*xEqual.^2);
     cEqual=InterpN(xEqual,yEqual);
     pvals = HornerN(cEqual,xEqual,x);
     iplot = iplot+1;
     subplot(2,2,iplot)
     plot(x,y,x,pvals,xEqual,yEqual,'*')
     title(sprintf('Chebyshev Spacing (n = %2.0f)',n))
  end
```

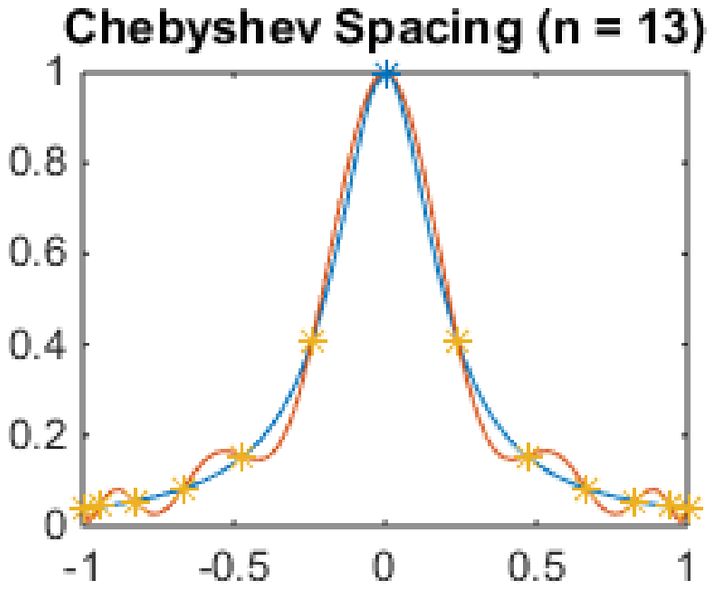
Runge's Example with Chebyshev Points

Runge's Example Chebyshev polynomial interpolation points true solution blue polynomial approximation









Legendre and Chebyshev Orthogonal Polynomials

Lagrange polynomials are not often used in realistic calculations as they are somewhat unwieldy in practice. Instead Legendre and Chebyshev polynomials are used

- **Legendre polynomials** satisfy the following recurrence relation to compute L_m

$$L_{m+1}(x) = \frac{2m+1}{m+1} x L_m(x) - \frac{m}{m+1} L_{m-1}(x)$$

- Starting with $L_0=1$ $L_1=x$ we obtain:

$$L_2(x) = \frac{3}{2} x L_1(x) - \frac{1}{2} L_0(x) = \frac{1}{2}\left(3x^2 - 1\right)$$

- Satisfy the orthogonality conditions

$$\int_{-1}^{1} L_p(x) L_m(x)\, dx = 0 \quad \text{if } p \neq m$$

$$\int_{-1}^{1} L_m(x)^2\, dx = \frac{2}{2m+1}$$

The Chebyshev polynomials are orthogonal

Chebyshev polynomials are an orthogonal set of functions in the interval [-1,1] with respect to the weight function $1/\sqrt{1-x^2}$ *such that*

$$\int_{-1}^{1} T_k(x)T_j(x)\frac{dx}{\sqrt{1-x^2}} = \begin{cases} 0 & for \quad k \neq j \\ \pi/2 & for \quad k = j > 0 \\ \pi & for \quad k = j = 0 \end{cases}$$

This is verified by noting that $x = \cos\varphi, \qquad dx = -\sin\varphi d\varphi$

$$T_k(x) = \cos(k\varphi), \quad T_j(x) = \cos(j\varphi)$$

Recurrence Relation $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$

Chebyshev polynomials - definition

$$\cos(n\phi) = T_n\big(\cos(\phi)\big) = T_n(x), where \quad x = \cos(\phi), and \quad x \in [-1,1],$$

Defines the Chebyshev polynomials $T_n(x)$. Note that because of $x=\cos(\varphi)$ they are defined in the interval [-1,1] (which - however - can be extended to $\mathfrak{R}$). The first polynomials are

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_2(x) = 2x^2 - 1$$
$$T_3(x) = 4x^3 - 3x$$
$$T_4(x) = 8x^4 - 8x^2 + 1 \qquad \text{where}$$
$$\left|T_n(x)\right| \le 1 \qquad \text{for} \quad x \in [-1,1]$$

Chebyshev Interpolation at Chebyshev Points

If function f(x) is defined only at the points $x_i = \cos\frac{\pi}{N} i$

in this case the Chebyshev coefficients are given by

$$c_k^* = \frac{2}{N}\sum_{j=1}^{N} f(\cos\phi_j)\cos(k\phi_j), \qquad k = 0,1,2,\ldots N/2$$

... leading to the polynomial ...

$$F_m^*(x) = \frac{1}{2}c_0^* T_0 + \sum_{k=1}^{m} c_k^* T_k(x)$$

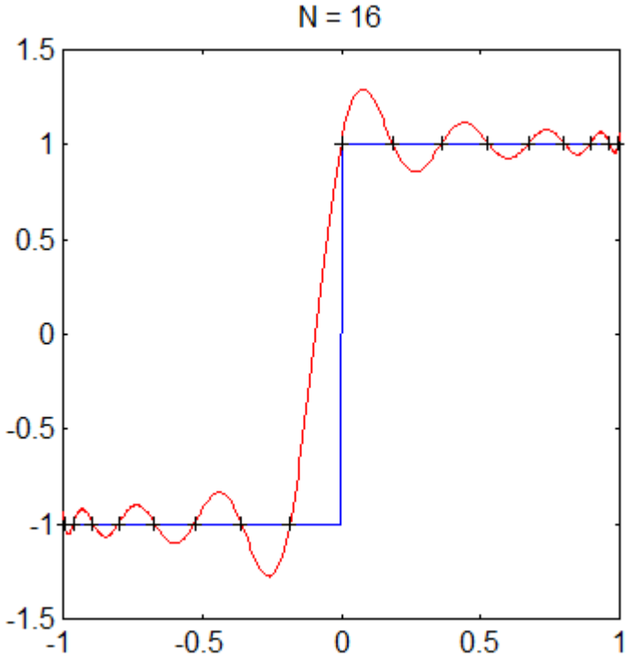
... with the property

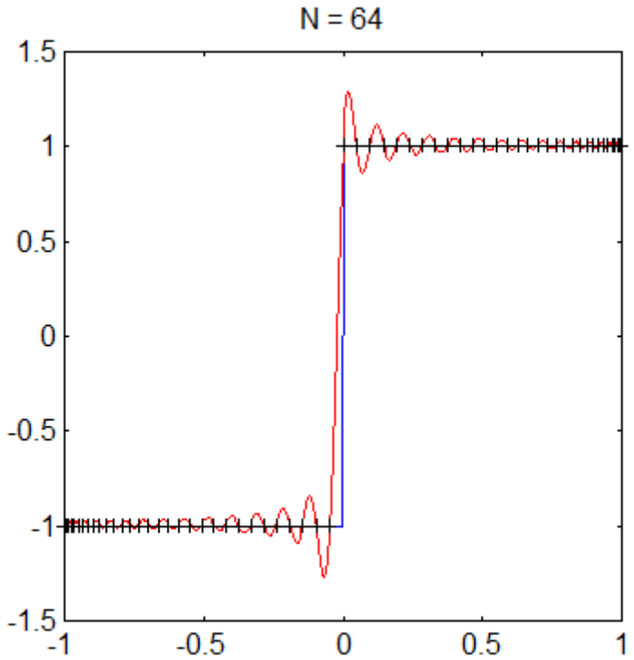
$$F_m^*(x) = f(x) \quad at \quad x_j = cos(\pi j/N) \quad j = 0,1,2,\ldots,N$$

Chebyshev - Gibb's Oscillations

https://en.wikipedia.org/wiki/Gibbs_phenomenon f(x)=sign(x-π) => f(x) - blue ; $F_N(x)$ - red; x_i - '+'

Chebyshev





For functions with steep gradients we get overshoots and undershoots in the interpolating polynomial these are the famous Gibbs oscillations originally discovered with Fourier Series..

Bernstein Polynomials

$$b_{i,n}(x) = \binom{n}{i} x^i (1-x)^{n-i}, i = 0,\ldots,n$$

where $\binom{n}{i} = \frac{n!}{i!(n-i)!}$

These polynomials are positive everywhere but are not interpolatory. They are used in the proof of the Weierstrass Theorem.

Linear combination of these polynomials is

$$B_n(x) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) b_{k,n}(x)$$

Recurrence relation

$$b_{k,n}(x) = (1-x)b_{k,n-1}(x) + xb_{k-1,n-1}(x)$$

Bernstein Polynomials

$$b_{0,0}(x) = 1$$

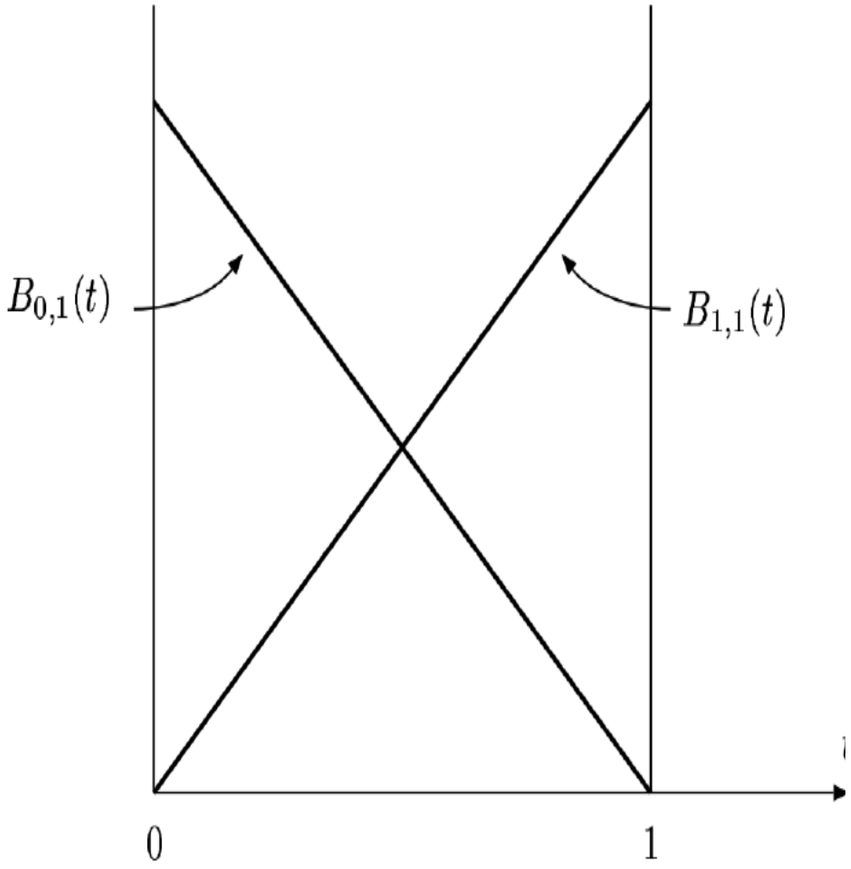
$$b_{0,1}(x) = 1 - x, b_{1,1}(x) = x$$

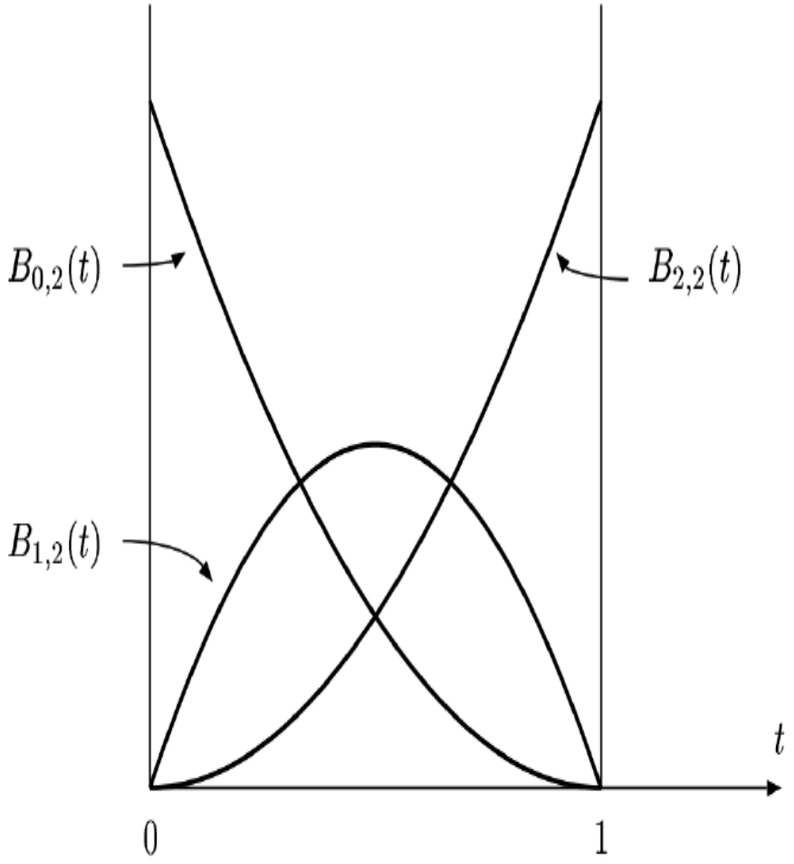
$$b_{0,2}(x) = (1-x)^2, b_{1,2}(x) = 2x(1-x), b_{2,2}(x) = x^2$$

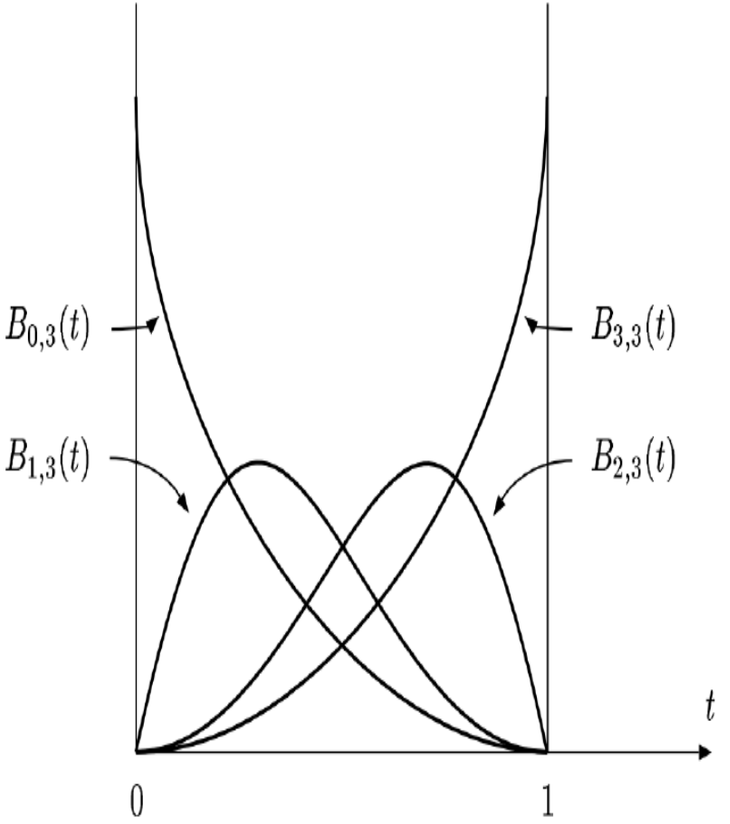
$$b_{0,3}(x) = (1-x)^3, b_{1,3}(x) = 3x(1-x)^2, b_{2,3}(x) = 3x^2(1-x), b_{3,3}(x) = x^3$$

Although convergence is guaranteed it is slow

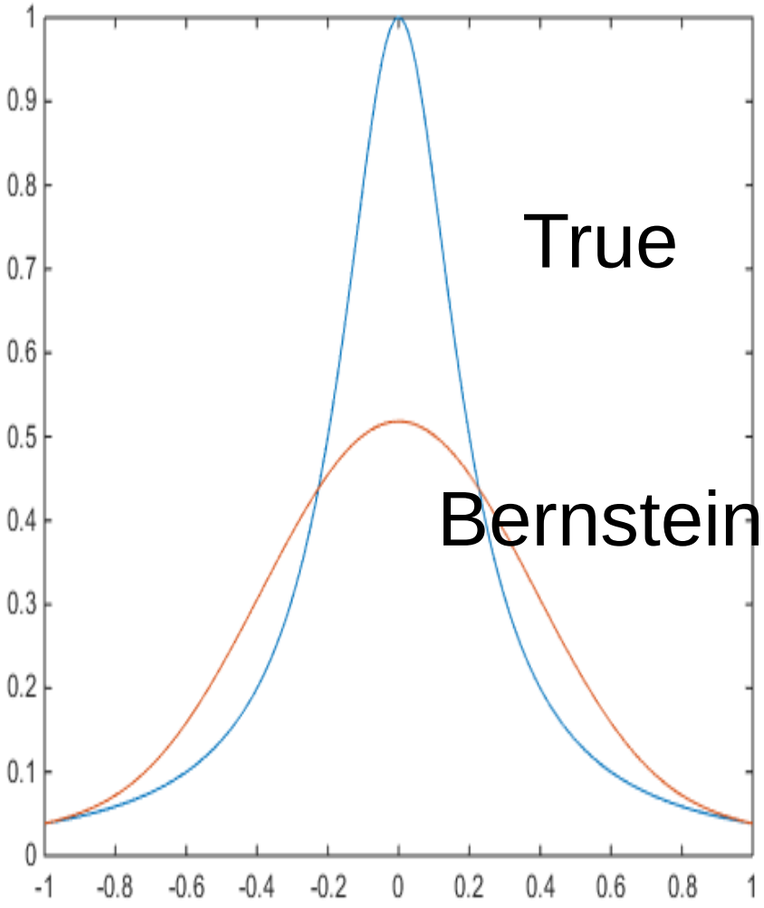
Bernstein Polynomials

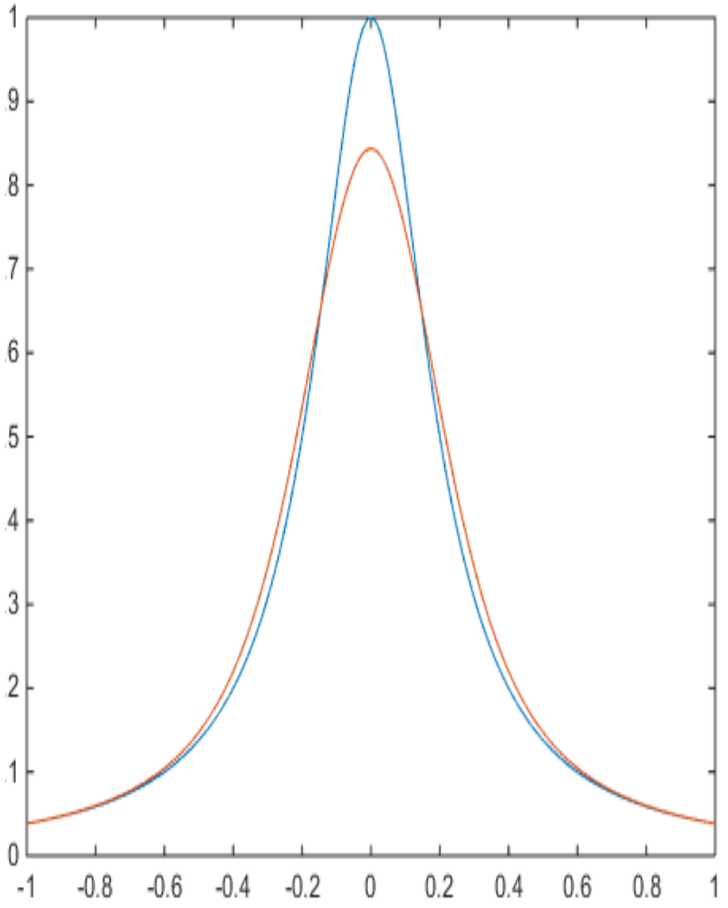


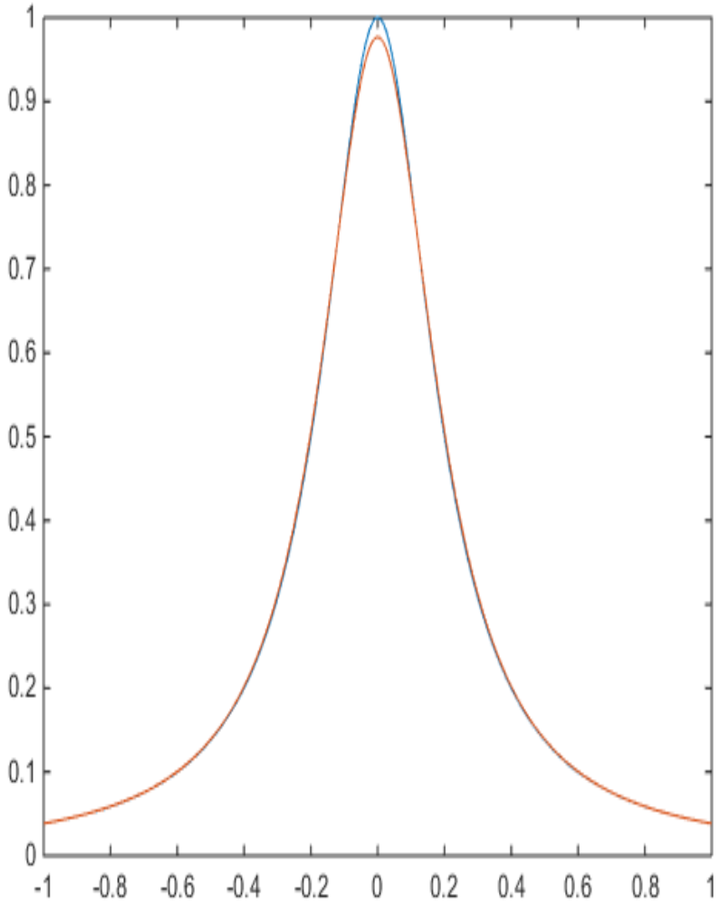




Bernstein Polynomials Approximating Runge's Function







Degree 11

Degree 101

Degree1001

Measuring the error, e(x), in an approximation?

For a polynomial of degree n, $p_n(x)$ approximating a function $f(x)$

The error $e(x) = p_n(x) - f(x)$, has to be calculated somehow.

We estimate the error over $[a,b]$ by using "norms".

The Lp norm of a function $g(x)$ is $\|g\|_p = \left(\int_a^b |g(x)|^p \, dx \right)^{1/p}$,

We now estimate the norm of the error i.e. this integral using pointwise errors

Let $e_i = p_n(x_i) - f(x_i)$, where $x_i = a + (i-1)\dfrac{(b-a)}{(m-1)}, i = 1...m,$

where $m >> n$, so that the interpolation points don't dominate.

Measuring measure the error, e(x), in an approximation?

Let $e_i = p_n(x_i) - f(x_i)$, where $x_i = a + (i-1)\frac{(b-a)}{(n-1)}, i = 1...n$

$L1$ function norm

$$p = 1,\ \|e\|_1 = \left(\int_a^b |\, e(x)\,| \ dx\right) \approx \frac{(b-a)}{(n-1)} \sum_{j=1}^{n-1} 0.5(|\, e_j\,| + |\, e_{j+1}\,|)$$

$L2$ function norm

$$p = 2,\ \|e\|_2 = \left(\int_a^b |\, e(x)\,|^2 \ dx\right)^{1/2} \approx \left[\frac{(b-a)}{(n-1)} \sum_{j=1}^{n-1} 0.5(|\, e_j\,|^2 + |\, e_{j+1}\,|^2)\right]^{\frac{1}{2}}$$

$L\inf$ function norm

$$p = \infty,\ \|e\|_\infty = \left(\int_a^b |\, e(x)\,|^\infty \ dx\right)^{1/\infty} \approx \max |\, e_j\,|,\ j = 1,...,n$$

Matlab Vector Norms

Let $e = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$

Matlab has norms of vectors they are related to the function norms in the last slide.

$L1$ vector norm

$$p = 1,\ norm(\mathrm{e},1) = \|\mathrm{e}\|_1 = \sum_{j=1}^{n1} | e_j |$$

$L2$ vector norm

$$p = 2, norm(e,2) = \ \|\mathrm{e}\|_2 = (\sum_{j=1}^{n} | e_j |^2)^{1/2}$$

$L\,\mathrm{inf}$ vector norm

$$p = \infty,\ \mathrm{norm(e,inf)} = \|\mathrm{e}\|_\infty = \max | e_j |,\ j = 1,\ldots,n$$

We can use the matlab vectors as approximations to functions normd if we we scale them appropriately

$$p=1,\ \|\mathrm{e}\|_1 \approx \frac{(b-a)}{(n-1)}\sum_{j=1}^{n-1} 0.5(|e_j|+|e_{j+1}|) \approx \frac{(b-a)}{(n-1)} norm(e,1)$$

$L2$ function norm

$$p=2,\ \|\mathrm{e}\|_2 \approx \left[\frac{(b-a)}{(n-1)}\sum_{j=1}^{n-1} 0.5(|e_j|^2+|e_{j+1}|^2)\right]^{\frac{1}{2}} \approx \sqrt{\frac{(b-a)}{(n-1)}} norm(e,2)$$

$L\inf$ function norm

$$p=\infty,\ \|\mathrm{e}\|_\infty \approx \max|e_j|,\ j=1,\ldots,n = norm(e,\inf)$$

Vectors and Functions

orthogonality a.b $(a,b)=0$ (f,g)=0

$$\sum_{i=1}^{n} a_i b_i = 0$$

$$\int_a^b f(x)g(x)dx = 0$$

Norm $\| a \|_2 = \sqrt{(a,a)}$

$$\| f \|_2 = \sqrt{\int_a^b f(x) f(x) dx}$$

(or size)

matlab $norm(a,2)$

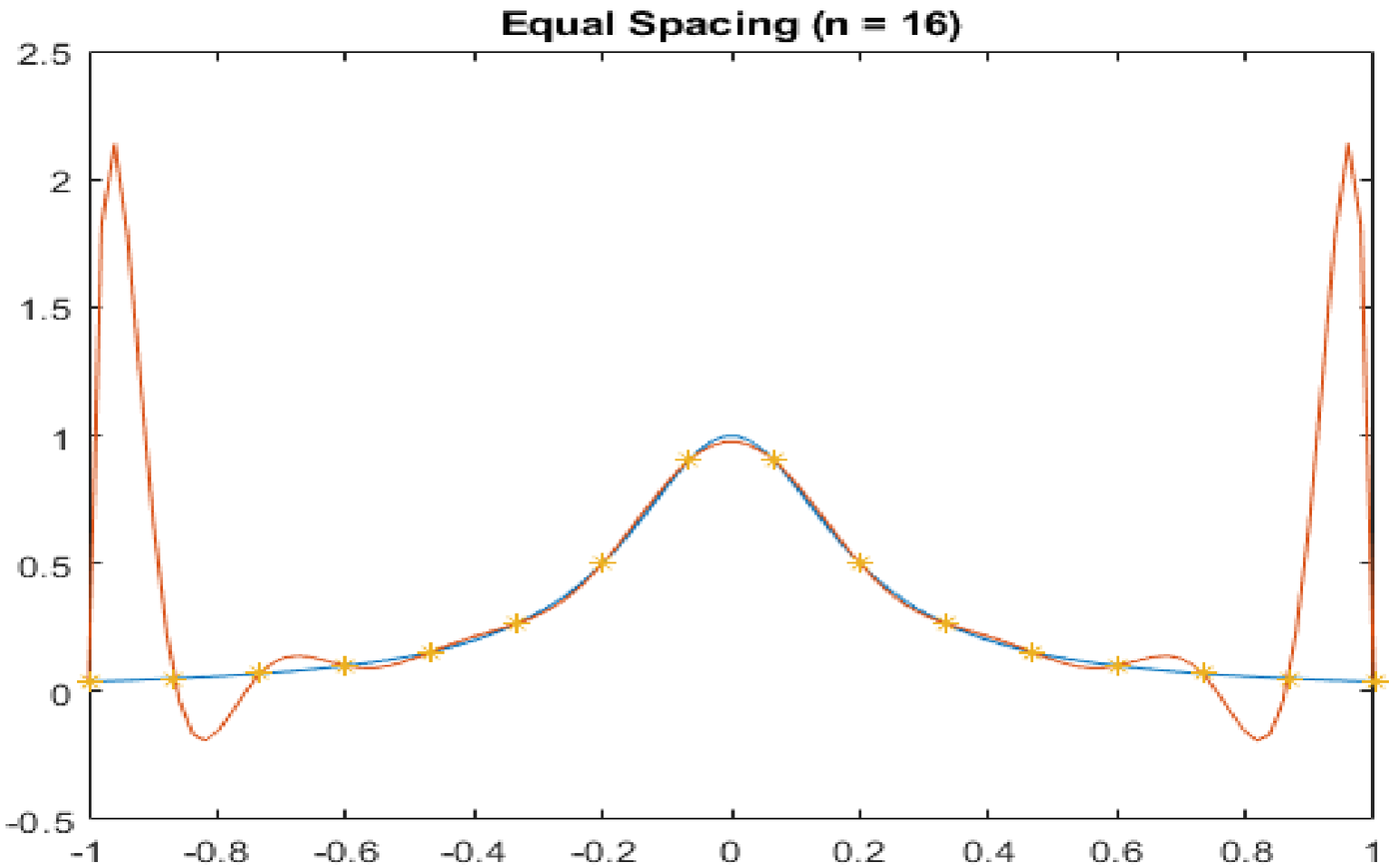
$$\sqrt{\frac{\text{(b-a)}}{n}} norm(f_n, 2)$$

vector $f_i = f(x_i), x_i$

evenly spaced

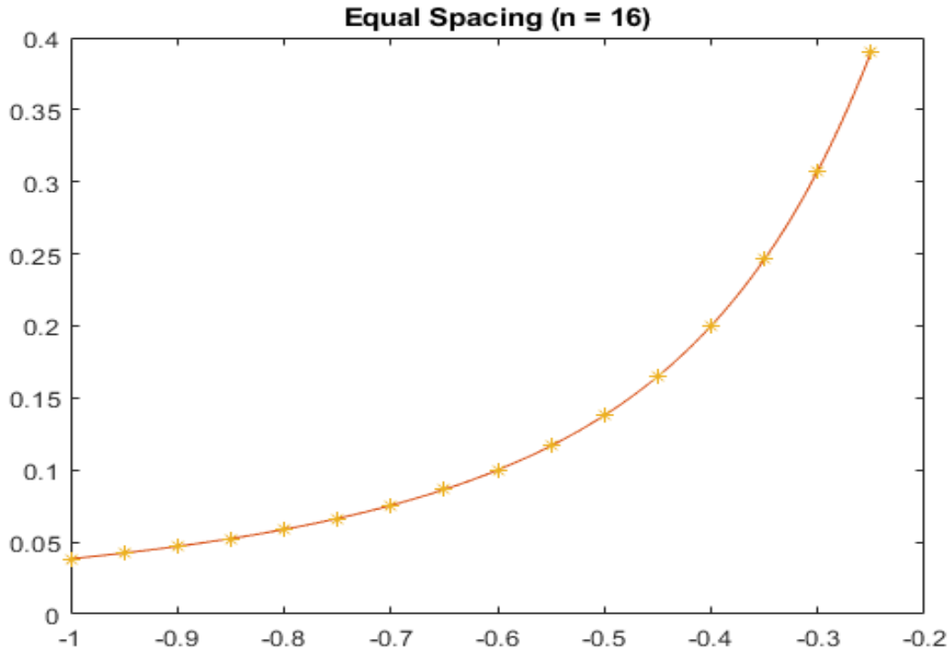
Piecewise Polynomials

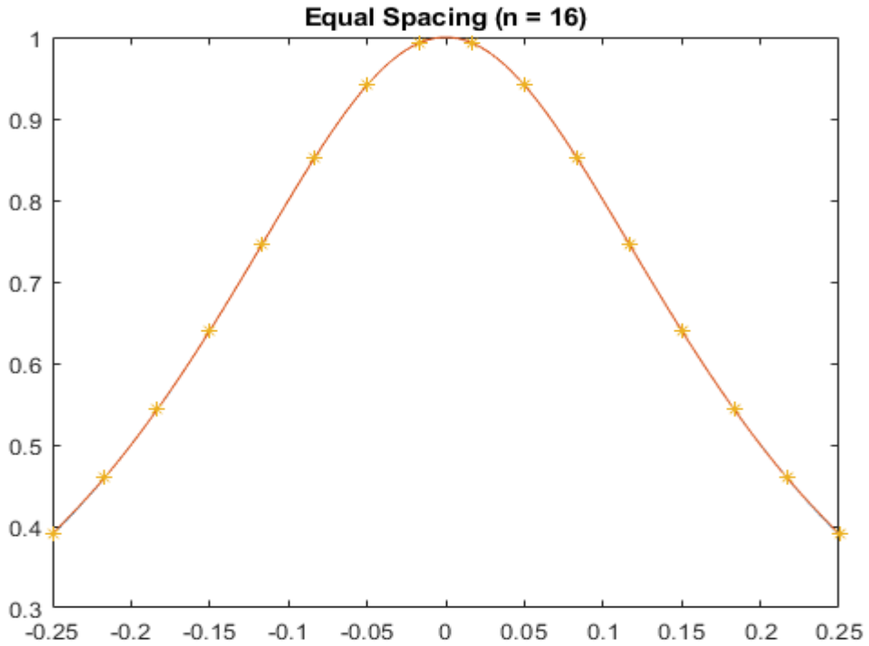
- A single polynomial can only do so much as shown below.
- What happens to the Runge Example if we use equal spacing and more than one polynomial?

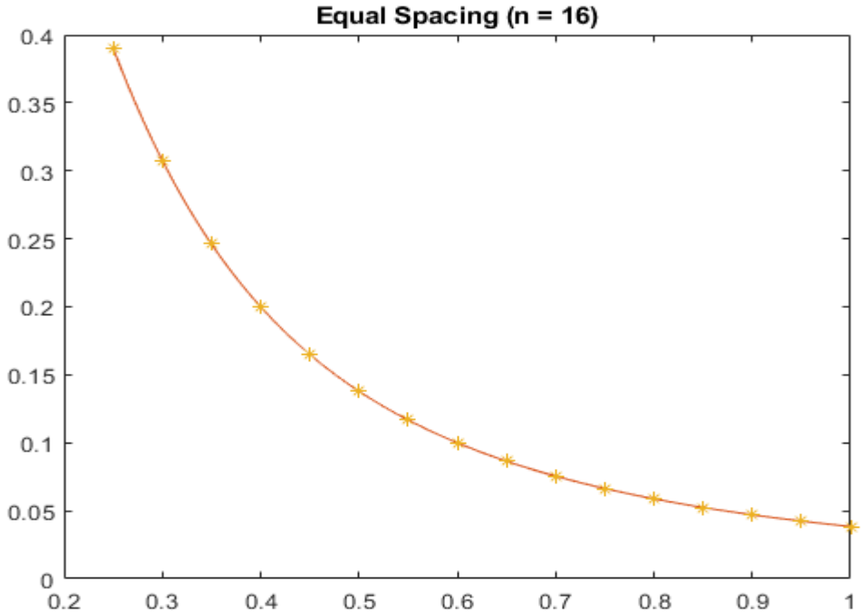


Piecewise Polynomials

- A single polynomial can only do so much.
- What happens to the Runge Example if we use equal spacing and 3 polynomials joined at -0.25 and 0.25?
- What we need is a way of joining polynomials together



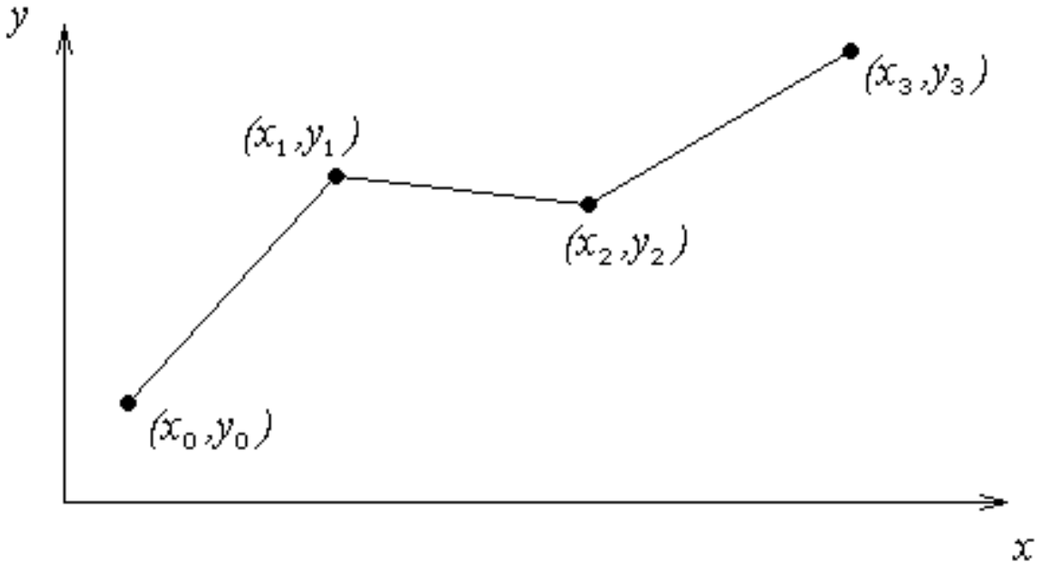




Splines

- A special form of interpolation using piecewise lines or polynomials on a subinterval
- The term spline refers to a wooden or plastic drawing tool that when passed through data points gives a smooth curve
- Often preferred to high-order polynomial interpolation because of reduced cost and (sometimes better) results without the oscillations of high-order polynomials

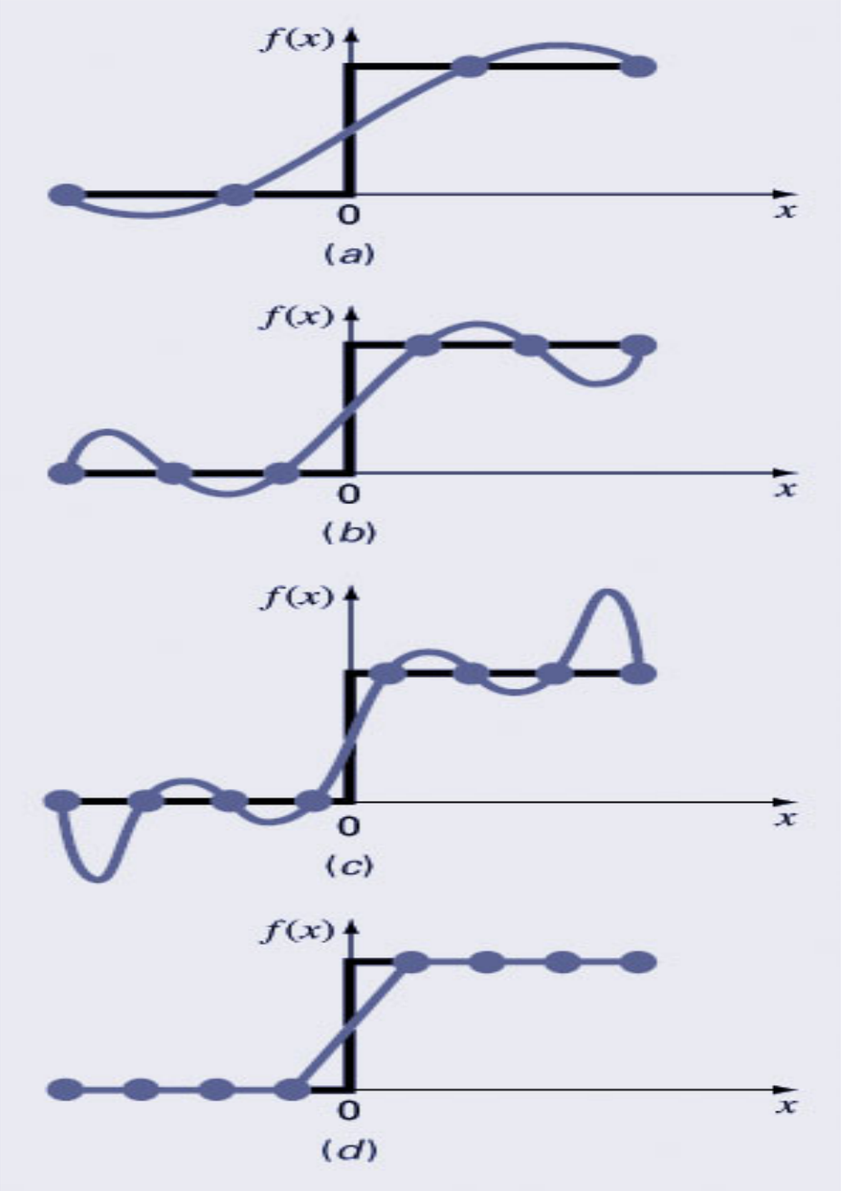
For example a linear spline passing through four points (not smooth in this case)



Higher Order Polynomials vs. Splines

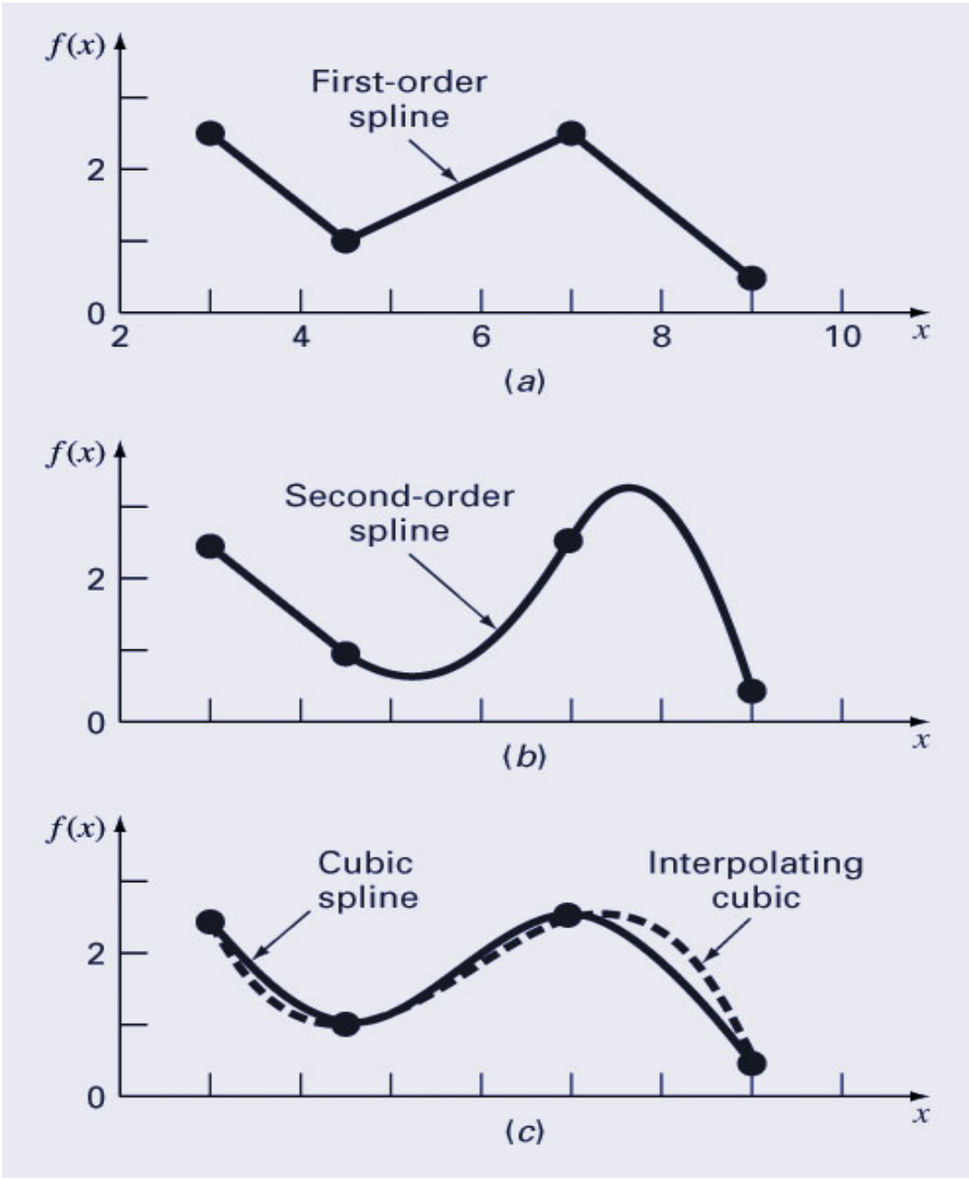
- Splines eliminate oscillations by using small subsets of points for each interval rather than every point. This is especially useful when there are jumps or steep gradients in the data:

a) 3rd order polynomial through four points
b) 5th order polynomial through six points
c) 7th order polynomial through eight points
d) Linear spline
    - seven 1st order polynomials generated by using pairs of points at a time



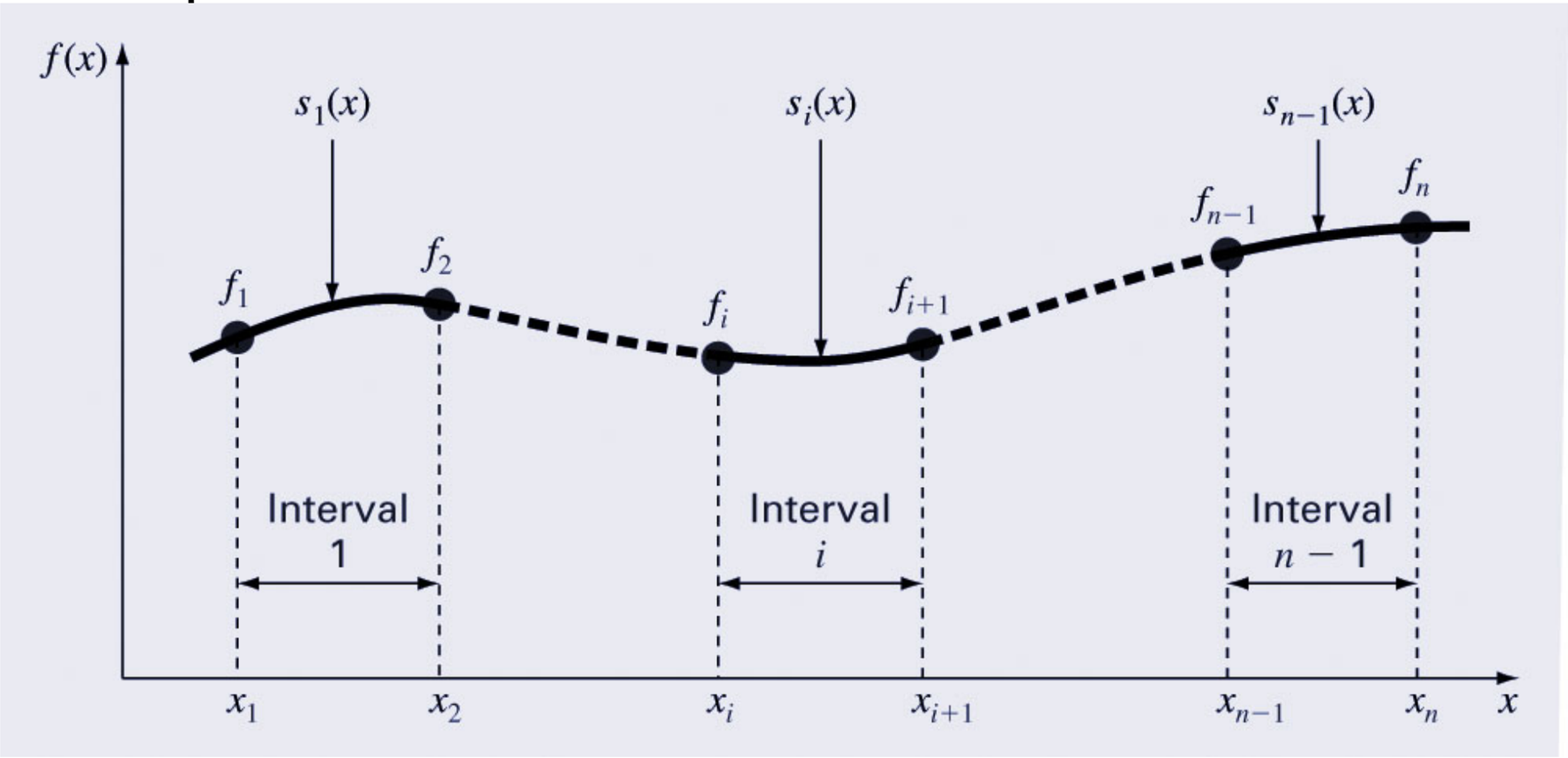
Spline Methods

- **First-order (linear) splines** find straight-line equations between each pair of points that
  - Go through the points
- **Second-order (quadratic) splines** find quadratic equations between each pair of points that
  - Go through the points
  - Match first derivatives at the interior points
- **Third-order (cubic) splines** find cubic equations between each pair of points that
  - Go through the points
  - Match first and second derivatives at the interior points



Spline Methods

- Spline function coefficients are calculated for each interval of a data set.
- The number of data points (f_i) used for each spline function depends on the order of the spline function.



Cubic splines

Cubic splines give the simplest representation with the appearance of smoothness and without the problems of higher order polynomials.

Linear splines have discontinuous first derivatives

Quadratic splines have discontinuous second derivatives and require setting the second derivative at some point to a pre-determined value

Quartic or higher-order splines tend to exhibit ill-conditioning or oscillations so B-splines are used.

The cubic spline function for the i^{th} interval $[x_i, x_{i+1}], i = 1, \ldots, N-1$ can be written as:

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

For N data points, there are (N-1) intervals and thus 4(N-1) unknowns to find to get all the spline function coefficients.

Conditions to determine the spline coefficients

The first condition → the spline function *goes through the first and last point of the interval*; this leads to 2(n-1) equations:

$$s_i(x_i) = f_i \Rightarrow a_i + b_i(x_i - x_i) + c_i(x_i - x_i)^2 + d_i(x_i - x_i)^3 = f_i \Rightarrow a_i = f_i$$

$$s_i(x_{i+1}) = f_{i+1} \Rightarrow s_i(x_{i+1}) = a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i)^3 = f_{i+1}$$

The second condition → *the first derivative should be continuous at each interior point*; this leads to (N-2) equations:

$$\frac{ds_i}{dx}(x_{i+1}) = \frac{ds_{i+1}}{dx}(x_{i+1}) \Rightarrow b_i + 2c_i(x_{i+1} - x_i) + 3d_i(x_{i+1} - x_i)^2 = b_{i+1}$$

Conditions to determine the spline coefficients

The third condition → *the second derivative should be continuous at each interior point*; this leads to (n-2) equations:

$$\frac{d^2 s_i}{dx^2}\left(x_{i+1}\right) = \frac{d^2 s_{i+1}}{dx^2}\left(x_{i+1}\right) \Rightarrow 2c_i + 6d_i\left(x_{i+1} - x_i\right) = 2c_{i+1}$$

So far we have (4N-6) equations; we need (4N-4) equations!

Natural end conditions → set the second derivative at the end points to zero.

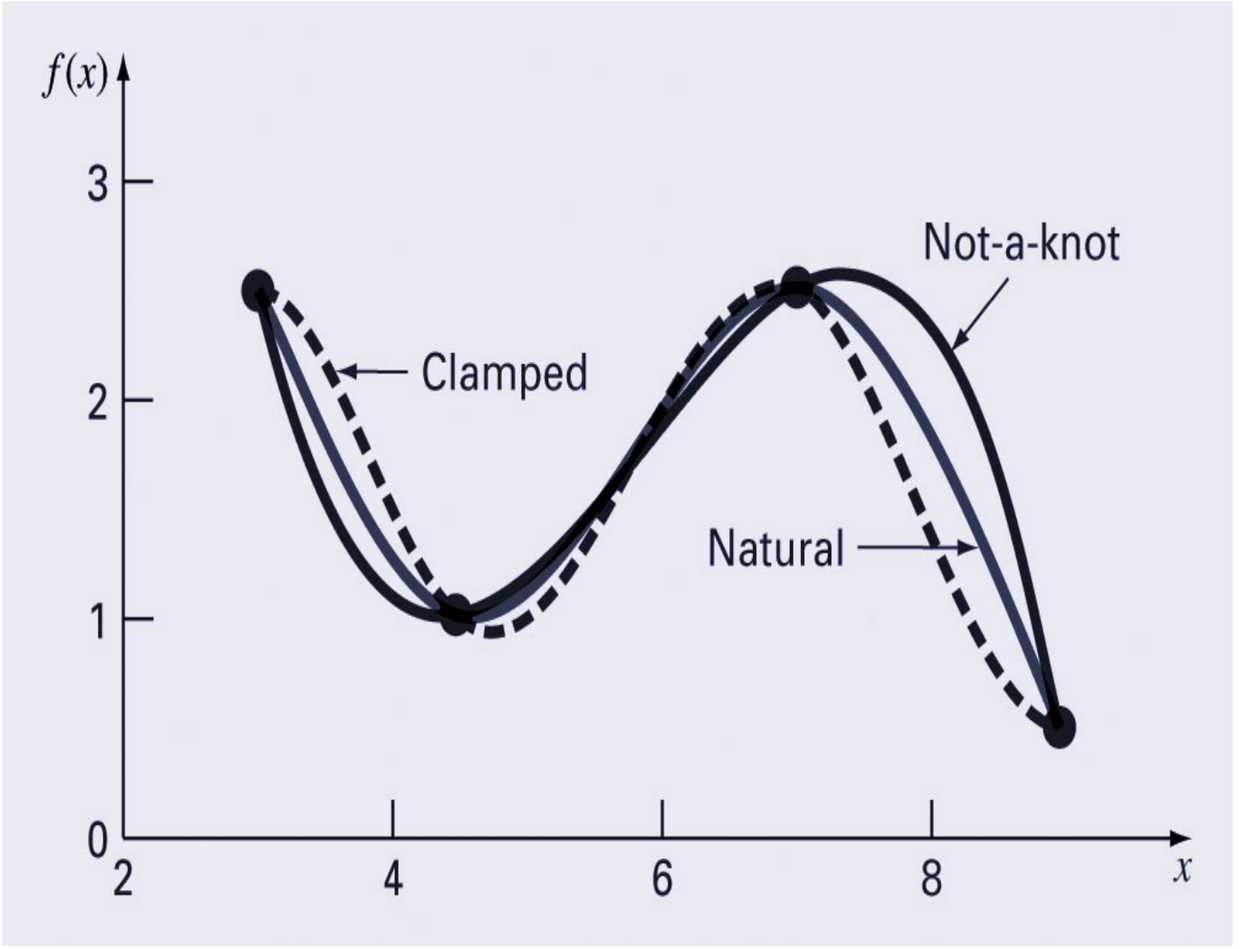
$$\Rightarrow 2c_1 = 0, 2c_{N-1} + 6d_{N-1}\left(x_N - x_{N-1}\right) = 0$$

Using End Conditions to define the last two equations

Other Options for the final two equations:

Clamped end conditions → the first derivatives at the first and last points are known.

“Not-a-knot” end conditions → force continuity of the *third* derivative at the second and penultimate points (results in the first two intervals having the same spline function and the last two intervals having the same spline function)



Matlab Cubic Spline P(x) (see page 100)

$$h = x_{i+1} - x_i \qquad s = x - x_i,$$

$$P(x) = \frac{3hs^2 - 2s^3}{h^3}\mathrm{y}_{i+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3}\mathrm{y}_i + \frac{s^2(s-h)}{h^3}\mathrm{D}_{i+1} + \frac{s(s-h)^2}{h^2}\mathrm{D}_i$$

where y_i are the values of the function at x_i

D_i is the gradient of the interpolating polynomial at x_i

At $x = x_i$, $P(x_i) = y_i$ as $s = 0$

$x = x_{i+1}$, $P(x_{i+1}) = y_{i+1}$ as $s = h$

We can also show $\frac{dP}{dx}(x_i) = D_i$ and $\frac{dP}{dx}(x_{i+1}) = D_{i+1}$

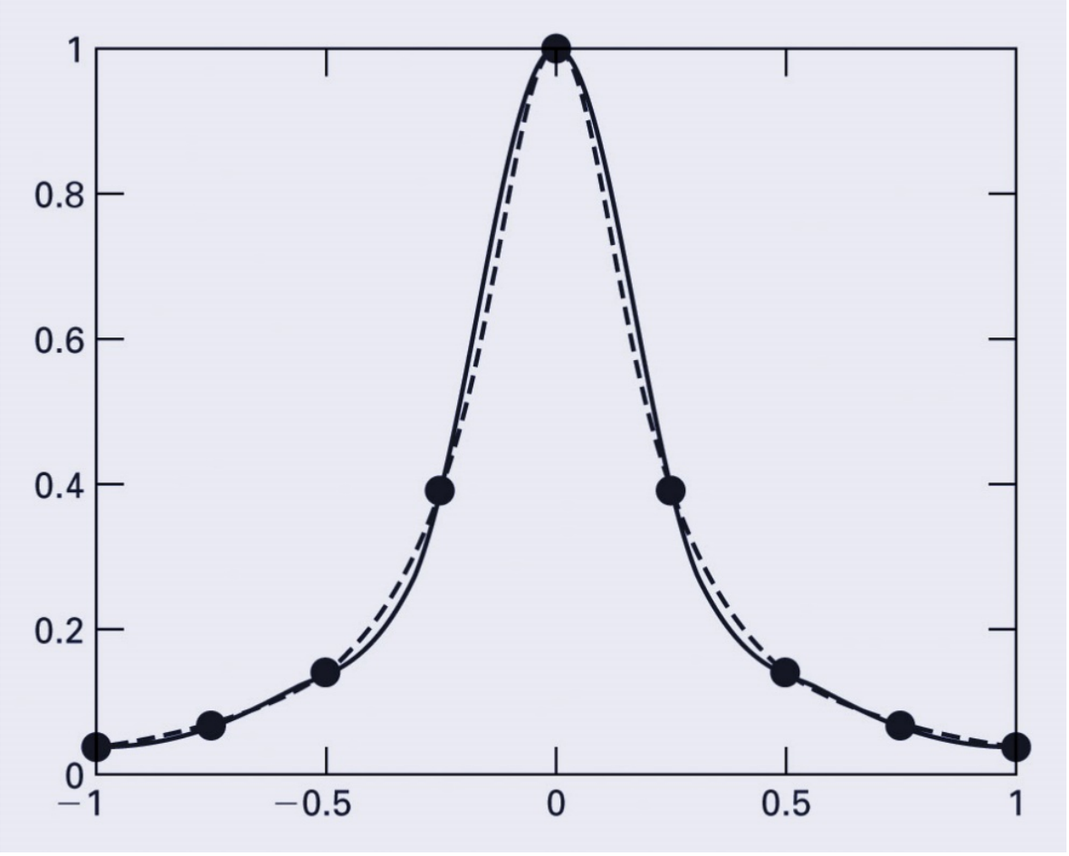
Tri-Diagonal Matrix for Natural Cubic Spline

In the case when the knots are all equally spaced and the spacing is one Pages 103-104 of the text book show that we have to solve the equations for the slopes D_i

We
Will consider
such systems of
equations later
and show that
they can be
solved quickly

$$\begin{bmatrix} 2 & 1 & & & & & & \\ 1 & 4 & 1 & & & & & \\ & 1 & 4 & 1 & & & & \\ & & 1 & 4 & 1 & & & \\ & & & . & . & . & . & \\ & & & & & 1 & 4 & 1 \\ & & & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ . \\ . \\ . \\ . \\ D_m \end{bmatrix} = \begin{bmatrix} 3(f_1 - f_0) \\ 3(f_2 - f_0) \\ . \\ . \\ . \\ 3(f_m - f_{m-2}) \\ 3(f_m - f_{m-1}) \end{bmatrix}$$

- MATLAB has several built-in functions to implement piecewise interpolation.
- spline→ yy=spline(x, y, xx)
  - Performs cubic spline interpolation, generally using not-a-knot conditions.
  - If y contains two more values than x has entries, then the first and last value in y are used as the derivatives at the end points (i.e. clamped)
- Example:
  - Generate data:
    x = linspace(-1, 1, 9);
    y = 1./(1+25*x.^2);
  - Calculate 100 model points and determine not-a-knot interpolation
    xx = linspace(-1, 1,100);
    yy = spline(x, y, xx);
  - Calculate actual function values at model points and data points, the 9-point not-a-knot interpolation (solid), and the actual function (dashed),
    yr = 1./(1+25*xx.^2)
    plot(x, y, 'o', xx, yy, '-', xx, yr, '--')



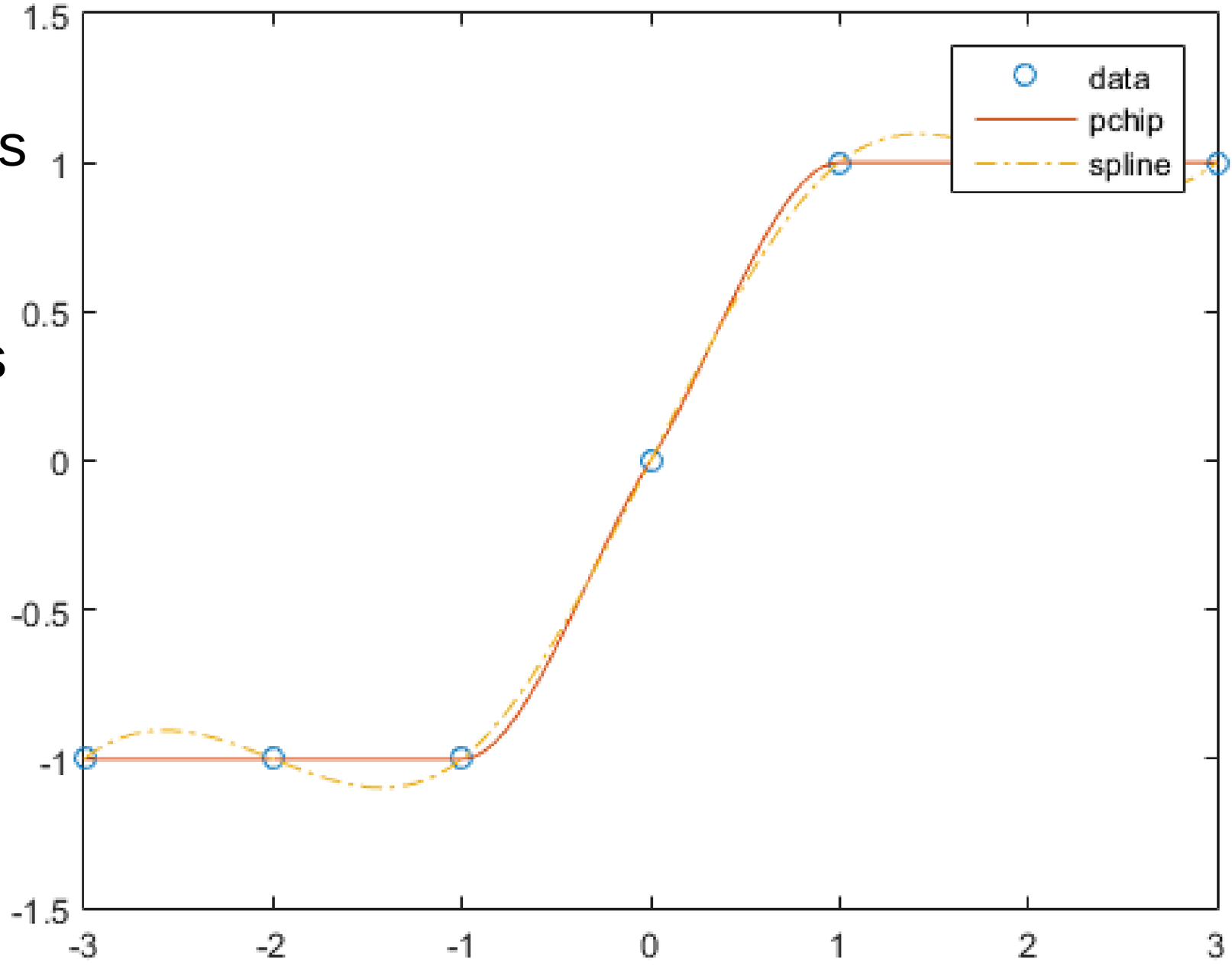
MATLAB's `interp1` Function

- While `spline` can only perform cubic splines, MATLAB's `interp1` function can perform several different kinds of interpolation:
  `yi = interp1(x, y, xi, 'method')`
  - `x` & `y` contain the original data
  - `xi` contains the points at which to interpolate
  - `'method'` is a string containing the desired method:
    - `'nearest'` - nearest neighbor interpolation
    - `'linear'` - connects the points with straight lines
    - `'spline'` - not-a-knot cubic spline interpolation
    - `'pchip'` or `'cubic'` - piecewise cubic Hermite interpolation

Example Pchip vs Spline

Pchip uses extra techniques to constrain the solution so that the polynomial is Bounded by the data points

```
x= -3:3;
y = [-1 -1 -1 0 1 1 1];
 t = -3:.01:3;
p = pchip(x,y,t);
s = spline(x,y,t);
plot(x,y,'o',t,p,'-',t,s,'-.')
legend('data','pchip','spline',4)
```



The importance of Polynomial Splines

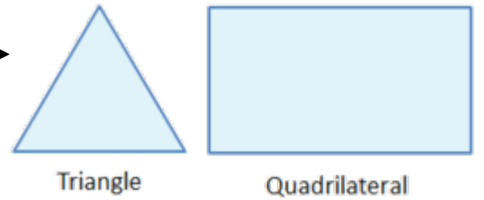
- Almost all practical use of polynomials in graphics computer aided design is done using some kind of splines or similar local polynomials
- Examples are B splines, Bezier functions, NURBS (Non-Uniform Rational Basis Splines), Catmull Rom etc
- All these polynomials are built up using similar ideas to the ones shown here.
- Some of these splines use Bezier polynomials

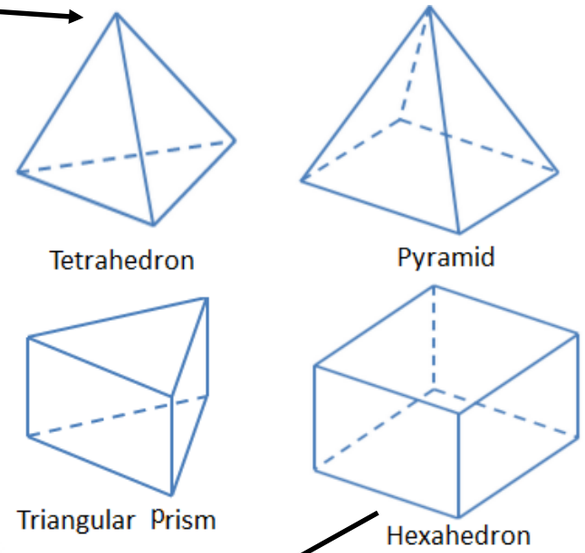
Multidimensional Simulations

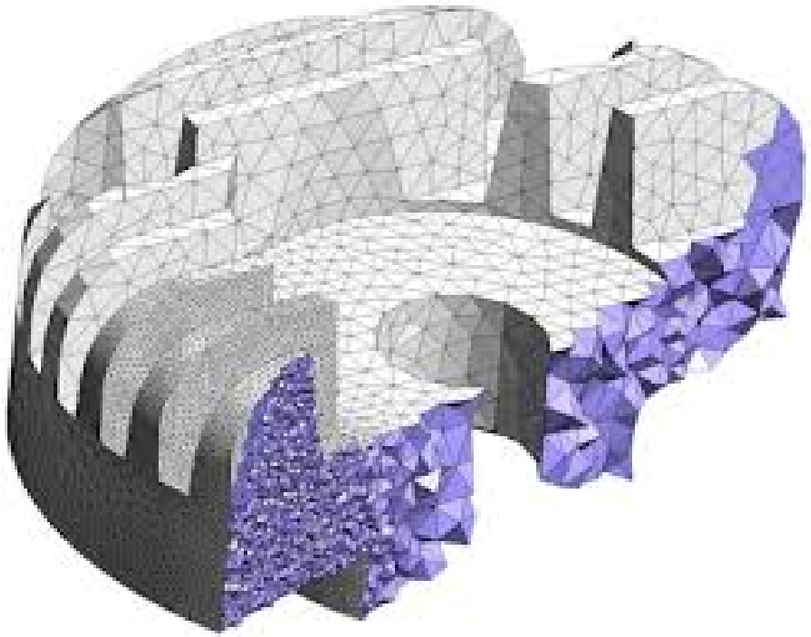
Objects are discretized in 2d using

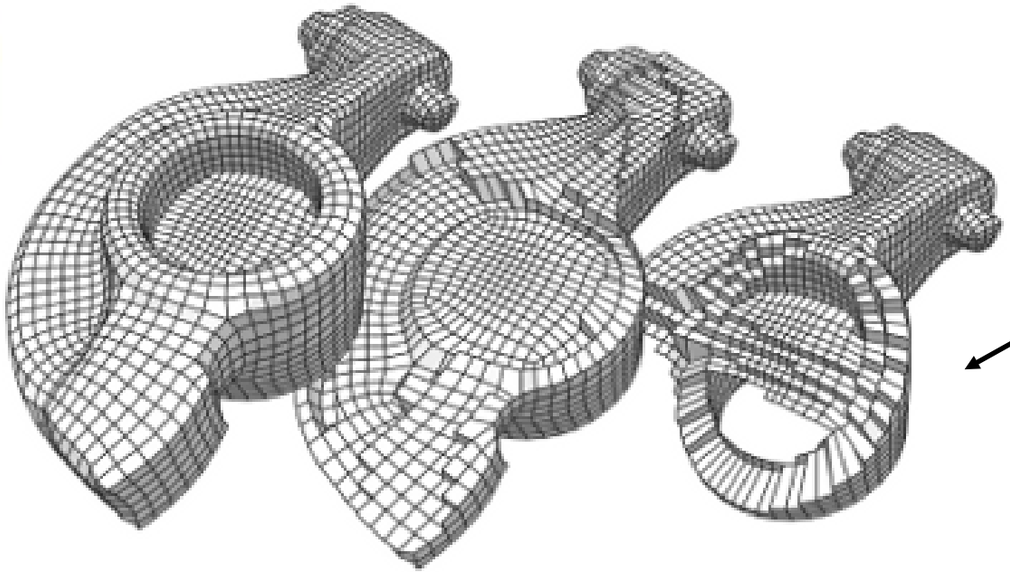
and in 3D using

Polynomials on these elements are used as the basis for the engineering calculation on the object

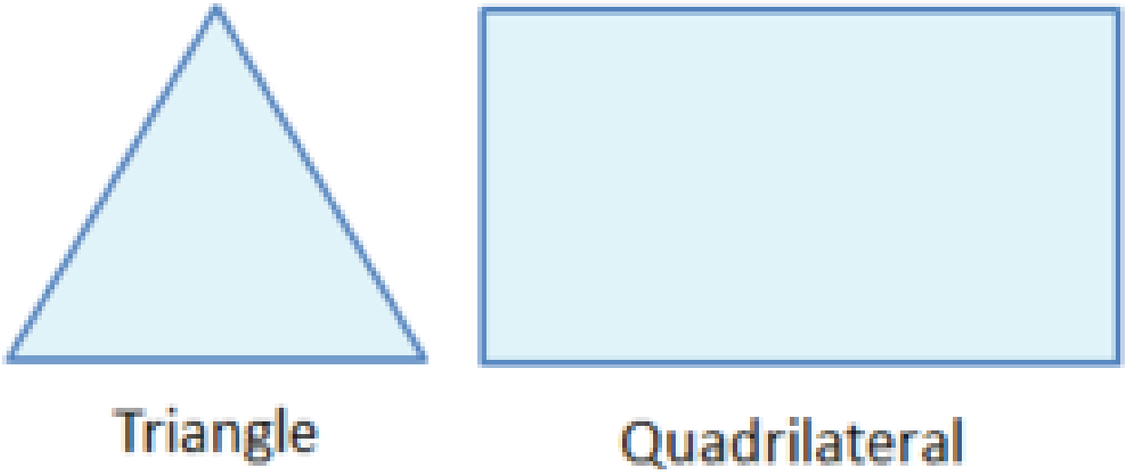


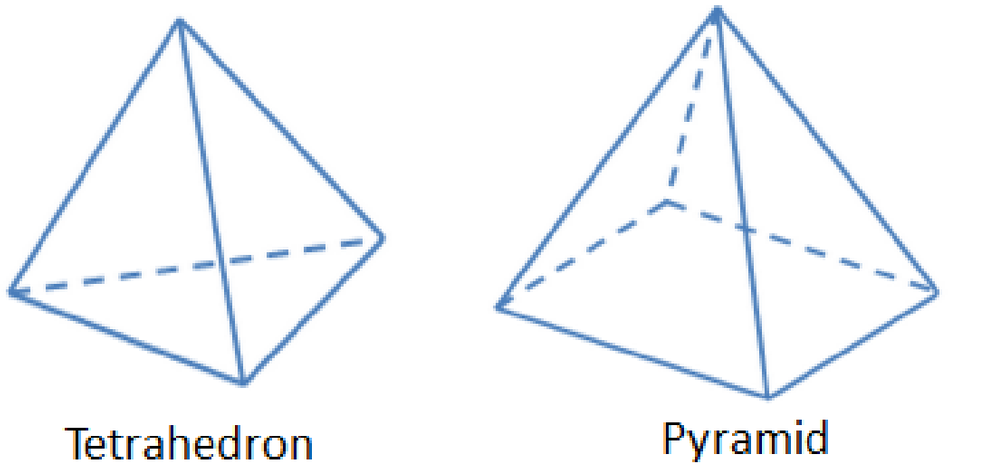




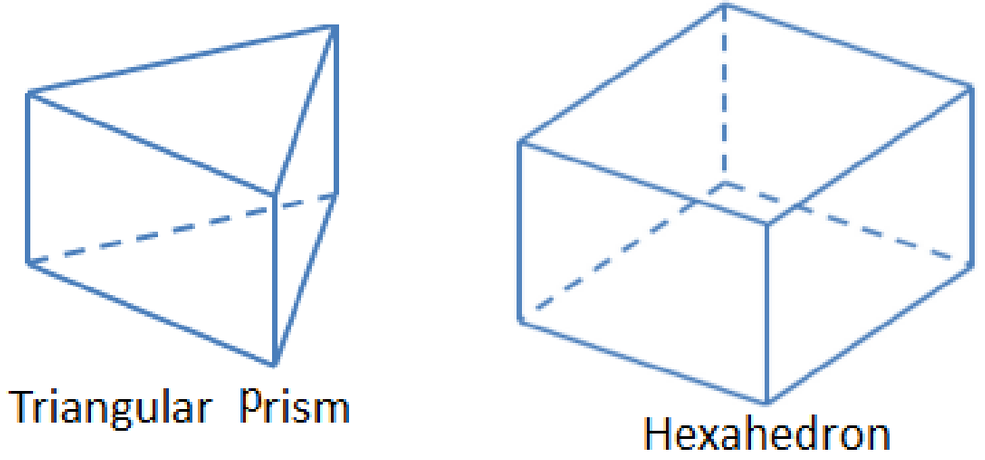


How do we interpolate on these objects?



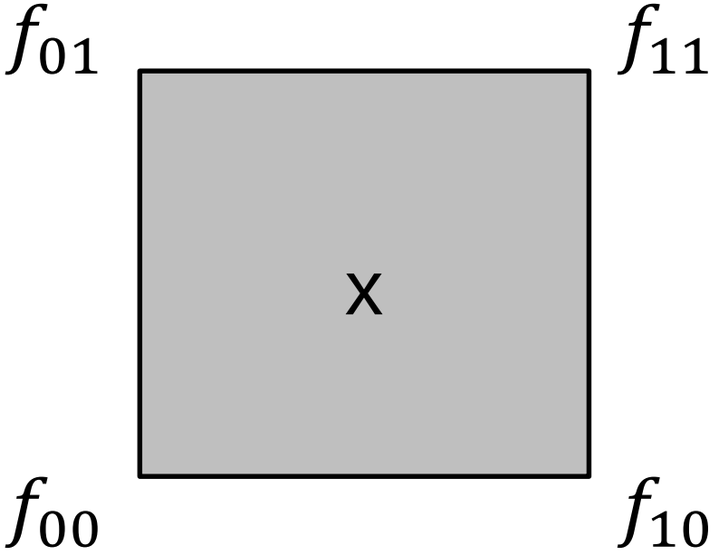


We will consider only triangles and quadrilaterals, but similar ideas apply in 3D to tetrahedral and hexahedra



Bilinear Interpolation

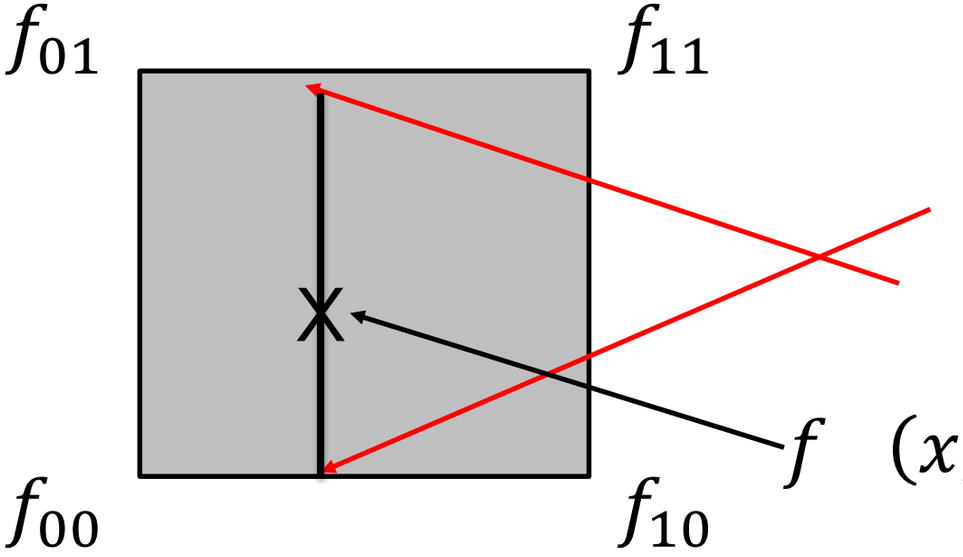
- Consider a rectangle:
  - Suppose the corners form a unit square, at $(0,0), (1,0), (1,1), (0,1)$
  - Let the values at the corners be $f_{00}, f_{10}, f_{11}, f_{01}$



How do we estimate the value at a point $\mathbf{f}(x, y)$ inside the square?

Bilinear Interpolation

- Carry out 3 linear interpolation operations:
  - Interpolate in the x direction between f_{00} and f_{10}
  - Interpolate in the x direction between f_{01} and f_{11}
  - Interpolate in the y direction between the x interpolants



$$f_0\ (x) = f_{00} + (x)(f_{10} - f_{00})$$

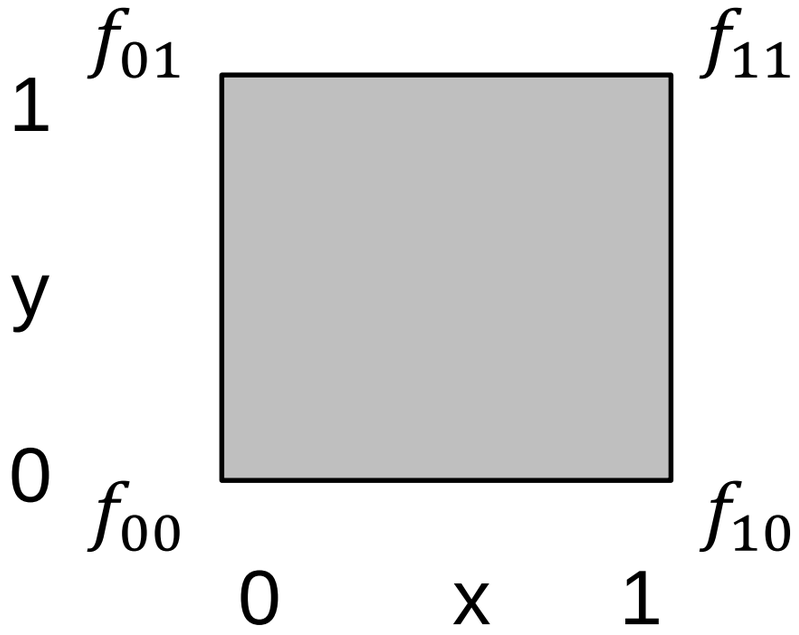
$$f_1\ (x) = f_{01} + (x)(f_{11} - f_{01})$$

$$f\ (x, y) = f_0\ (x) + (y)\left(f_1\ (x) - f_0\ (x)\right)$$

Bilinear Interpolation

- Expanded and reordered:

$$f(x,y) = (1-x)(1-y)f_{00} + x(1-y)f_{10} + (1-x)yf_{01} + xyf_{11}$$



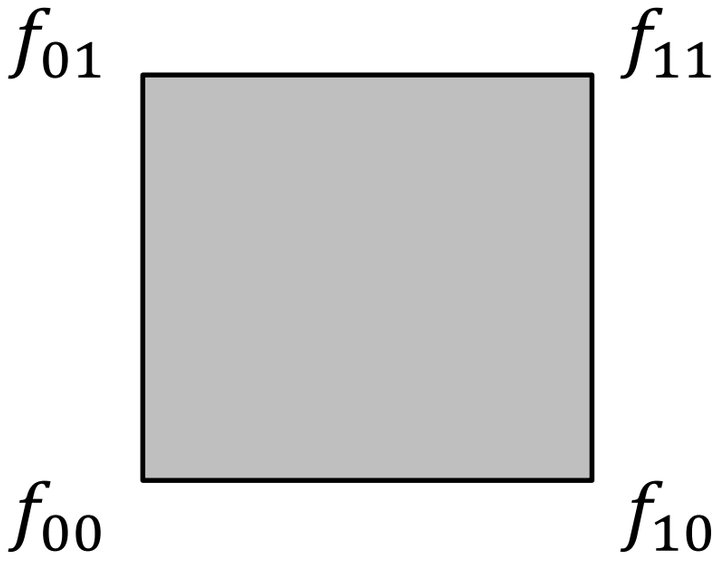
Each of these functions has value 1 at one corner and is zero at the other three corners

Generalization

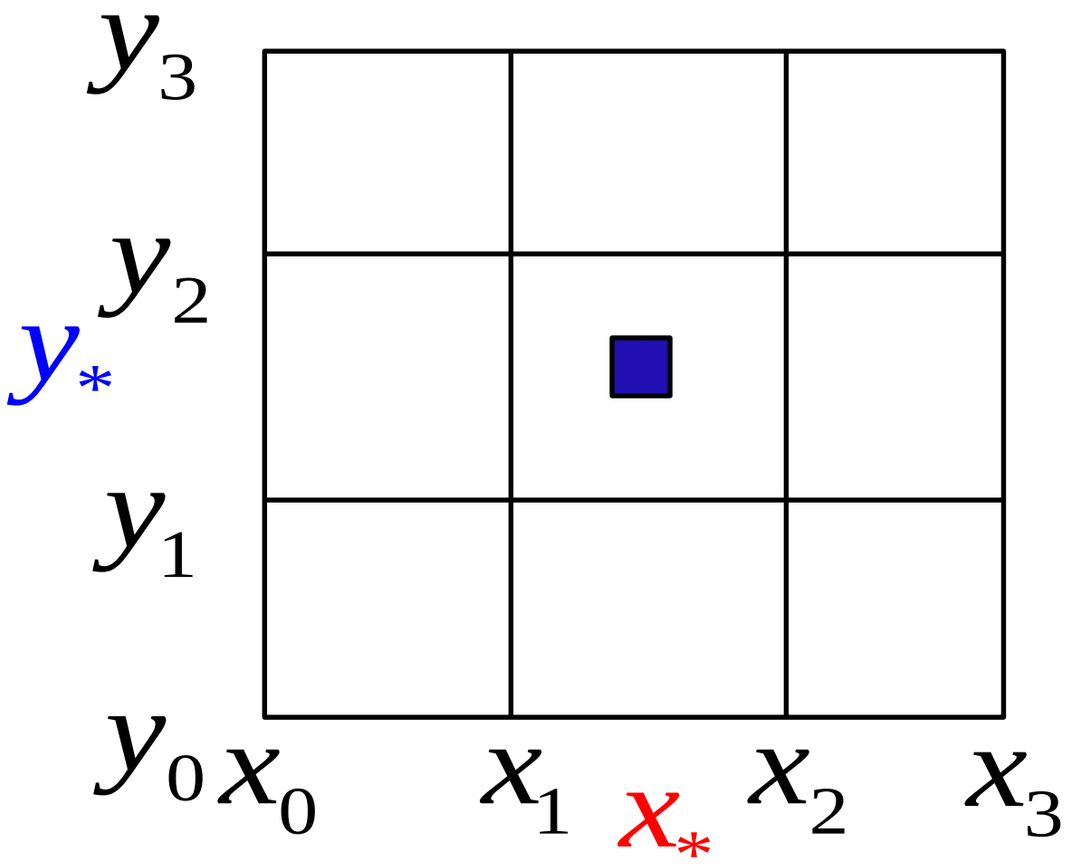
- How do you apply bilinear interpolation to non-unit square?
- For example, square corners $(2,3), (4,3), (4,5), (2,5)$ for locations $(2.5,4.5)$
  - Remember: unit square corners $(0,0), (1,0), (1,1), (0,1)$

Bilinear Interpolation

- For any rectangle:
  - $t_x = \frac{x-x_0}{x_1-x_0}, t_y = \frac{y-y_0}{y_1-y_0}$
  - $f(x,y) = (1-t_x)(1-t_y)f_{00} + t_x(1-t_y)f_{10} + (1-t_x)t_y f_{01} + t_x t_y f_{11}$



Extension Cubic Polynomials in 2D

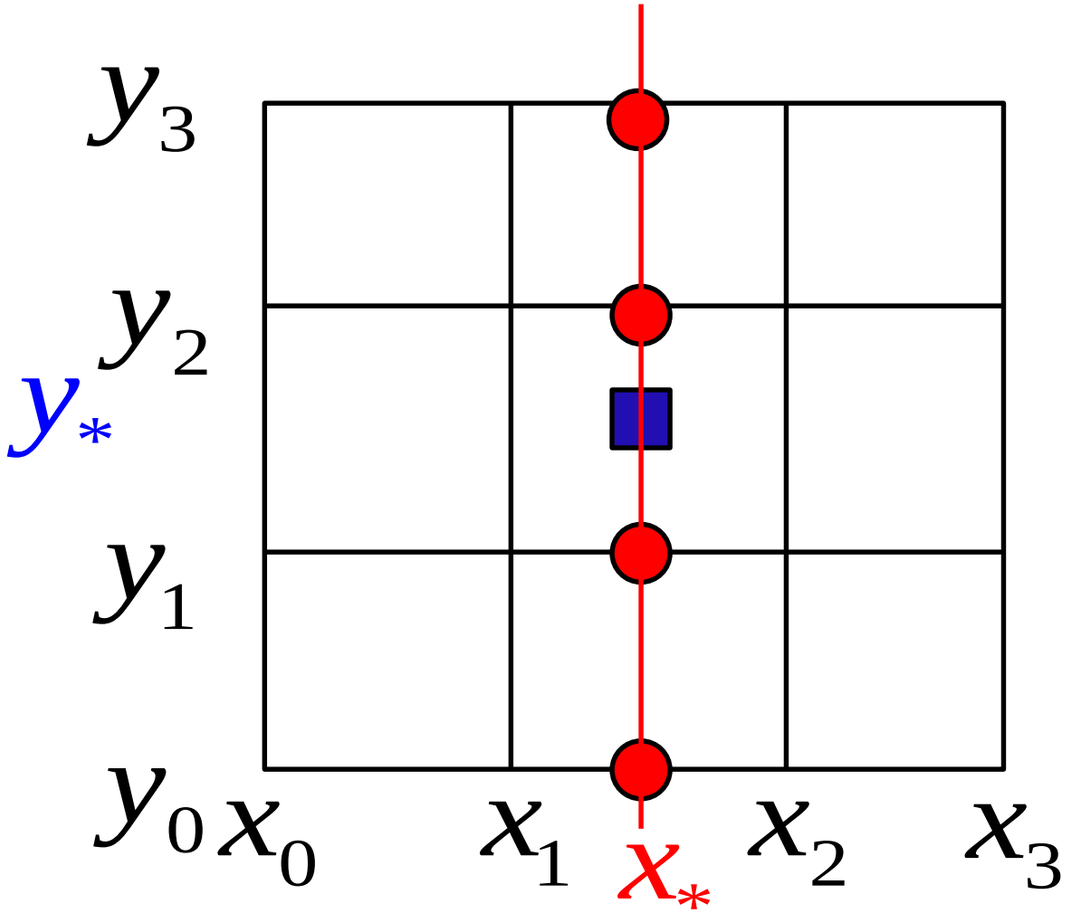


We have values at the points

$u(x_i, y_j), i = 0,1,2,3,\ j = 0,1,2,3$

How do we use 1D cubic polynomials to calculate the value at $u(x_*, y_*)$?

Extension Cubic Polynomials in 2D



Cubic in y through

$u(x_*, y_0), u(x_*, y_0), u(x_*, y_0), u(x_*, y_0)$ to get $u(x_*, y_*)$

Cubic in x through

$u(x_0, y_0), u(x_1, y_0), u(x_2, y_0), u(x_3, y_0)$

to get $u(x_*, y_0)$

Cubic in x through

$u(x_0, y_1), u(x_1, y_1), u(x_2, y_1), u(x_3, y_1)$

to get $u(x_*, y_1)$

Cubic in x through

$u(x_0, y_2), u(x_1, y_2), u(x_2, y_2), u(x_3, y_2)$

to get $u(x_*, y_2)$

Cubic in x through

$u(x_0, y_3), u(x_1, y_3), u(x_2, y_3), u(x_3, y_3)$

to get $u(x_*, y_3)$

$x_* = 0.5(x_1 + x_2),\ y_* = 0.5(y_1 + y_2),$ Example

$$u(x_*, y_0) = \frac{1}{16}(-u(x_0, y_0) + 9u(x_1, y_0) + 9u(x_2, y_0) - u(x_3, y_0))$$

$$u(x_*, y_1) = \frac{1}{16}(-u(x_0, y_1) + 9u(x_1, y_1) + 9u(x_2, y_1) - u(x_3, y_1))$$

$$u(x_*, y_2) = \frac{1}{16}(-u(x_0, y_2) + 9u(x_1, y_2) + 9u(x_2, y_2) - u(x_3, y_2))$$

$$u(x_*, y_3) = \frac{1}{16}(-u(x_0, y_3) + 9u(x_1, y_3) + 9u(x_2, y_3) - u(x_3, y_3))$$

then

$$u(x_*, y_*) = \frac{1}{16}(-u(x_*, y_3) + 9u(x_*, y_3) + 9u(x_*, y_3) - u(x_*, y_3))$$

Notes

- In 2D with cubics we need 4 + 1 interpolations
- In 3D with cubics we need 4x(4+1) + 1 interpolations

Triangle Interpolation – Method 1

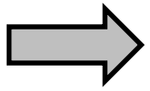
- Given the constraints of 3 points:

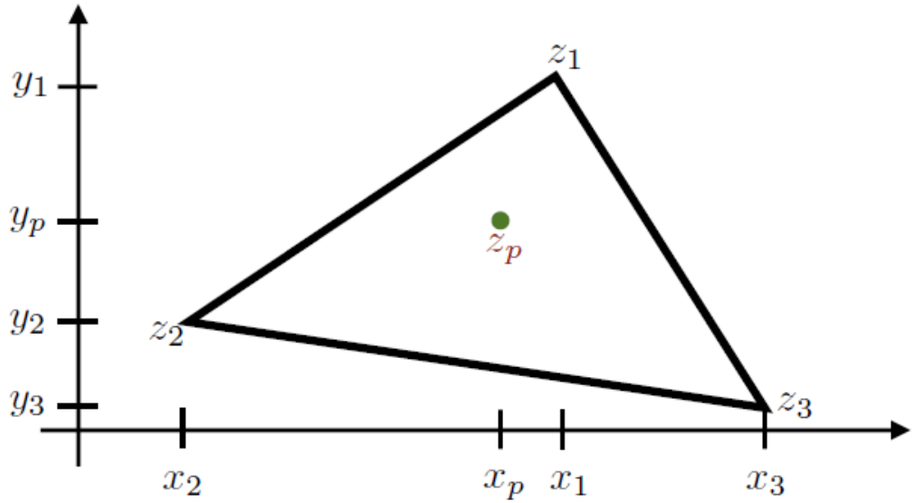
$$f(x_1, y_1) = z_1, f(x_2, y_2) = z_2, f(x_3, y_3) = z_3$$

- Find the linear equation satisfying :

$$z(x, y) = A + B\mathrm{x} + C\mathrm{y}$$

- We know…
  - $A + B\mathrm{x}_1 + Cy_1 = z_1$
  - $A + B\mathrm{x}_2 + Cy_2 = z_2$
  - $A + B\mathrm{x}_3 + Cy_3 = z_3$





Triangle Interpolation – Method 1

- Given:

$$A + B\mathrm{x}_1 + Cy_1 = z_1$$
$$A + B\mathrm{x}_2 + Cy_2 = z_2$$
$$A + B\mathrm{x}_3 + Cy_3 = z_3$$

- Is equivalent to:

$$\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

- Or:

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}^{-1} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

Triangle Interpolation – Method 1 Solve

$$\begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_{31} \end{pmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

Subtract row1 from rows 2 and 3

$$\begin{pmatrix} 1 & x_1 & y_1 \\ 0 & x_2 - x_1 & y_2 - y_1 \\ 0 & x_3 - x_1 & y_3 - y_1 \end{pmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 - z_1 \\ z_3 - z_1 \end{bmatrix}$$

Now subtract a multiple of row 2 from row 3

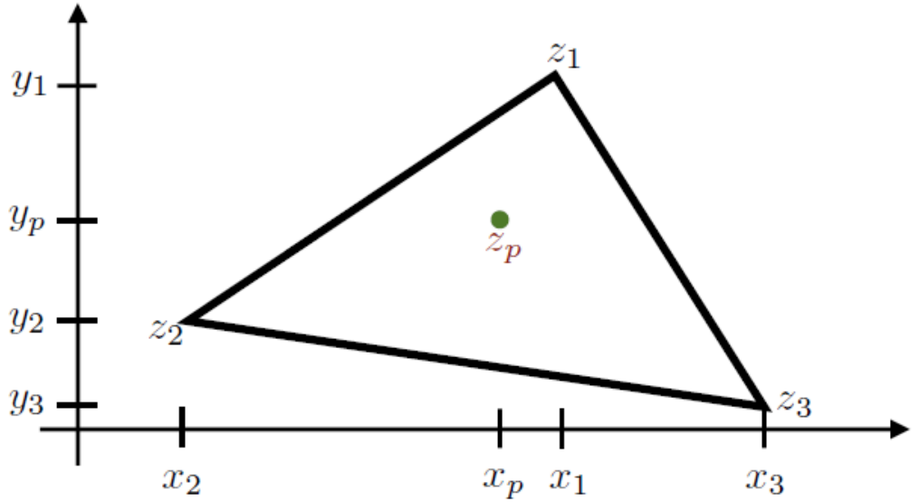
$$\begin{pmatrix} 1 & x_1 & y_1 \\ 0 & x_2 - x_1 & y_2 - y_1 \\ 0 & 0 & y_3 - y_1 - (y_2 - y_1)\dfrac{(x_3 - x_1)}{(x_2 - x_1)} \end{pmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 - z_1 \\ z_3 - z_1 - (z_2 - z_1)\dfrac{(x_3 - x_1)}{(x_2 - x_1)} \end{bmatrix}$$

Now we can solve for C then for B and finally for A to get

$$z(x,y) = A + Bx + Cy$$

Triangle Interpolation – Method 2 Barycentric* Interpolation

- Define a set of weights $(\omega_1, \omega_2, \omega_3)$, which measure closeness of a particular point (x_p, y_p),value z_p to the corners:
  - $x_p = x_1\omega_1 + x_2\omega_2 + x_3\omega_3$
  - $y_p = y_1\omega_1 + y_2\omega_2 + y_3\omega_3$
  - $z_p = z_1\omega_1 + z_2\omega_2 + z_3\omega_3$



- Constrain $\omega_1 + \omega_2 + \omega_3 = 1$, so $\omega_3 = 1 - \omega_1 - \omega_2$
  - $x_p = x_1\omega_1 + x_2\omega_2 + x_3(1 - \omega_1 - \omega_2)$
  - $y_p = y_1\omega_1 + y_2\omega_2 + y_3(1 - \omega_1 - \omega_2)$

*center of mass

Triangle Interpolation – Method 3 Barycentric Interpolation

- Using:

$$x_p = x_1\omega_1 + x_2\omega_2 + x_3(1 - \omega_1 - \omega_2)$$
$$y_p = y_1\omega_1 + y_2\omega_2 + y_3(1 - \omega_1 - \omega_2)$$

- Solve for the ω's:

$$\begin{bmatrix}\omega_1\\ \omega_2\end{bmatrix} = \begin{bmatrix}(x_1 - x_3) & (x_2 - x_3)\\ (y_1 - y_3) & (y_2 - y_3)\end{bmatrix}^{-1} \begin{bmatrix}(x_p - x_3)\\ (y_p - y_3)\end{bmatrix}$$

$$\omega_3 = 1 - \omega_1 - \omega_2$$

- Then compute the value $z_p\ from$:

$$z_p = z_1\omega_1 + z_2\omega_2 + z_3\omega_3$$

Triangle Interpolation – Method 3 Barycentric Interpolation

- Aside $\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1}$ = 1/(ad-bc) $\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

- Solve for the ω's:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} = 1/D \begin{bmatrix} (y_2 - y_3) & -(x_2 - x_3) \\ -(y_1 - y_3) & (x_1 - x_3) \end{bmatrix} \begin{bmatrix} (x_p - x_3) \\ (y_p - y_3) \end{bmatrix}$$

$$\text{D} = (x_1 - x_3)(y_2 - y_3) - (y_1 - y_3)(x_2 - x_3)$$

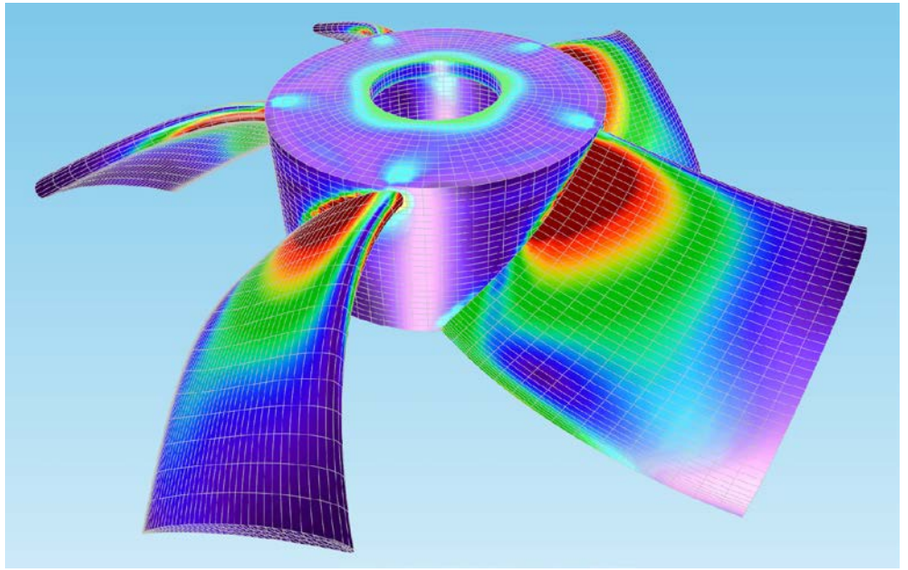
$$\omega_3 = 1 - \omega_1 - \omega_2$$

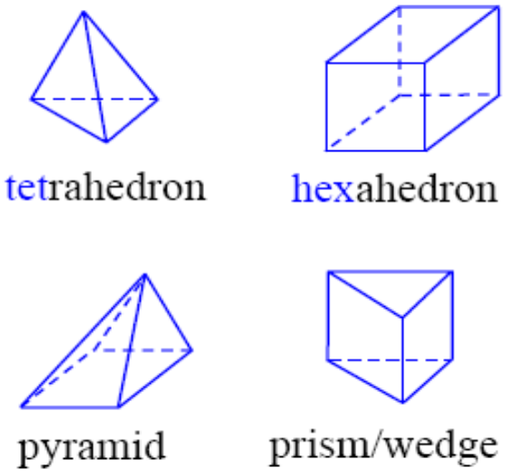
- Then compute the value $z_p \; from$:

$$z_p = z_1\omega_1 + z_2\omega_2 + z_3\omega_3$$

Summary

- Introduction to polynomials, piecewise polynomials and multi-dimensional polynomial approximations.
- Such approaches form the basis of many of the scientific and engineering computations today
- As floating point operations become relatively less expensive e.g. GPUs and Intel Xeon Phi there is a move towards higher order polynomial approximations on an element such as
- It is likely that there will be many more methods such as this in the future
- For example one possible next generation weather modeling code is based upon such methods





Recommended Reading

- An applet illustrating the benefit of using splines instead of a single polynomial:
  - http://terpconnect.umd.edu/~petersd/interp.html
- A more full-featured applet, with a wider range of interpolations methods but a little harder to use:
  - http://www.isaacdooley.com/java/ctd/index.html
- Notes and a wealth of other resources on interpolation methods, at http://numericalmethods.eng.usf.edu/:
  - Direct interpolation (linear, quadratic, cubic)
  - Lagrange interpolation
  - Spline interpolation

Newton's Divided-Difference Interpolating Polynomials (For reference only)

Linear Interpolation Is the simplest form of interpolation, connecting two data points with a straight line.

$$\frac{f_1(x) - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Linear-interpolation formula

$$f_1(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

$f_1(x)$ is a first-order interpolating polynomial

Newton's Divided-Difference Interpolating Polynomials (for reference only)

If three data points are available, A simple procedure can be used to determine the values of the quadratic coefficients:

$$f_2(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$

$$x = x_0 \qquad b_0 = f(x_0)$$

$$x = x_1 \qquad b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$x = x_2 \qquad b_2 = \frac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

General Form of Newton's Interpolating Polynomials (for reference only)

$$f_n(x) = f(x_0) + (x - x_0) f[x_1, x_0] + (x - x_0)(x - x_1) f[x_2, x_1, x_0]$$
$$+ \cdots + (x - x_0)(x - x_1) \cdots (x - x_{n-1}) f[x_n, x_{n-1}, \cdots, x_0]$$

$$b_0 = f(x_0)$$

$$b_1 = f[x_1, x_0]$$

$$b_2 = f[x_2, x_1, x_0]$$

$$\vdots$$

$$b_n = f[x_n, x_{n-1}, \cdots, x_1, x_0]$$

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j}$$

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k}$$

$$\vdots$$

$$f[x_n, x_{n-1}, \ldots, x_1, x_0] = \frac{f[x_n, x_{n-1}, \ldots, x_1] - f[x_{n-1}, x_{n-2}, \ldots, x_0]}{x_n - x_0}$$

Bracketed function evaluations are finite divided differences these are defined recursively as shown