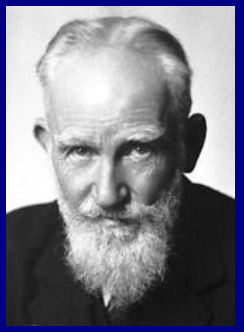
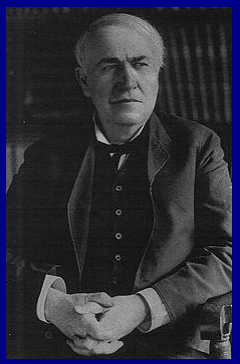
Polynomials and Interpolation

Selis Önel, PhD

Effort quotes

- Success is a ladder you cannot climb with your hands in your pockets. *~American Proverb*
- Be not afraid of going slowly; be afraid only of standing still. *~Chinese Proverb*
- When I was young, I observed that nine out of ten things I did were failures. So I did ten times more work. *~George Bernard Shaw, Irish playwright, 1856-1950*
- Opportunity is missed by most people because it is dressed in overalls and looks like work. *~Thomas Edison, American inventor and businessman, 1847-1931*
- All the so-called "secrets of success" will not work unless you do. *~Author Unknown*





When do we need data/function interpolation?

- Need to obtain estimates of function values at other points
- Need to use the closed form representation of the function as the basis for other numerical techniques

Goals

- Fit a polynomial to values of a function at discrete points to estimate the functional values between the data points
- Derive numerical integration schemes by integrating interpolation polynomials
  - Power series
  - Lagrange interpolation forms
- Differentiation and integration of interpolation polynomials
- Interpolation polynomials using nonequispaced points → Chebyshev nodes (roots of the Chebyshev polynomial of the 1st kind)

Polynomials

- Power series form:

  $y=c_1x^n+c_2x^{n-1}+\ldots+c_nx+c_{n+1}$

  n : order of polynomial

  c_i : coefficients
- Cluster form:

  $y=((\ldots((c_1x+c_2)x+c_3)x\ldots+c_n)x+c_{n+1})$
- Factorized form:

  $y=c_1(x-r_1)(x-r_2)\ldots(x-r_n)$

  r_i= roots of polynomial

Ex: Polynomials

- Power series form:

  $y=x^4+2x^3-7x^2-8x+12$
- Cluster form:

  $y=((((x+2)x-7)x-8)x+12)$
- Factorized form:

  $y= (x-1)(x-2)(x+2)(x+3)$

Polynomials

- A polynomial of order n has n roots:
  - Multiple roots
  - Complex roots
  - Real roots
- If all c_i are real, all the complex roots are found in complex conjugate pairs

Polynomials in MATLAB®

% $y=4x^4+2x^3-7x^2+x+7=0$
% p = [4 2 -7 1 7]
% **roots**: Finds the roots of a polynomial

```
>> p = [4 2 -7 1 7]
p =  4    2    -7    1    7
>> x=roots(p)
x =
  0.93158276438947 + 0.60071610876714i
  0.93158276438947 - 0.60071610876714i
 -1.18158276438947 + 0.16770340687492i
 -1.18158276438947 - 0.16770340687492i
```

Polynomials in MATLAB®

% $y=4x^4+2x^3-7x^2+x+7$
% **poly**: Determines the coefficients of the original polynomial knowing the roots.
% The polynomial is normalized to make the leading coefficient 1

```
>> r=[  0.93158276438947 + 0.60071610876714i
         0.93158276438947  - 0.60071610876714i
        -1.18158276438947 + 0.16770340687492i
        -1.18158276438947  - 0.16770340687492i];
>> p=poly(r); p'
ans =
    1.00000000000000
    0.50000000000000
   -1.75000000000000
    0.25000000000000
    1.74999999999999
```

(4) →

4
2
-7
1
7

Accuracy of Conversions

Conversions may not be accurate due to rounding errors in computations

Coefficients → Roots

Roots → Coefficients

- Multiple roots: Less accurate conversion

"Computing a highly multiple root is one of the most difficult problems for numerical methods"[1]

Ex:

$y=(x-5)^5$ (This equation can be expanded symbolically using simple(y) or

```
>> expand(y)
y=x^5-25*x^4+250*x^3-1250*x^2+3125*x-3125
>> p=sym2poly(y)
p =    1        -25        250       -1250        3125       -3125
>> roots(p)
ans =
  5.00482653710827 + 0.00350935026218i
  5.00482653710827 - 0.00350935026218i
  4.99815389493583 + 0.00567044756022i
  4.99815389493583 - 0.00567044756022i
  4.99403913591176
```

Accuracy of Conversions

Use the ***round*** and ***real*** functions to get **integer** results in taking the roots of polynomials:

```
>>round(real(5.0048+0.0035i))
ans =     5

>> p =    [1  -25   250  -1250  3125  -3125]
>> r=round(real(roots(p)));     r'
ans =
    5     5     5     5     5
```

Symbolic Calculations

```
>> syms x y z
>> sym2poly(4*x^4+2*x^3-7*x^2+x+7)
ans =     4     2    -7     1     7

>> poly2sym([4     2    -7     1     7])
ans = 4*x^4+2*x^3-7*x^2+x+7

>> f1=poly2sym([4     2    -7     1     7],sym('t'))
f1 = 4*t^4+2*t^3-7*t^2+t+7

>> horner(f1)
 ans = (((4*t+2)*t-7)*t+1)*t+7

>> f2=x^5-25*x^4+250*x^3-1250*x^2+3125*x-3125
>> factor(f2)
 ans = (x-5)^5
```

Symbolic Calculations to Numbers

Symbolic substitution:

subs(f): replaces all the variables in the symbolic expression f with values obtained from the calling function, or the MATLAB workspace

Ex:

```
>> f = (x-5)^5;
>> x=6;
>> subs(f)
ans =     1
>> subs(f,7)
ans =    32
```

subs(f,old,new) replaces old with new in the symbolic expression f

```
>> syms a
>> subs(a*x,a,5)
ans = 5*x
```

Polynomial Calculations in MATLAB®

polyval: Evaluates polynomials

Ex:

```
>> syms x
>> f=4*x^4+2*x^3-7*x^2+x+7;
>> p=sym2poly(f)
p =     4     2    -7     1     7
>> x=1;
>> fx=polyval(p,x)
fx =     7
>> fx=polyval(p,10)
fx =    41317
```

Polynomial Calculations in MATLAB®

polyvalm(p,x): Evaluates polynomial with matrix argument
x must be a square matrix

Ex:

```
>> syms x;    f=4*x^4+2*x^3-7*x^2+x+7;
>> p=sym2poly(f)
p =     4     2    -7     1     7

>> x=[1 10; 10 1];
>> fx=polyvalm(p,x)
fx = 42307       18090
      18090       42307
>> fx=polyval(p,x)
fx =     7       41317
      41317       7
```

Polynomial Calculations in MATLAB®

polyfit: Fits polynomial to data

p= polyfit(x,y,n): Finds the coefficients of a polynomial p(x) best in a least-squares sense

n: degree of polynomial

y: data

p: row vector of length n+1 (polynomial coefficients In descending powers)

A polynomial of order n is determined uniquely if n+1 data points (x_i,y_i), i=1,2,…,n+1 are given

Ex: Polynomial Calculations in MATLAB®

```
%Example for using the Polyfit function
x=[-2.4 -0.8 0.7 1.5 3.6];
y=[10 8 5 3 1.5];
figure(1)
plot(x,y,'*-'); grid on,  xlabel('x'); ylabel('y');
n=length(x)-1;
an=polyfit(x,y,n); an,
a3=polyfit(x,y,3);  a3,
a2=polyfit(x,y,2);   a2,
fitn=poly2sym(an);   fitn,
fit3=poly2sym(a3);   fit3,
fit2=poly2sym(a2);   fit2,
figure(2)
x1=-3:0.1:4;
plot(x,y,'*',x1,subs(fitn,x1),x1,subs(fit3,x1),x1,subs(fit2,x1));
xlabel('x'); ylabel('y'); grid on,
legend('data points','n=length(x)-1','n=3','n=2');
```

Ex: Polynomial Calculations in MATLAB®

Results of the Polyfit example in the command window:

an = 3.0510e-002 3.6803e-002 -3.5038e-001 -2.0526e+000 6.5885e+000

a3 = 9.2996e-002 -8.9465e-002 -2.2459e+000 6.3930e+000

a2 = 8.5356e-002 -1.6096e+000 5.9597e+000

fitn =

879390857921576 7/28823037615171174 4*x^4+5303925840414337/144115188075855872*x^3-1577961026311739/450359627370496*x^2-4621972395784683/2251799813685248*x+7418029510271909/1125899906842624

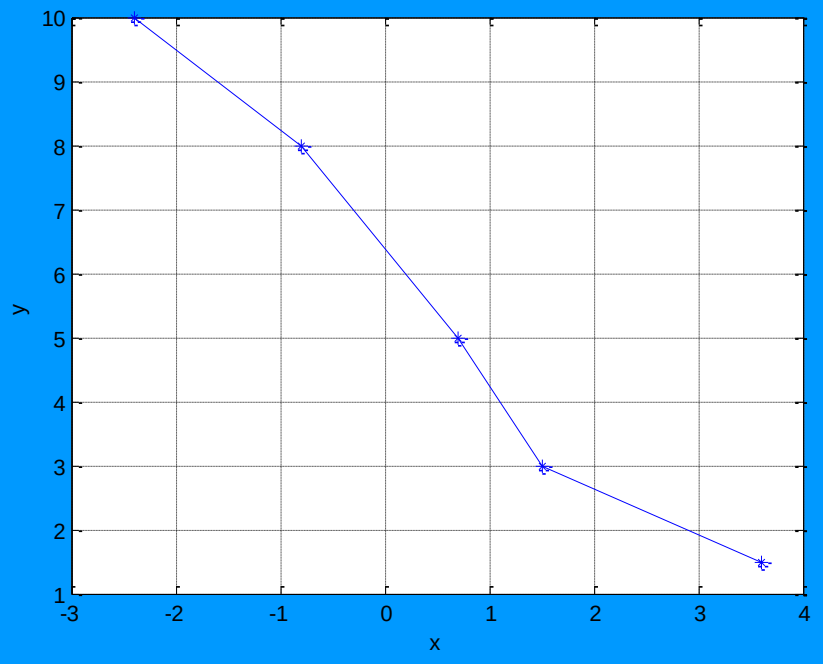
fit3 =

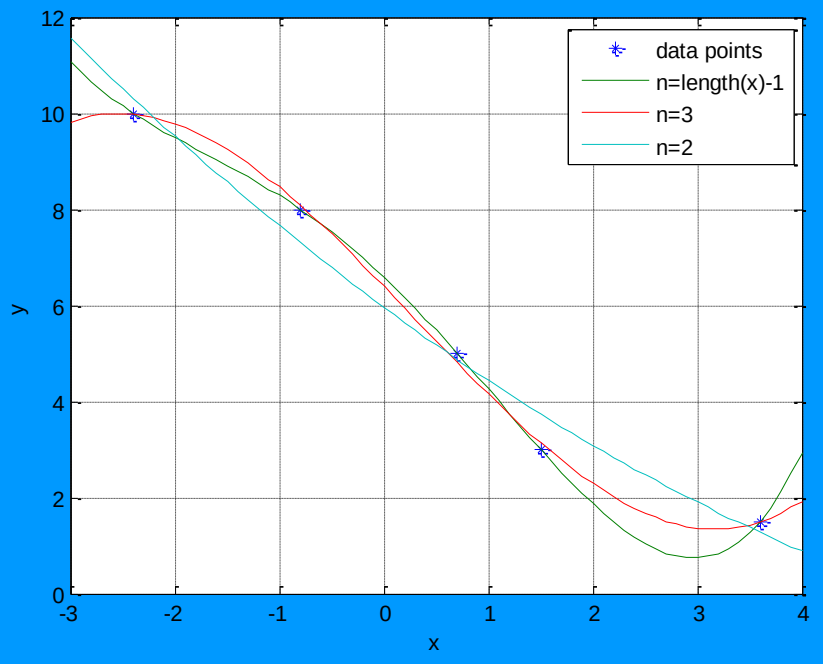
6701046569015015/7205759403792793 6*x^3-6446656677519129/7205759403792793 6*x^2-5057276482062349/2251799813685248*x+3598950055733773/562949953421312

fit2 =

6150542143648561/7205759403792793 6*x^2-3624406419963885/2251799813685248*x+6710023340029315/1125899906842624

Ex: Polynomial Calculations in MATLAB®





Integration of Polynomials

$y=c_1x^n+c_2x^{n-1}+\ldots+c_nx+c_{n+1}$

$$\int y dx = \frac{c_1}{n+1}x^{n+1} + \frac{c_2}{n}x^n + \ldots + \frac{c_n}{2}x^2 + c_{n+1}x + c_{n+2}$$

where c_{n+2} is an integration constant

If the coefficients of the polynomial are given in a row vector p:

- polyint: Integrates the polynomial analytically
- polyint(p,K) returns a polynomial representing the integral of polynomial p, using a scalar constant of integration K
- polyint(p) assumes a constant of integration K=0

Differentiation of Polynomials

$y=c_1x^n+c_2x^{n-1}+\ldots+c_nx+c_{n+1}$

$$\frac{dy}{dx} = nc_1x^{n-1} + (n-1)c_2x^{n-2} + \ldots + c_n$$

If the coefficients of the polynomial are given in a row vector p:

- polyder: Differentiates the polynomial
- polyder(p) returns the derivative of the polynomial whose coefficients are the elements of vector p
- polyder(A,B) returns the derivative of polynomial A*B
- [Q,D]=polyder(B,A) returns the numerator Q and denominator D of the derivative of the polynomial quotient B/A

Remember: Quotient Rule

Quotient rule, in calculus, is a method of finding the derivative of a function f(x) that is the quotient of two other functions g(x) and h(x), i.e., f(x)=g(x)/h(x) where h(x)≠0, for which derivatives exist

$$\text{If } f(x)=\frac{g(x)}{h(x)}, \text{ then } \frac{d}{dx}f(x)=f'(x)=\frac{g'(x)h(x)-g(x)h'(x)}{[h(x)]^2}=\frac{Q}{D}$$

Ex: Let $f(x)=(x^2-5)/(2x^3+4x+8)$

$$f(x)=\frac{(x^2-5)}{(2x^3+4x+8)}, \text{ then } \frac{d}{dx}f(x)=f'(x)=\frac{(2x)(2x^3+4x+8)-(x^2-5)(6x^2+4)}{(2x^3+4x+8)^2}$$

```
>> B=[1 0 -5]; A=[2 0 4 8];
>> [q,d]=polyder(B,A)
q =    -2     0    34    16    20
d =     4     0    16    32    16    64    64
```

Ex: Differentiation of Polynomials

```
>> syms x;  A=x^2+3*x+2; B=4*x^4+2*x^3-x^2+3*x+5;
>> A=sym2poly(A), B=sym2poly(B)
A =     1     3     2
B =     4     2    -1     3     5

>> polyder(A,B)
ans =    24    70    52    12    24    21
>> poly2sym(polyder(A,B))
ans =    24*x^5+70*x^4+52*x^3+12*x^2+24*x+21

>>[Q,D]=polyder(B,A)
Q =     8    38    44     6   -14    -9
D =     1     6    13    12     4
```

Product (Convolution) of Polynomials

Product of two polynomials of order m and order n gives a polynomial of order d=m+n

$$y_c = y_a y_b = c_1 x^d + c_2 x^{d-1} + \ldots + c_d x + c_{d+1}$$

- conv: Convolution and polynomial multiplication

C = conv(A,B) convolves vectors A and B

The resulting vector is length: length(A)+length(B)-1

If A and B are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials

Ex: Convolution of Polynomials

```
>> PA=[3 5 -1 0]; PB=[4 -7 21];
>> A=poly2sym(PA)
 A = 3*x^3+5*x^2-x
>> B=poly2sym(PB)
 B = 4*x^2-7*x+21
>> PC=conv(PA,PB)
PC =    12    -1    24   112   -21     0
>> C=poly2sym(PC)
 C = 12*x^5-x^4+24*x^3+112*x^2-21*x
```

Division of Polynomials

Division of a polynomial y_a by polynomial y_b satisfies

$$y_a = y_q y_b + y_r$$

y_q : quotient

y_r : remainder upon division

deconv: Deconvolution and polynomial division

[Q,R] = deconv(A,B) deconvolves vector B out of vector A. The result is returned in vector Q and the remainder in vector R such that A = conv(B,Q) + R

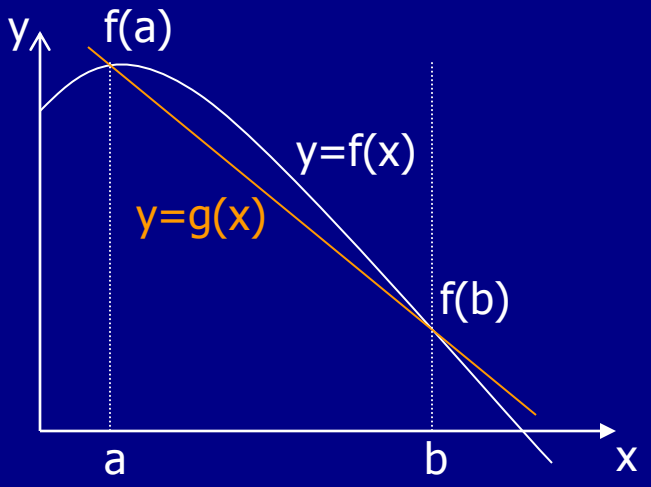
If A and B are vectors of polynomial coefficients, deconvolution is equivalent to polynomial division. The result of dividing A by B is quotient Q and remainder R

Ex: Deconvolution of Polynomials

```
>> PA=[3 5 -1 0]; PB=[4 -7 21];
>> A=poly2sym(PA)
 A = 3*x^3+5*x^2-x
>> B=poly2sym(PB)
 B = 4*x^2-7*x+21
>> PC=deconv(PA,PB)
PC =    0.75000000000000   2.56250000000000
>> C=poly2sym(PC)
C = 3/4*x+41/16
```

Linear Interpolation

- It is a line fitted to two data points
- Basis for many numerical schemes:
  - Integral of the linear interpolation: Trapezoidal Rule
  - Gradient of the linear interpolation: An approximation for the first derivative of the function



Lagrange form

$$g(x) = \frac{b-x}{b-a} f(a) + \frac{x-a}{b-a} f(b), \quad \text{or}$$

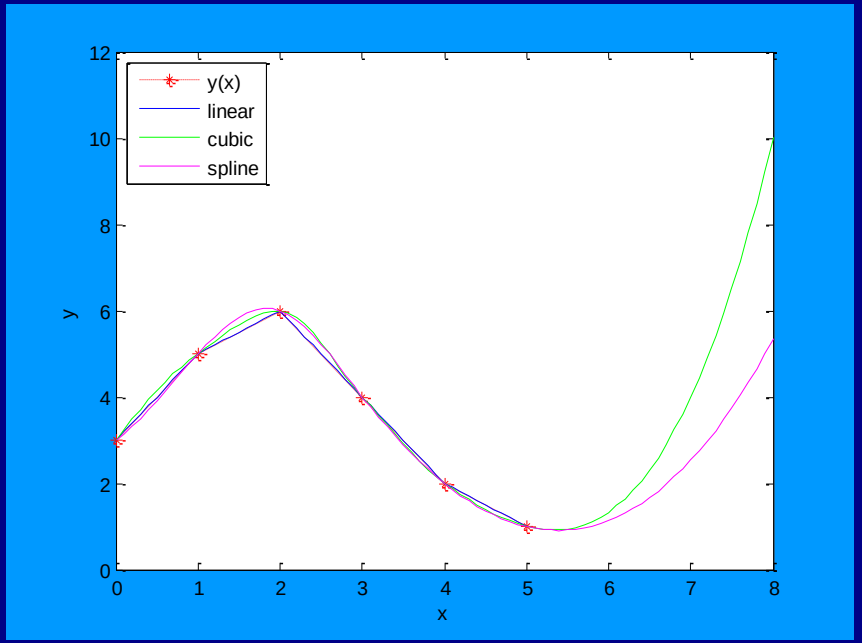
Newton form

$$g(x) = \frac{f(b)-f(a)}{b-a}(x-a) + f(a)$$

Interpolation in MATLAB®: interp1

- x need to be monotonic
- Cubic interpolation requires that x be equispaced

```
%interp1 does 1-D interpolation
x=[0 1 2 3 4 5]; y=[3 5 6 4  2 1];
xi=0:.1:8;
g=interp1(x,y,xi,'linear');
g1=interp1(x,y,xi,'cubic');
g2=interp1(x,y,xi,'spline');
plot(x,y,'*:r',xi,g,'b',xi,g1,'g',xi,g2,'m');
xlabel('x'); ylabel('y');
legend('y(x)','linear','cubic','spline',2);
```

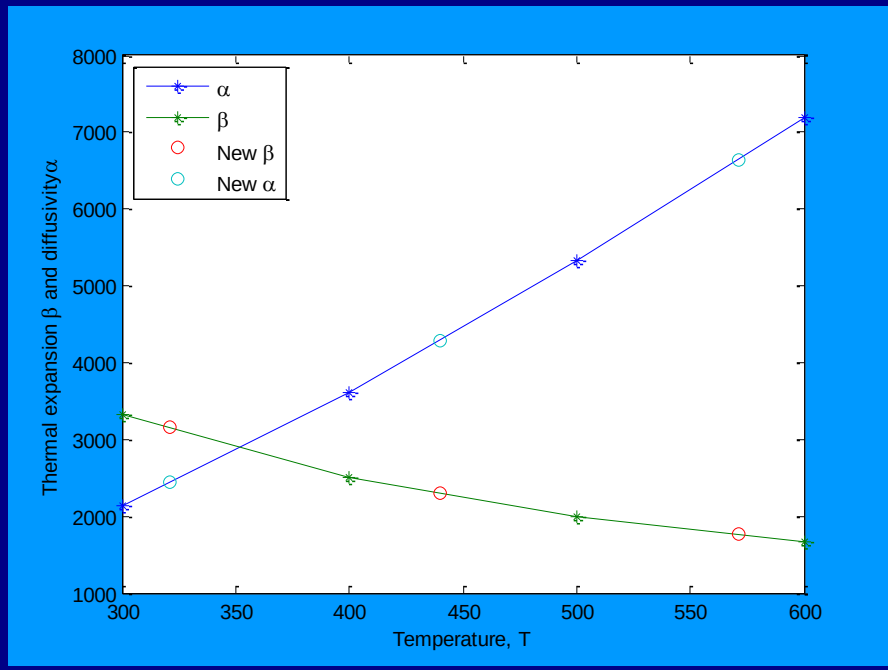


Interpolation in MATLAB®: interp1

```
%Calculate material properties of Carbon
%Values adapted from S. Nakamura, 2nd ed., p.169

T=[300 400 500 600]';
beta=[3330 2500 2000 1670]';
alpha=10^4*[.2128 .3605 .5324 .7190]';
Ti=[321 440 571]';
PropertyC=interp1(T,[beta,alpha],Ti,'linear');
[Ti PropertyC]
plot(T,alpha,'*-',T,beta,'*-',Ti,PropertyC(:,1),'o',Ti,PropertyC(:,2),'o')
legend('\alpha','\beta','New \beta','New \alpha',2);
xlabel('Temperature, T');
ylabel('Thermal expansion \beta and diffusivity \alpha');
```

```
ans =
  1.0e+003 *
    0.3210    3.1557    2.4382
    0.4400    2.3000    4.2926
    0.5710    1.7657    6.6489
```



Polynomial Interpolation with Power Series

Suppose n+1 data points are given as:

x_1 x_2 ... x_{n+1}

y_1 y_2 ... y_{n+1}

where

- x are abscissas of data points in increasing order

- the increment between x's is arbitrary

Polynomial of order n passing through n+1 data points may be written in power series form as

$$g(x) = c_1 x^n + c_2 x^{n-1} + \ldots + c_{n+1}$$

where c are coefficients

Setting $g(x_i)=y_i$ for n+1 data points gives n+1 linear equations, i.e.,

Ac=y

To find c:

- Solve Ac=y, i.e. c=A\y, OR

- Use polyfit(x,y,n)

Ex: Polynomial Interpolation with Power Series Unique solution

Determine the polynomial that passes through 3 data points:

(0,2), (1,1.5), (2,0.2)

Write the 2nd order polynomial as:

$g(x)= c_1x^2+c_2x+c_3$

Setting the polynomial at each data point gives:

$C_1(0)^2+c_2(0)+c_3=2$

$C_1(1)^2+c_2(1)+c_3=1.5$

$C_1(2)^2+c_2(2)+c_3=0.2$

Solving the above gives:

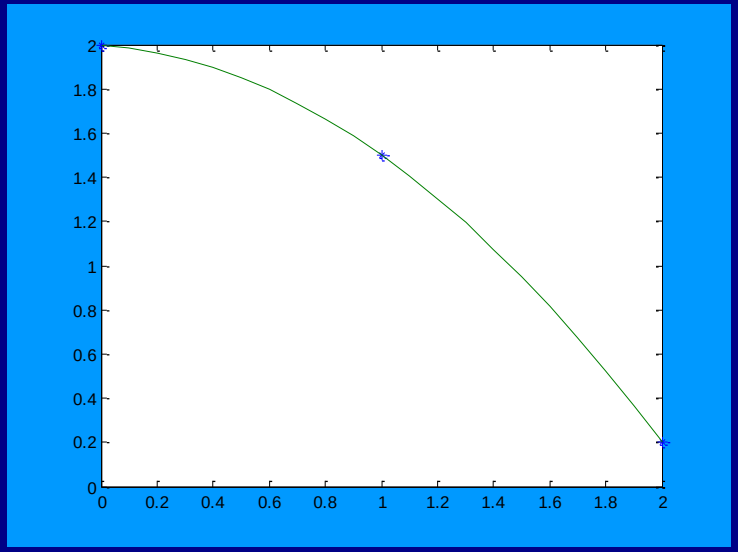
$c_3=2$, $c_2=-0.1$, $c_1=-0.4$

i.e.

$g(x)=-0.4x^2-0.2x+2$

```
>> A=[0 0 1;1 1 1; 4 2 1]; y=[2;1.5;0.2];
>> c=A\y; c'
ans =    -0.4000    -0.1000    2.0000

>> a=[0;1;2];  C=polyfit(a,y,2)
C =   -0.4000   -0.1000    2.0000
>>C=poly2sym(C); xi=0:.1:2;
>> Y=subs(C,xi);
>> plot(a,y,'*',xi,Y)
```



Lagrange Polynomial Interpolation

- The Lagrange form of the equation of a straight line passing through two points

$$p(x)=\frac{(x-x_2)}{(x_1-x_2)}y_1+\frac{(x-x_1)}{(x_2-x_1)}y_2$$

- The Lagrange form of the parabola passing through three points

$$p(x)=\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1+\frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2+\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3$$

Ex: Lagrange Interpolation Parabola

- The quadratic (2nd order) polynomial for three given data points

x_i	y_i (i=1:3)
-2	4
0	2
2	8

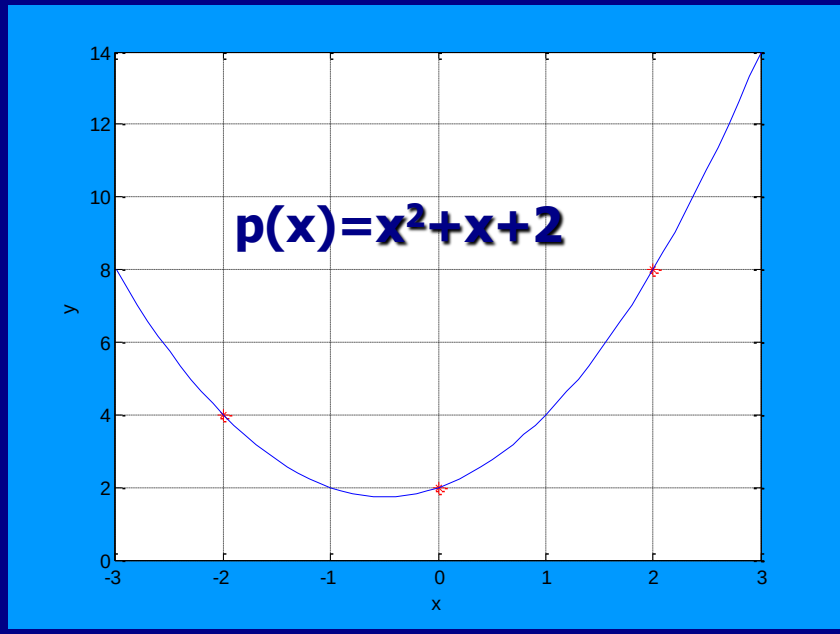
- Substituting into the Lagrange formula gives:

$$p(x)=\frac{(x-0)(x-2)}{(-2-0)(-2-2)}4+\frac{(x-(-2))(x-2)}{(0-(-2))(0-2)}2+\frac{(x-(-2))(x-0)}{(2-(-2))(2-0)}8$$

Which simplifies to:

$$p(x)=\frac{x(x-2)}{8}4+\frac{(x+2)(x-2)}{-4}2+\frac{x(x+2)}{8}8$$

$$= \mathbf{x^2+x+2}$$



Ex. Adopted from L. V. Fausett, 2nd ed., p.279

Newton Polynomial Interpolation

The Newton form of the equation of a straight line passing through two points

$p(x)=a_1+a_2(x-x_1)$

- The Newton form of the equation of a parabola passing through three points

$p(x)=a_1+a_2(x-x_1)+a_3(x-x_1)(x-x_2)$

$a_1=y_1 \quad a_2=(y_2-y_1)/(x_2-x_1) \quad a_3=\dfrac{(y_3-y_2)/(x_3-x_2)-(y_2-y_1)/(x_2-x_1)}{(x_3-x_1)}$

Ex: Newton Interpolation Parabola

- The quadratic (2nd order) polynomial for three given data points

x_i	y_i (i=1:3)
-2	4
0	2
2	8

- Substituting into the Newton formula gives:

$p(x)=a_1+a_2(x-(-2))+a_3(x-(-2))(x-0)$

Where the coefficients are:

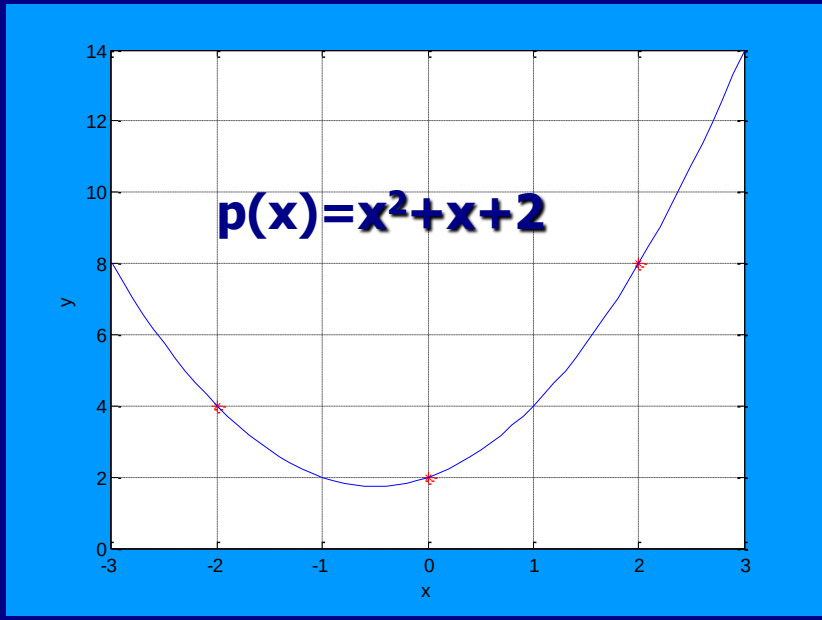
$a_1=y_1=4 \quad a_2=(y_2-y_1)/(x_2-x_1)=(2-4)/(0-(-2))=-1$

$a_3=[(y_3-y_2)/(x_3-x_2)-(y_2-y_1)/(x_2-x_1)]$

$=[(8-2)/(2-0)-(2-4)/(0-(-2))]/(2-(-2))=1$

$p(x)=4-(x+2)+x(x+2)$

$= \mathbf{x^2+x+2}$



Ex. Adopted from L. V. Fausett, 2nd ed., p.285

Advantages & Disadvantages

In many types of problems polynomial interpolation through moderate number of data points works very poorly

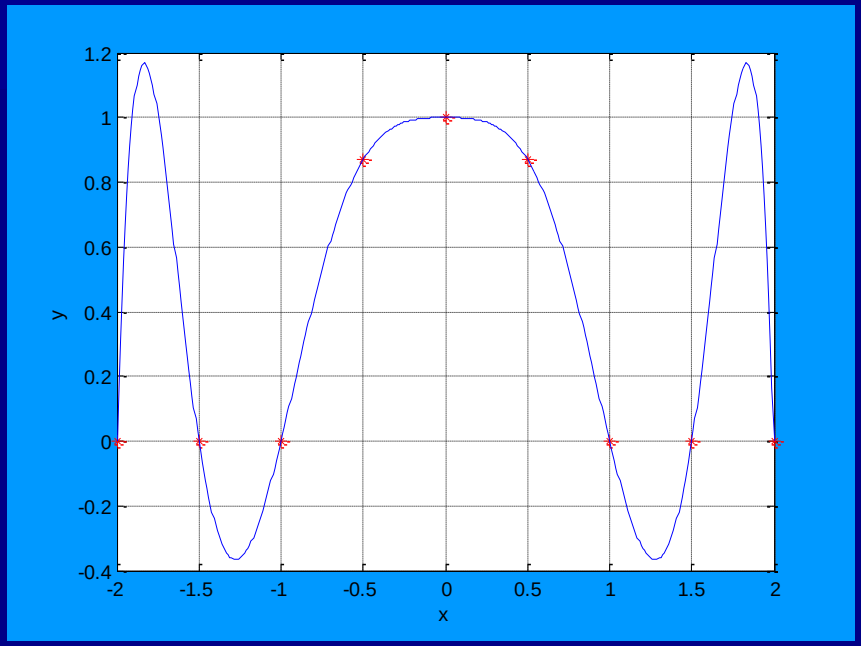
- Lagrange form:
  - Convenient when the values of x (independent variable) may be the same for different values of the corresponding y
  - Less convenient than the Newton form when
    - additional data points may be added to the problem
    - The appropriate degree of the interpolating polynomial is not known
- Newton form:
  - Convenient when the spacing between the x data values is constant
  - More data points can be incorporated and a higher degree polynomial can be generated by making use of the calculations for the lower order polynomial

Difficulties of Polynomial Interpolation

Humped and flat data if

- number of data points is moderately large
- the curve changes shape significantly over the interval

Then, there is difficulty with high-order polynomial interpolation



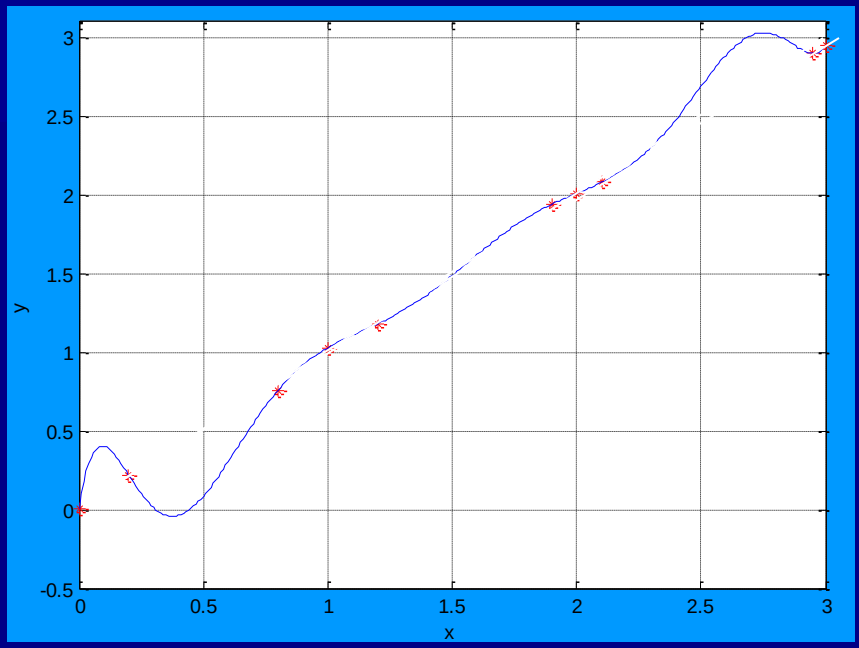

```
%Shows difficulty with high-order
%polynomial fit to humped and flat data
xi=[-2 -1.5 -1 -.5 0 .5 1 1.5 2];
yi=[0 0 0 .87 1 .87 0 0 0];
p=poly2sym(polyfit(xi,yi,8));
x=-2:0.01:2;
plot(xi,yi,'*r',x,subs(p,x));
xlabel('x'); ylabel('y'); grid on
```

Difficulties of Polynomial Interpolation

Noisy Straight Line if

- number of data points is moderately large
- Distance between x-values is not even

Then, there is difficulty with high-order polynomial interpolation




```
%Shows difficulty with high-order
%polynomial fit to noisy straight line data
xi=[0 .2 .8 1 1.2 1.9 2 2.1 2.95 3];
yi=[.01 .22 .76 1.03 1.18 1.94 2.01 2.08 2.9 2.95];
p=poly2sym(polyfit(xi,yi,9));
x=-0:0.01:3;
plot(xi,yi,'*r',x,subs(p,x));
xlabel('x'); ylabel('y'); grid on
```

Difficulties of Polynomial Interpolation

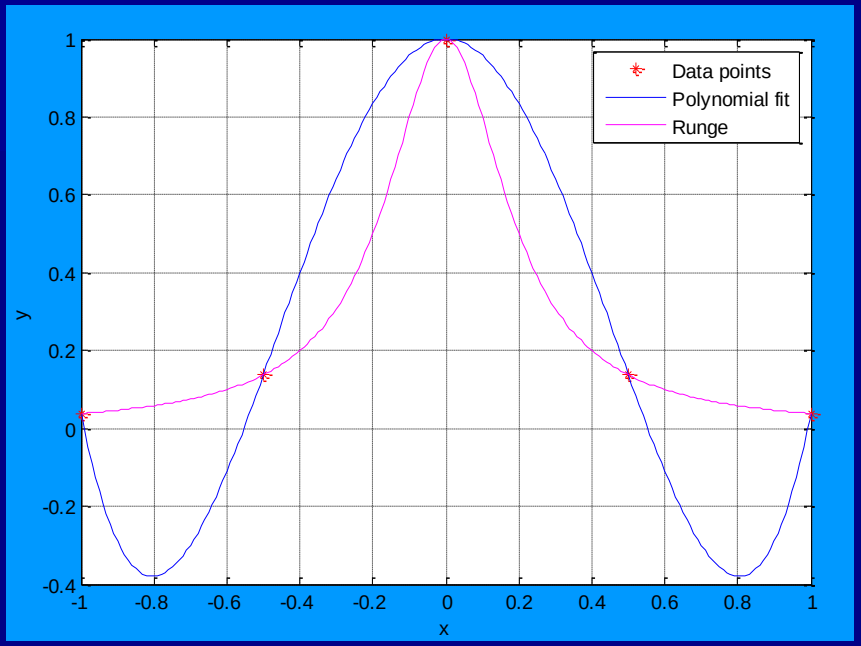
Runge function

$f(x)=(1+25x^2)^{-1}$

Using **five** equally spaced x values:

- Polynomial interpolation does not give a good approximation
- Using more function values at evenly spaced x-values is of no use

There is difficulty with high-order polynomial interpolation




```
%Shows difficulty with high-order
%polynomial fit to the Runge function
xi=[-1 -.5 0 .5 1];
yi=[.0385 .1379 1 .1379 .0385];
p=poly2sym(polyfit(xi,yi,4));
x=-1:0.01:1;
f=(1+25*x.^2).^(-1);
plot(xi,yi,'*r',x,subs(p,x),x,f,'m');
xlabel('x'); ylabel('y'); grid on
legend('Data points','Polynomial fit','Runge');
```

Difficulties of Polynomial Interpolation

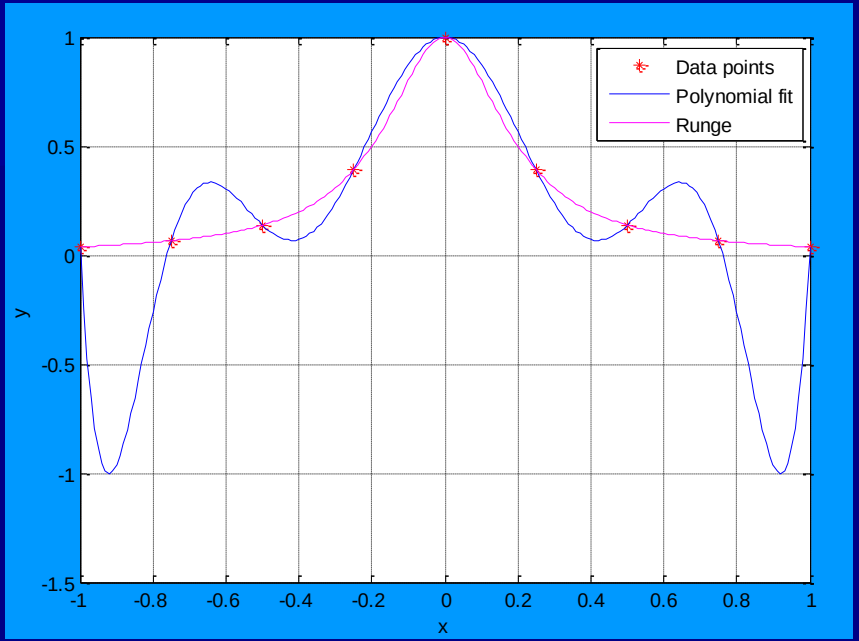
Runge function

$f(x)=(1+25x^2)^{-1}$

Using **nine** equally spaced x values:

- Interpolation polynomial (of order 8) gives a relatively better approximation compared to polynomial for order 4, but still overshoots the true function
- Using more function values at evenly spaced x-values is of no use

There is difficulty with high-order polynomial interpolation




```
%Shows difficulty with high-order
%polynomial fit to the Runge function
xi=[-1 -.75 -.5 -.25 0 .25 .5 .75 1];
yi=[.039 .066 .138 .39 1 .39 .138 .066 .039];
p=poly2sym(polyfit(xi,yi,8));
x=-1:0.01:1;
f=(1+25*x.^2).^(-1);
plot(xi,yi,'*r',x,subs(p,x),x,f,'m');
xlabel('x'); ylabel('y'); grid on
legend('Data points','Polynomial fit','Runge');
```

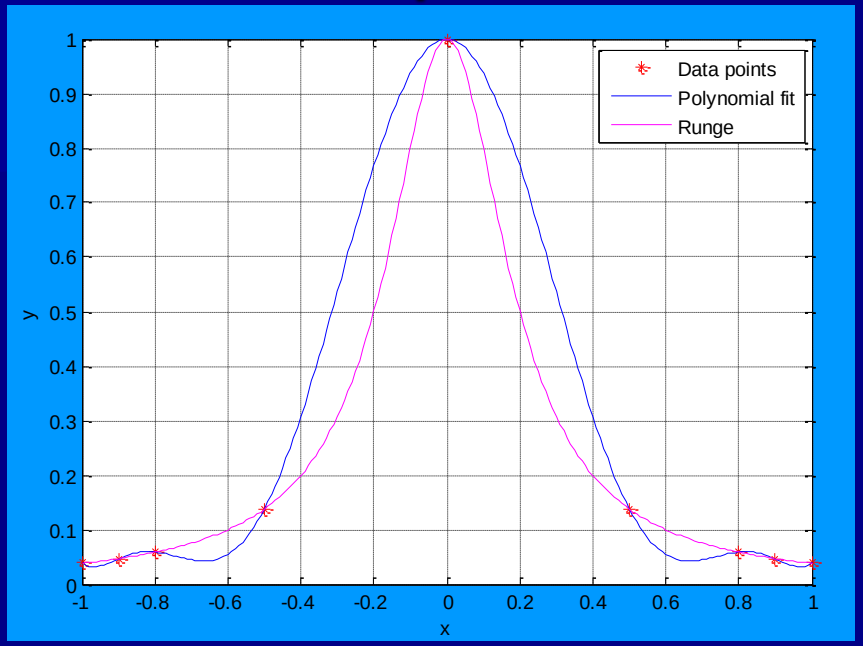
Difficulties of Polynomial Interpolation

Runge function

$f(x)=(1+25x^2)^{-1}$

Using **better distribution** of the data points, with more points towards the ends of the interval and fewer in the center:

- Gives better results
- Optimum interpolation (minimizing maximum deviation between the function and the interpolating polynomial) is achieved using zeros of the Chebyshev polynomial as the nodes




```
%Shows difficulty with high-order
%polynomial fit to the Runge function
xi=[-1 -.9 -.8 -.5 0 .5 .8 .9 1];
yi=[.039 .047 .059 .138 1 .138 .059 .047 .039];
p=poly2sym(polyfit(xi,yi,8));
x=-1:0.01:1;
f=(1+25*x.^2).^(-1);
plot(xi,yi,'*r',x,subs(p,x),x,f,'m');
xlabel('x'); ylabel('y'); grid on
legend('Data points','Polynomial fit','Runge');
```

Chebyshev Polynomials ?

- Sturm-Liouville Boundary Value Problem

$[p(x)y']'+[q(x)+\lambda r(x)]y=0$, BC.1 $a_1y(a)+a_2y'(a)=0$, BC.2 $b_1y(b)+b_2y'(b)=0$ has a special case where

$a=-1$, $b=1$, $p(x)=(1-x^2)^{1/2}$, $q(x)=0$, $r(x)=(1-x^2)^{-1/2}$, $\lambda=n^2$

is called Chebyshev's Differential Equation defined as:

$(1-x^2)y''-xy'+n^2y=0$ where n is a real number.

The solutions of this equation are called Chebyshev Functions of degree n

- If n is a non-negative integer, i.e., n=0,1,2,…, the Chebyshev Functions are often referred to as Chebyshev Polynomials $T_n(x)$
- $T_n(x)$ form a complete orthogonal set on the interval $-1\le x\le 1$ w.r.t. r(x)
- Using Rodrigues' Formula:

$$T_n(x)=\frac{\sqrt{1-x^2}}{(-1)^n(2n-1)(2n-3)\ldots 1}\frac{d^n}{dx^n}(1-x^2)^{n-\frac{1}{2}},\quad \text{where n=0,1,2,3,...}$$

- For more information, http://www.efunda.com/math/chebyshev/index.cfm

Chebyshev Polynomials ?

$$T_n(x) = \frac{\sqrt{1-x^2}}{(-1)^n(2n-1)(2n-3)\ldots 1}\frac{d^n}{dx^n}(1-x^2)^{n-\frac{1}{2}}, \quad \text{where n=0,1,2,3,...}$$

$T_0(x)=1$

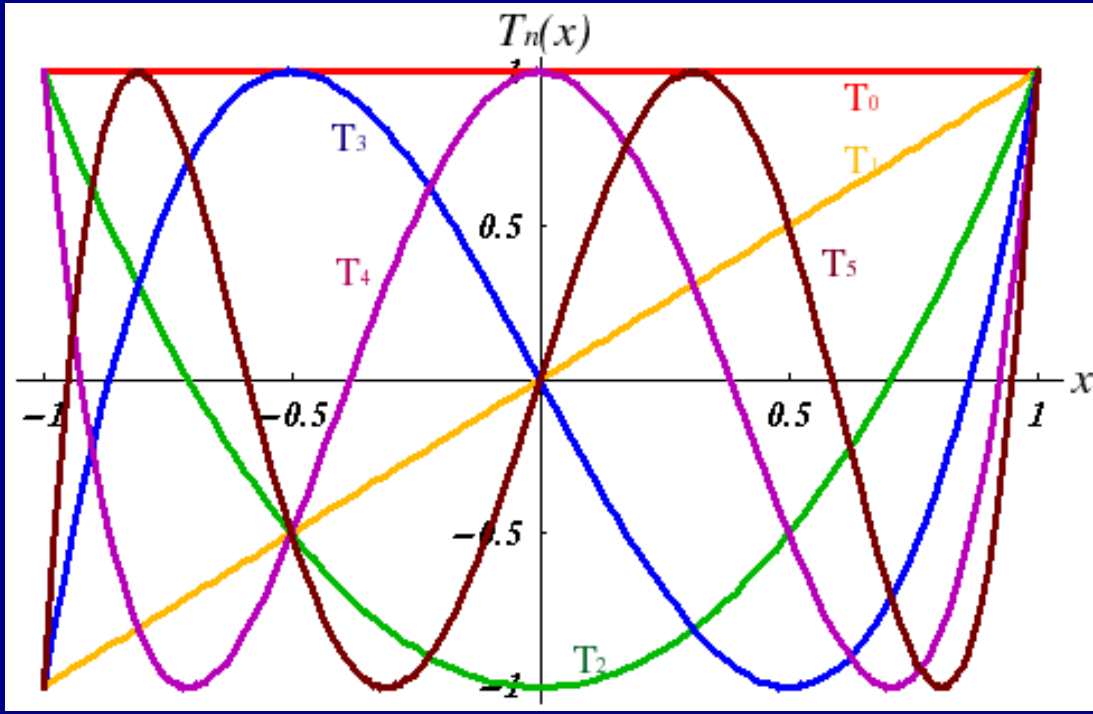
$T_1(x)=x-1$

$T_2(x)=2x^2-1$

$T_3(x)=4x^3-3x$

...

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



Ref: http://www.efunda.com/math/chebyshev/index.cfm

Hermite Interpolation

- Allows to find a polynomial that matches both the function values and some of the derivative values at specified values of the independent variable
- Simplest case: function values and first-derivative values are given at each point

Ex: Data for the position and velocity of a vehicle at several different times, i.e.

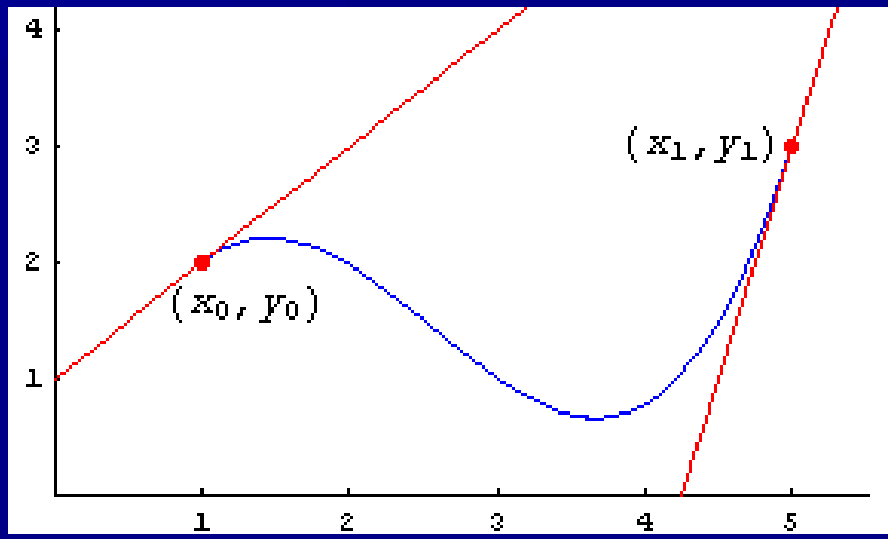
t (time), X (Position), v=dX/dt (Velocity)

Hermite Interpolation

- The cubic Hermite polynomial p(x) has the interpolative properties:

  $p(x_0)=y_0 \quad p(x_1)=y_1$

  $p'(x_0)=d_0 \quad p'(x_1)=d_1$
- Both the function values and their derivatives are known at the endpoints of the interval $[x_0,x_1]$.
- Hermite polynomials were studied by the French Mathematician Charles Hermite (1822-1901), and are referred to as a "clamped cubic," where "clamped" refers to the slope at the endpoints being fixed (see figure)



- If f[x] is continuous on $[x_0,x_1]$, there exists a unique cubic polynomial $p[x]=ax^3+bx^2+cx+d$ such that

  $p[x_0]=f[x_0]$

  $p[x_1]=f[x_1]$

  $p'[x_0]=f'[x_0]$

  $p'[x_1]=f'[x_1]$

Ref:http://math.fullerton.edu/mathews/n2003/HermitePolyMod.html

Hermite Interpolation

- Summary
  - Finds a polynomial that agrees with function values and their derivatives at the node points
  - Uses the Newton form and repeats it for the data points
  - For cubic Hermite polynomial on [0,1] the basis functions are:
    - $y_a=-0.5t^3+t^2-0.5t$
    - $y_b=1.5t^3-2.5t^2+1$
    - $y_c=-1.5t^3+2t^2+0.5t$
    - $y_d=0.5t^3-0.5t^2$

    if given: $f(0)=f_b$, $f(1)=f_c$, $f'(0)=s_a$, $f'(1)=s_b$

    define: $f_a=f_c-2s_a$, $f_d=2s_b+f_b$

    Then the required interpolating function on [0,1] is $y=f_a y_a+f_b y_b+f_c y_c+f_d y_d$
- Additional Information
  - The cubic Hermite polynomial is a generalization of both the Taylor polynomial and Lagrange polynomial, and it is referred to as an "osculating polynomial."
  - Hermite polynomials can be generalized to higher degrees by using more nodes $\{x_0, x_1,\ldots,x_n\}$ and extending the agreement at higher derivatives

    $p^{(k)}[x_i]=f^{(k)}[x_i]$ for $i=1,2,\ldots,n$ and $k=1,2,..m_i$

    Ref:http://math.fullerton.edu/mathews/n2003/HermitePolyMod.html

Piecewise Polynomial Interpolation

If there are large number of data points:

- Use piecewise polynomials instead of using a single polynomial (of high degree) to interpolate these points
- A spline of degree m is a piecewise polynomial (of degree m) with the maximum possible smoothness at each of the points where the polynomials join
- A linear spline is continuous
- A quadratic spline has continuous first derivatives
- A cubic spline has continuous second derivatives

Piecewise Linear Interpolation

Simplest form of piecewise polynomial interpolation:

- Consider set of four data points:

(x_1,y_1), (x_2,y_2), (x_3,y_3), (x_4,y_4) with $x_1<x_2<x_3<x_4$

- Defining three subintervals of the x-axis gives:

$I_1=[x_1,x_2]$, $I_2=[x_2,x_3]$, $I_3=[x_3,x_4]$

Subintervals join at the **knots**, which are **nodes** where the data values are given

Piecewise Linear Interpolation

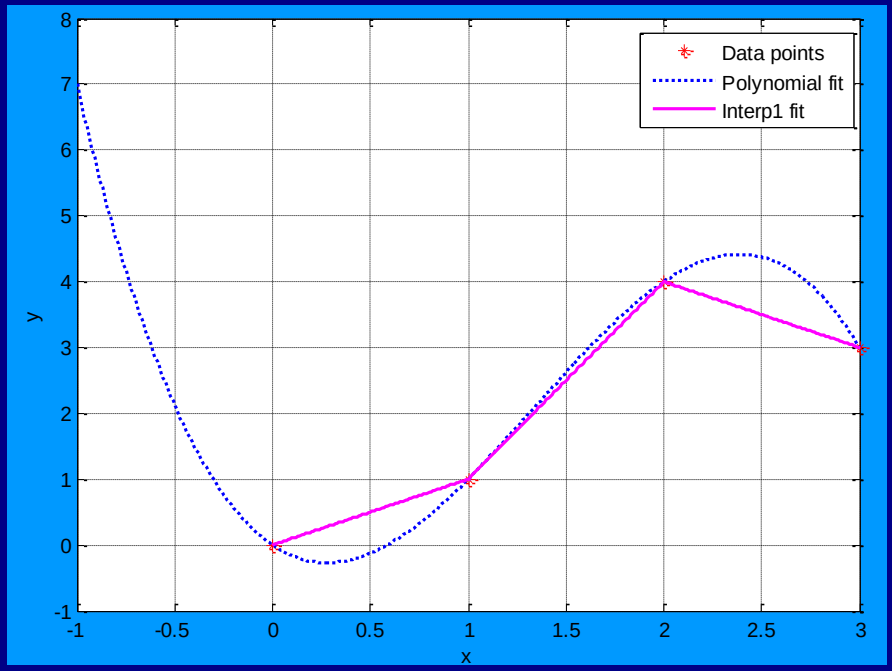
Using a straight line on each subinterval, the data can be interpolated using a piecewise linear function

$$P(x) = \begin{cases} \dfrac{x - x_2}{x_1 - x_2} y_1 + \dfrac{x - x_1}{x_2 - x_1} y_2 , & x_1 \le x \le x_2 \\ \dfrac{x - x_3}{x_2 - x_3} y_2 + \dfrac{x - x_2}{x_3 - x_2} y_3 , & x_2 \le x \le x_3 \\ \dfrac{x - x_4}{x_3 - x_4} y_3 + \dfrac{x - x_3}{x_4 - x_3} y_4 , & x_3 \le x \le x_4 \end{cases}$$

A piecewise linear interpolating function is continuous, but not smooth at the nodes.

Ex: Piecewise Linear Interpolation

```
%Shows linear piecewise interpolation
xi=[ 0 1 2 3]; yi=[0 1 4 3];
p=poly2sym(polyfit(xi,yi,3));
x=-1:0.01:3;
yp=subs(p,x);
ylin=interp1(xi,yi,x);
plot(xi,yi,'*r',x,yp,':',x,ylin,'m');
xlabel('x'); ylabel('y'); grid on
legend('Data points','Polynomial
    fit','Interp1 fit');
```



Piecewise Quadratic Interpolation

- Using a quadratic equation on each subinterval, the functions and their derivatives can be made to agree at the nodes
- For n+1 data points, there are 2n+n-1 equations:
  - n intervals and 3 unknown constant coefficients for each quadratic polynomial → 3*n unknowns
  - 2 equations for each interval (quadratic equations written for each of the end points of the intervals), i.e. (x_1,y_1) and (x_2,y_2) must satisfy the quadratic equation on the first interval
  - n-1 points at which the intervals meet where the derivatives of the parabolas on the adjacent intervals are required to be continuous
- As a result, there are 3n unknowns and 3n-1 equations
- There are possible approaches for defining one additional condition

Piecewise Quadratic Interpolation

I. "Knots"="Nodes" Approach

Define quadratic functions as

$S_i(x) = y_i + m_i(x - x_i) + \frac{m_{i+1} - m_i}{2(x_{i+1} - x_i)}(x - x_i)^2$ where m is the slope of the function at x_i

$S_i'(x) = m_i + (m_{i+1} - m_i)\frac{(x - x_i)}{(x_{i+1} - x_i)}$, $S_i'(x_i) = m_i$, $S_i'(x_{i+1}) = m_{i+1}$ Thus, continuity is satisfied

We need to know continuity of functions at the nodes. In particular at $x = x_{i+1}$ we need:

$$S_i(x_{i+1}) = y_i + m_i(x_{i+1} - x_i) + \frac{m_{i+1} - m_i}{2(x_{i+1} - x_i)}(x_{i+1} - x_i)^2 = y_{i+1}$$

Simplifying gives: $m_{i+1} = 2\frac{y_{i+1} - y_i}{x_{i+1} - x_i} - m_i$ Thus, knowing m_i, other slopes can be found

Ex: Piecewise Quadratic Interpolation (knots=nodes)

Data points: x=[0, 1, 2, 3] and y=[0, 1, 4, 3]

Set $m_1=0$

$m_2=2(y_2-y_1)/(x_2-x_1)-m_1=2(1-0)/(1-0)-0=2$

$m_3=2(y_3-y_2)/(x_3-x_2)-m_2=2(4-1)/(2-1)-2=4$

$m_4=2(y_4-y_3)/(x_4-x_3)-m_3=2(3-4)/(3-2)-4=-6$ So,

$$S_i(x)=y_i+m_i(x-x_i)+\frac{m_{i+1}-m_i}{2(x_{i+1}-x_i)}(x-x_i)^2 \text{ for each interval gives:}$$

$$S_1(x)=0+0(x-0)+\frac{2-0}{2(1-0)}(x-0)^2=x^2, \qquad 0\le x\le 1$$

$$S_2(x)=1+2(x-1)+\frac{4-2}{2(2-1)}(x-1)^2=1+2(x-1)+(x-1)^2, \qquad 1\le x\le 2$$

$$S_3(x)=4+4(x-2)+\frac{-6-4}{2(3-2)}(x-2)^2=4+4(x-2)-5(x-2)^2, \qquad 2\le x\le 3$$

In this "knots"="nodes" approach, the choice of slope at x_1 ($m_1=0$) influences the overall shape of the curve!

Piecewise Quadratic Interpolation

II. Alternative scheme to the knots=nodes approach

Take the knots as the midpoints between the nodes

(function values are given at the nodes)

For four data points: (x_1,y_1), (x_2,y_2), (x_3,y_3), (x_4,y_4)

Define the knots as: $z_1=x_1$, $z_2=(x_1+x_2)/2$, $z_3=(x_2+x_3)/2$, $z_4=(x_3+x_4)/2$, $z_5=x_4$

Spacing between consecutive data points:

$h_1=x_2-x_1$, $h_2=x_3-x_2$, $h_3=x_4-x_3$

Then

$z_2-x_1=h_1/2$, $z_3-x_2=h_2/2$, $z_4-x_3=h_3/2$, $z_2-x_2=-h_1/2$, $z_3-x_3=-h_2/2$, $z_4-x_4=-h_3/2$

Piecewise Quadratic Interpolation

II. Alternative scheme to the knots=nodes approach (continued)

- Define the quadratic polynomials for each interval as:

$P_1(x)=a_1(x-x_1)^2+b_1(x-x_1)+c_1, \qquad x \in [z_1,z_2]$

$P_2(x)=a_2(x-x_2)^2+b_2(x-x_2)+c_2, \qquad x \in [z_2,z_3]$

$P_3(x)=a_3(x-x_3)^2+b_3(x-x_3)+c_3, \qquad x \in [z_3,z_4]$

$P_4(x)=a_4(x-x_4)^2+b_4(x-x_4)+c_4, \qquad x \in [z_4,z_5]$

- For $x=x_k \rightarrow P_k(x_k)=c_k$ an additional interpolation condition $P_k(x_k)=y_k$ may be imposed, then

$c_k = y_k$ for k=1,2,3,4

- Imposing continuity conditions on the polynomials at the interior nodes gives 3 equations:

$P_1(z_2)=P_2(z_2)$: $h_1^2a_1-h_1^2a_2+2h_1b_1+2h_1b_2=4(y_2-y_1)$

$P_2(z_3)=P_3(z_3)$: $h_2^2a_2-h_2^2a_3+2h_2b_2+2h_2b_3=4(y_3-y_2)$

$P_3(z_4)=P_4(z_4)$: $h_3^2a_3-h_3^2a_4+2h_3b_3+2h_3b_4=4(y_4-y_3)$

Piecewise Quadratic Interpolation

II. Alternative scheme to the knots=nodes approach (continued)

- Imposing continuity conditions on the derivative of the polynomials

$P_i'(x)=2a_i(x-x_i)+b_i$ at the interior nodes gives 3 more equations:

$P_1'(z_2)=P_2'(z_2)$: $h_1a_1+h_1a_2+b_1-b_2=0$

$P_2'(z_3)=P_3'(z_3)$: $h_2a_2+h_2a_3+b_2-b_3=0$

$P_3'(z_4)=P_4'(z_4)$: $h_3a_3+h_3a_4+b_3-b_4=0$

- 6 equations and 8 unknown coefficients (a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4)

- For $P_k'(x)=2a_k(x-x_k)+b_k$ $\rightarrow$ b_1 and b_4 can be found by imposing conditions on the derivative values at the interval endpoints, x_1 and x_4:

Setting $P_1'(x_1)=0$ gives $b_1=0$, and setting $P_4'(x_4)=0$ gives $b_4=0$

Piecewise Quadratic Interpolation

II. Alternative scheme to the knots=nodes approach (continued)

- By setting the zero-slope conditions at the interval endpoints, for $b_1=0$ and $b_4=0$, the 3 quadratic and 3 derivative equations for the coefficients become:

$$a_1h_1^2 - a_2h_1^2 + 0 + 0 + 0 + 2b_2h_1 + 0 + 0 = 4(y_2-y_1)$$

$$0 + a_2h_2^2 - a_3h_2^2 + 0 + 0 + 2b_2h_2 + 2b_3h_2 + 0 = 4(y_3-y_2)$$

$$0 + 0 + a_3h_3^2 - a_4h_3^2 + 0 + 0 + b_32h_3 + 0 = 4(y_4-y_3)$$

$$a_1h_1 + a_2h_1 + 0 + 0 + 0 - b_2 + 0 + 0 = 0$$

$$0 + a_2h_2 + a_3h_2 + 0 + 0 + b_2 - b_3 + 0 = 0$$

$$0 + 0 + a_3h_3 + a_4h_3 + 0 + 0 + b_3 - 0 = 0$$

Ex: Piecewise Quadratic Interpolation

- Consider data points (0,0), (1,1), (2,4), (3,3). Set up the linear system of equations using piecewise quadratic interpolation with the knots placed at the midpoints of the data intervals. Determine the coefficients of the linear system. Write the interpolating piecewise polynomial for each data interval.

$$A=\begin{pmatrix} 1 & -1 & 0 & 0 & 2 & 0 \\ 0 & 1 & -1 & 0 & 2 & 2 \\ 0 & 0 & 1 & -1 & 0 & 2 \\ 1 & 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 & 1 & -1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}, \quad r=\begin{pmatrix} 4 \\ 12 \\ -4 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \text{c=A\r}, \quad c=\begin{pmatrix} 0.7429 \\ 1.7714 \\ -3.3714 \\ 2.4571 \\ 2.5143 \\ 0.9143 \end{pmatrix} \rightarrow \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ b_2 \\ b_3 \end{pmatrix}$$

$$P_1(x)=0.7429(x-0)^2, \qquad x\in[0.0,0.5]$$

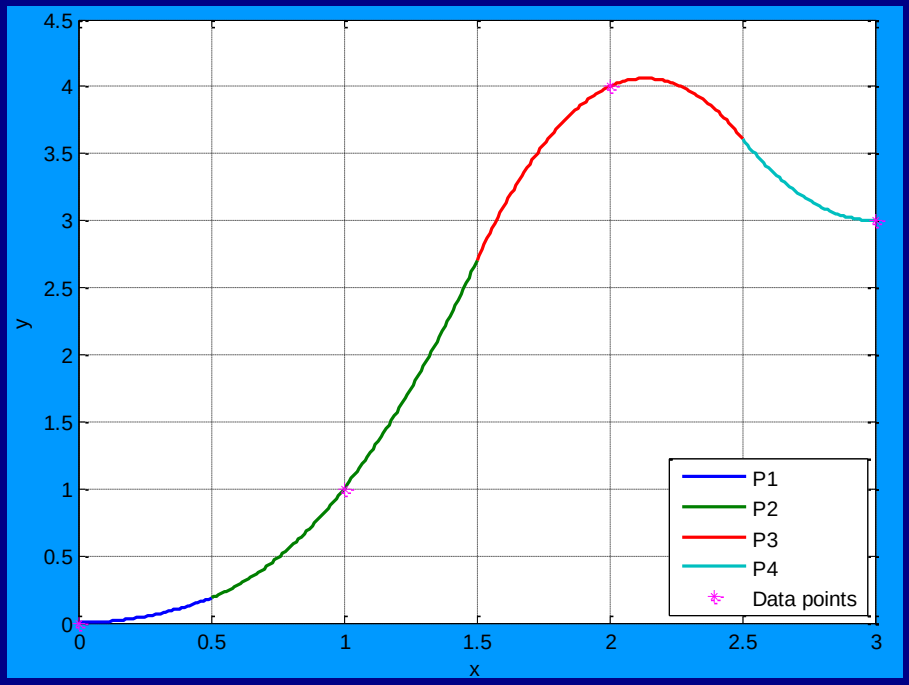
$$P_2(x)=1.7714(x-1)^2+2.5143(x-1)+1, \qquad x\in[0.5,1.5]$$

$$P_3(x)=-3.3714(x-2)^2+0.9143(x-2)+4, \qquad x\in[1.5,2.5]$$

$$P_4(x)=2.4571(x-3)^2+3, \qquad x\in[2.5,3.0]$$

Ex: Piecewise Quadratic (Spline) Interpolation

```
%plotting the quadratic piecewise polynomial solution
format short
syms x
xi=[0;1;2;3]; yi=[0;1;4;3];
A=[1 -1 0 0 2 0; 0 1 -1 0 2 2; 0 0 1 -1 0 2; 1 1 0 0 -1 0; 0 1 1 0 1 -1; 0 0 1 1 0 1];
r=[4;12;-4;0;0;0];
c=A\r;  %since b1=0 and b4=0:
c=[c(1); c(2); c(3); c(4); 0; c(5); c(6); 0];
n=length(xi);
for k=1:n,
    if k<n, h(k)=(xi(k)+xi(k+1))/2;end
   a(k)=c(k); b(k)=c(k+n); C(k)=yi(k);
   P(k)=c(k)*(x-xi(k))^2+b(k)*(x-xi(k))+C(k);
end
h1=[xi(1):.01:h(1)];  h2=[h(1):.01:h(2)];
h3=[h(2):.01:h(3)];  h4=[h(3):.01:xi(4)];
plot(h1,subs(P(1),h1),h2,subs(P(2),h2),h3,...
       subs(P(3),h3),h4,subs(P(4),h4),xi,yi,'*m')
legend('P1','P2','P3','P4','Data points',4);
xlabel('x'); ylabel('y');
grid on
```



Disadvantages of Quadratic Spline Interpolation

Even though better than the "nodes=knots" approach,

- it requires more computational effort to solve the linear system as the number of data points increase
- the coefficient matrix does not have a nice structure (not tridiagonal or banded), which would have reduced the computational effort

Ex: Piecewise Cubic Hermite Interpolation

- This method can be used to preserve monotonicity of x-data
- MATLAB® built-in function pchip

Define an interval xx, then the following commands provide in vector yy, the values of the interpolant at xx

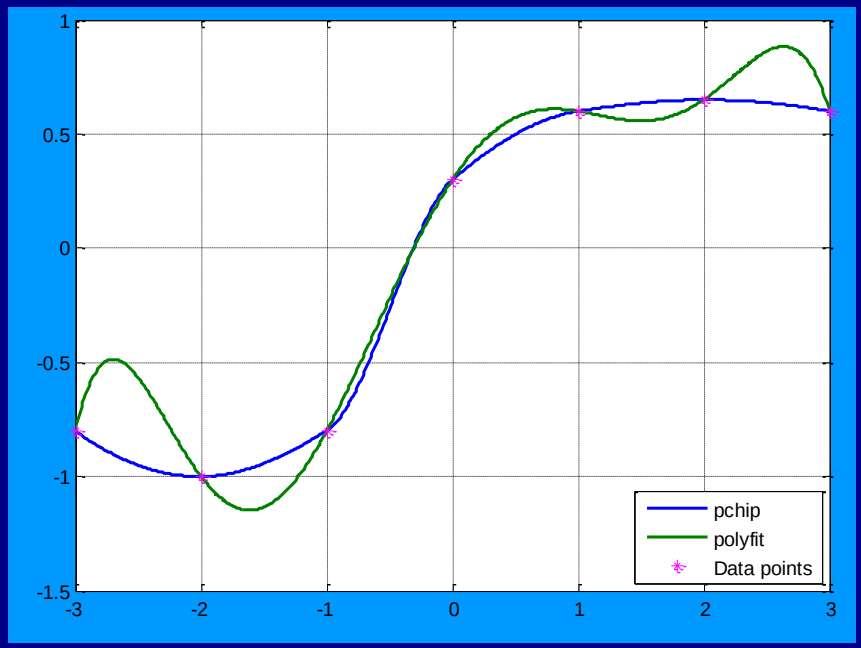
yy = pchip(x,y,xx), or yy = ppval(pchip(x,y),xx)

The pchip interpolating function, p(x), satisfies:

- On each subinterval, x(k) <= x <= x(k+1), p(x) is the cubic Hermite interpolant to the given values and certain slopes at the two endpoints
- Therefore, p(x) interpolates y, i.e., p(x(j)) = y(:,j), and the first derivative, p'(x), is continuous, but p''(x) is probably not continuous; there may be jumps at x(j)
- The slopes at x(j) are chosen in such a way that p(x) is "shape preserving" and "respects monotonicity". This means that, on intervals where the data is monotonic, so is p(x); at points where the data have a local extremum, so does p(x)

Ex: Piecewise Cubic Hermite Interpolation and Polyfit Interpolation

```
%Ex for piecewise cubic Hermite interpolation
xi=[-3 -2 -1 0 1 2 3];
yi=[-.8 -1 -.8 .3 .6 .65 .6];
x=[-3:.01:3];
hermitecoef=pchip(xi,yi);
yh=ppval(hermitecoef,x);
polynomialcoef=polyfit(xi,yi,(length(xi)-1));
yp=polyval(polynomialcoef,x);
plot(x,yh,x,yp,xi,yi,'*m')
legend('pchip','polyfit','Data points',4);
grid on
```



Cubic Spline Interpolation (Piecewise Cubic Polynomial)

- Better than other methods
- Requires continuity of the function as well as its first and second derivatives at each of the "knots" (boundaries of the subintervals)
- For n knots, $x_1<x_2<\ldots<x_i<\ldots<x_n$, define n-1 subintervals

$I_1=[x_1,x_2]$, …, $I_i=[x_i,x_{i+1}]$, …, $I_{n-1}=[x_{n-1},x_n]$

- Spacing between x values does not need to be uniform, so let

$h_i= x_{i+1}-x_i$

- On $I_i=[x_i,x_{i+1}]$, assume the cubic has the following form:

$$P_i(x) = a_i\frac{(x_{i+1}-x)^3}{h_i}+a_{i+1}\frac{(x-x_i)^3}{h_i}+b_i(x_{i+1}-x)+c_i(x-x_i)$$

- Continuity of the second derivative, $P_i''(x)$ is guaranteed by the form of this function

Cubic Spline Interpolation (Piecewise Cubic Polynomial), continued

$$P_i(x) = a_i\frac{(x_{i+1}-x)^3}{h_i}+a_{i+1}\frac{(x-x_i)^3}{h_i}+b_i(x_{i+1}-x)+c_i(x-x_i)$$

- Knowing that at $P_i(x_i)=y_i$ and $P_i(x_{i+1})=y_{i+1}$, b_i and c_i can be expressed in terms of a_i:

$b_i=(y_i/h_i)-a_ih_i$, $c_i=(y_{i+1}/h_i)-a_{i+1}h_i$

- Using the condition for continuity of P'(x) at the knots, obtain n-2 equations for the n unknowns $a_1,\ldots,a_n$

For i=1,…,n-2

$h_ia_i+2(h_i+h_{i+1})a_{i+1}+h_{i+1}a_{i+2}=(y_{i+2}-y_{i+1})/h_{i+1} - (y_{i+1}-y_i)/h_i$

- There are several choices for the conditions on P''(x) at the endpoints, which provide the additional conditions (equations) to determine all the unknowns
  - Simplest choice: Natural cubic spline assigns $P''(x_1)=0$ and $P''(x_n)=0$, which makes $a_1=a_n=0$

Cubic Spline Interpolation (Piecewise Cubic Polynomial), continued

$$P_i(x) = a_i\frac{(x_{i+1}-x)^3}{h_i}+a_{i+1}\frac{(x-x_i)^3}{h_i}+b_i(x_{i+1}-x)+c_i(x-x_i)$$

- For n=6 the resulting equations are:

$$2(h_1+h_2)a_2 + h_2a_3 = (y_3-y_2)/h_2-(y_2-y_1)/h_1$$

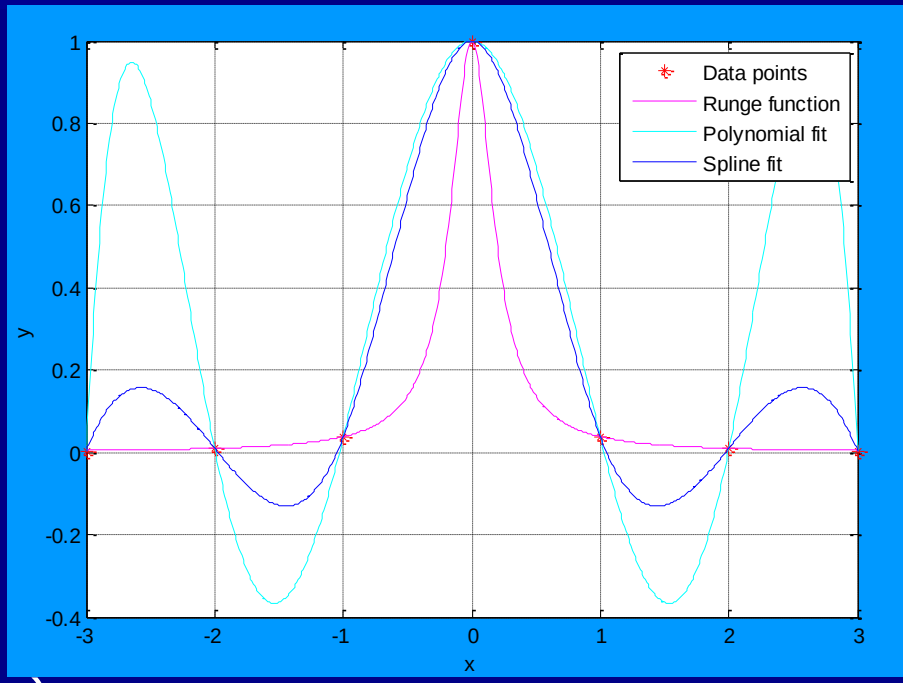
$$h_2a_2 + 2(h_2+h_3)a_3 +h_3a_4 = (y_4-y_3)/h_3-(y_3-y_2)/h_2$$

$$h_3a_3 + 2(h_3+h_4)a_4 + h_5a_5 = (y_5-y_4)/h_4-(y_4-y_3)/h_3$$

$$h_4a_4 + 2(h_4+h_5)a_5 = (y_6-y_5)/h_5-(y_5-y_4)/h_4$$

MATLAB® Ex: Spline

```
%Shows difficulty with high-order
%polynomial fit to the Runge function
syms a;
f=(1+25*a.^2).^(-1);
xi=[-3 -2 -1 0 1 2 3];
yi=subs(f,xi);
yp=poly2sym(polyfit(xi,yi,6));
x=-3:0.01:3;
y=subs(f,x);
ys=spline(xi,yi,x);
plot(xi,yi,'*r',x,y,'m',x,subs(yp,x),'c',x,ys);
xlabel('x'); ylabel('y'); grid on
legend('Data points','Runge function','Polynomial
    fit','Spline fit');
```

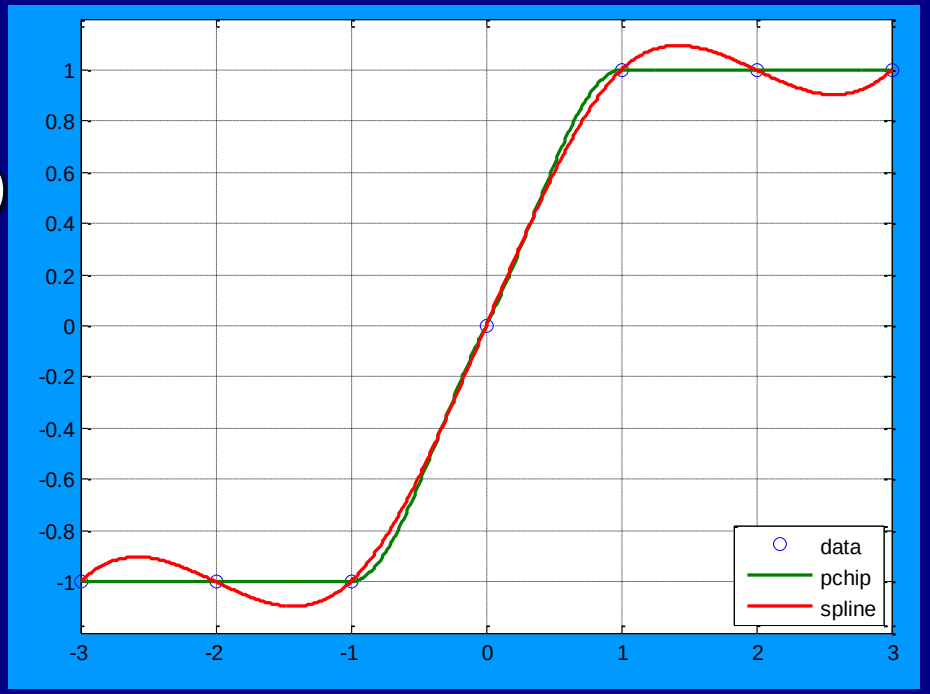


Comparing PCHIP with SPLINE

- The function s(x) supplied by SPLINE is constructed in exactly the same way, except that the slopes at the x(j) are chosen differently, namely to make even s”(x) continuous. This has the following effects:
  - SPLINE is smoother, i.e., s”(x) is continuous
  - SPLINE is more accurate if the data are values of a smooth function
  - PCHIP has no overshoots and less oscillation if the data are not smooth
  - PCHIP is less expensive to set up
  - The two are equally expensive to evaluate

Ex: Comparing PCHIP with SPLINE

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



Ref: Example is adopted from MATLAB® Help

Symbolic Plot: ezplot

Ezplot (f) plots the function f(x) over the default domain -2*PI < x < 2*PI

Ezplot(f) plots the implicitly defined function f(x,y)=0 over the default domain -2*PI<x<2*PI and -2*PI<y<2*PI

ezplot(f,[A,B]) plots f(x) over A<x<B and f(x,y)=0 over A<x<B and A<y<B

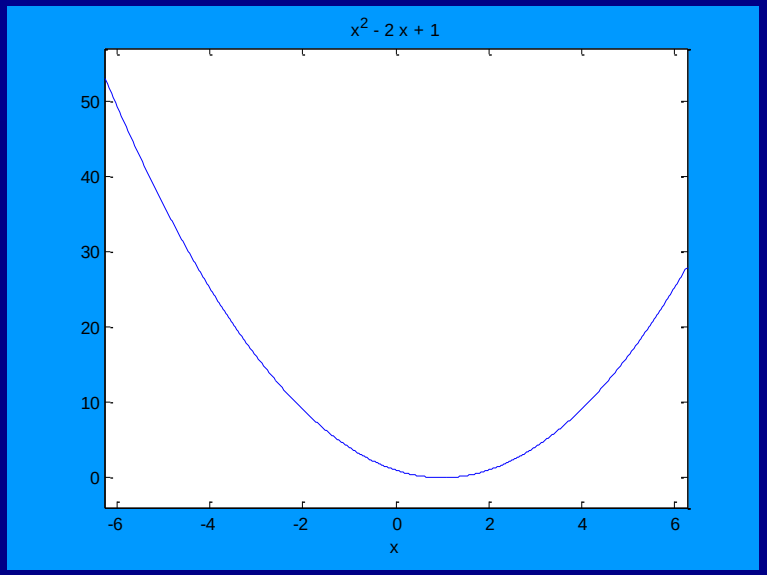
Using a string to express the function:

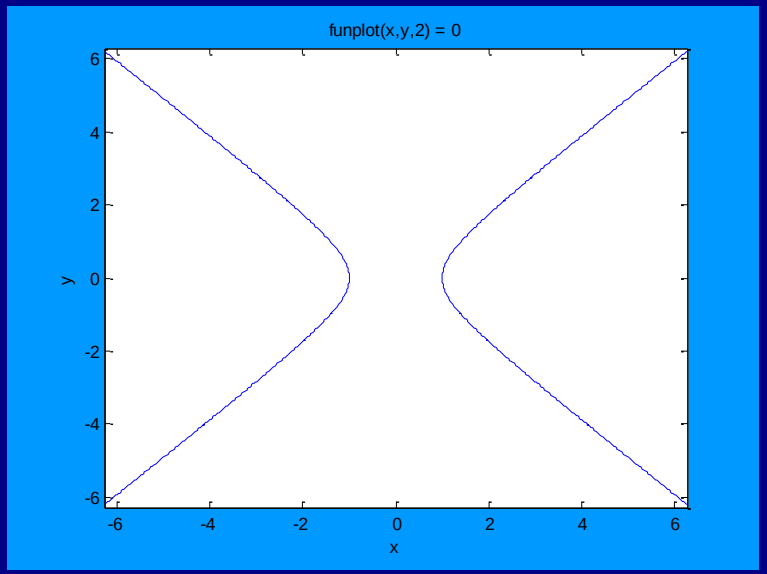
```
>> ezplot('x^2 - 2*x + 1')
```

Using a function handle (in case there are other parameters, k, in the function)

```
%function for ezplot
function  z = funplot(x,y,k)
      z = x.^k - y.^k - 1;

>> for k= 1:10,
ezplot(@(x,y)funplot(x,y,k))
end
```





Other Web Sources of Information

- http://planetmath.org/encyclopedia/LectureNotesOnPolynomialInterpolation.html