Interpolation and Curve Fitting

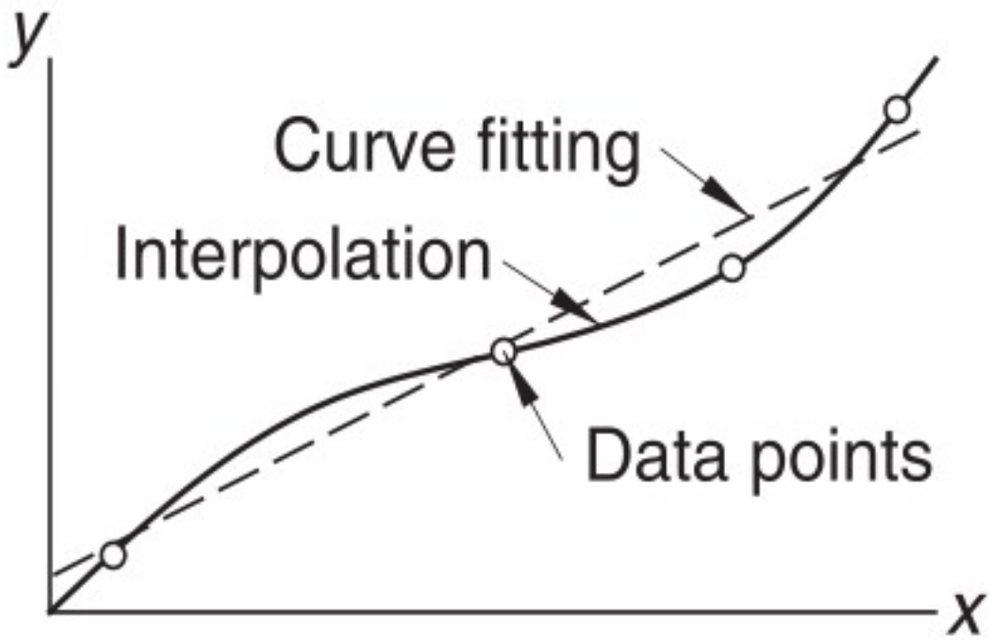
Discrete Data

x_1	x_2	x_3	$\cdots$	x_n
y_1	y_2	y_3	$\cdots$	y_n

Polynomial of order *n*

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

Intractable functions replaced by interpolating polynomials



Laplace's method: Unique polynomial of degree n-1 that can pass through *n* points

$$P_{n-1}(x) = \sum_{i=1}^{n} y_i \ell_i(x)$$

$$\ell_i(x) = \frac{x - x_1}{x_i - x_1} \cdot \frac{x - x_2}{x_i - x_2} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_n}{x_i - x_n}$$

$$= \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_i}{x_i - x_j}, \quad i = 1, 2, \ldots, n$$

Fitting predominantly done with polynomials

$$\ell_i(x_j) = \left\{ \begin{matrix} 0 \text{ if } i \neq j \\ 1 \text{ if } i = j \end{matrix} \right\} = \delta_{ij}$$

for *n* = 2, linear polynomial

$$\ell_1(x) = \frac{x - x_2}{x_1 - x_2} \qquad \ell_2(x) = \frac{x - x_1}{x_2 - x_1}$$

Obviously the polynomial passes through all the points

$$P_1(x) = y_1 l_1(x) + y_2 l_2(x)$$

error in polynomial interpolation

$$f(x) - P_{n-1}(x) = \frac{(x - x_1)(x - x_2) \cdots (x - x_n)}{n!} f^{(n)}(\xi)$$

Lagrange's method is straightforward, but does not lend itself to algorithm. Newton's method is better

Newton's Method

$$P_{n-1}(x) = a_1 + (x - x_1)a_2 + (x - x_1)(x - x_2)a_3 + \cdots + (x - x_1)(x - x_2)\cdots(x - x_{n-1})a_n$$

$$P_3(x) = a_1 + (x - x_1)a_2 + (x - x_1)(x - x_2)a_3 + (x - x_1)(x - x_2)(x - x_3)a_4$$

$$= a_1 + (x - x_1)\{a_2 + (x - x_2)[a_3 + (x - x_3)a_4]\}$$

$$P_0(x) = a_4$$

$$P_1(x) = a_3 + (x - x_3)P_0(x)$$

$$P_2(x) = a_2 + (x - x_2)P_1(x)$$

$$P_3(x) = a_1 + (x - x_1)P_2(x)$$

$$P_0(x) = a_n \qquad P_k(x) = a_{n-k} + (x - x_{n-k})P_{k-1}(x), \quad k = 1, 2, \ldots, n-1$$

```
function p = newtonPoly(a,xData,x)
% Returns value of Newton's polynomial at x.
% USAGE: p = newtonPoly(a,xData,x)
% a     = coefficient array of the polynomial;
%         must be computed first by newtonCoeff.
% xData = x-coordinates of data points.

n = length(xData);
p = a(n);
for k = 1:n-1;
    p = a(n-k) + (x - xData(n-k))*p;
end
```

Evaluation of Coefficients in Newton

$$y_i = P_{n-1}(x_i)$$

$$y_1 = a_1$$

$$y_2 = a_1 + (x_2 - x_1)a_2$$

$$y_2 = a_1 + (x_3 - x_1)a_2 + (x_3 - x_1)(x_3 - x_2)a_3$$

$$\vdots$$

$$y_n = a_1 + (x_n - x_1)a_1 + \cdots + (x_n - x_1)(x_n - x_2)\cdots(x_n - x_{n-1})a_n$$

Introducing divided differences

$$\nabla y_i = \frac{y_i - y_1}{x_i - x_1}, \quad i = 2, 3, \ldots, n$$

$$\nabla^2 y_i = \frac{\nabla y_i - \nabla y_2}{x_i - x_2}, \quad i = 3, 4, \ldots, n$$

$$\nabla^3 y_i = \frac{\nabla^2 y_i - \nabla^2 y_3}{x_i - x_3}, \quad i = 4, 5, \ldots n$$

$$\vdots$$

$$\nabla^n y_n = \frac{\nabla^{n-1} y_n - \nabla^{n-1} y_{n-1}}{x_n - x_{n-1}}$$

$$a_1 = y_1 \qquad a_2 = \nabla y_2 \qquad a_3 = \nabla^2 y_3 \qquad \cdots \qquad a_n = \nabla^n y_n$$

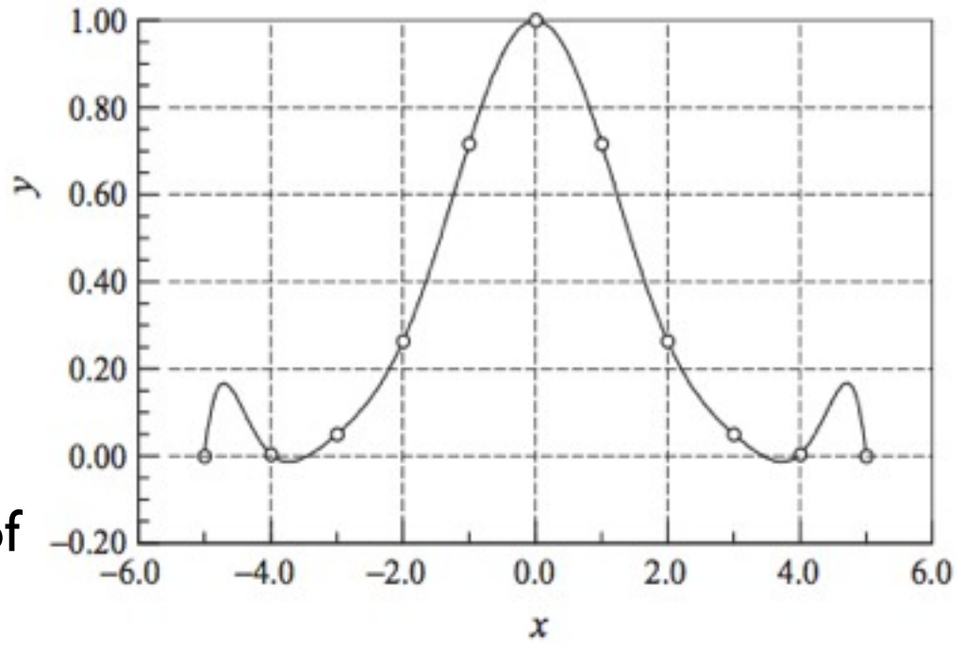
```
function a = newtonCoeff(xData,yData)
% Returns coefficients of Newton's polynomial.
% USAGE: a = newtonCoeff(xData,yData)
% xData = x-coordinates of data points.
% yData = y-coordinates of data points.

n = length(xData);
a = yData;
for k = 2:n
    a(k:n) = (a(k:n) - a(k-1))./(xData(k:n) - xData(k-1));
end
```

for n = 5

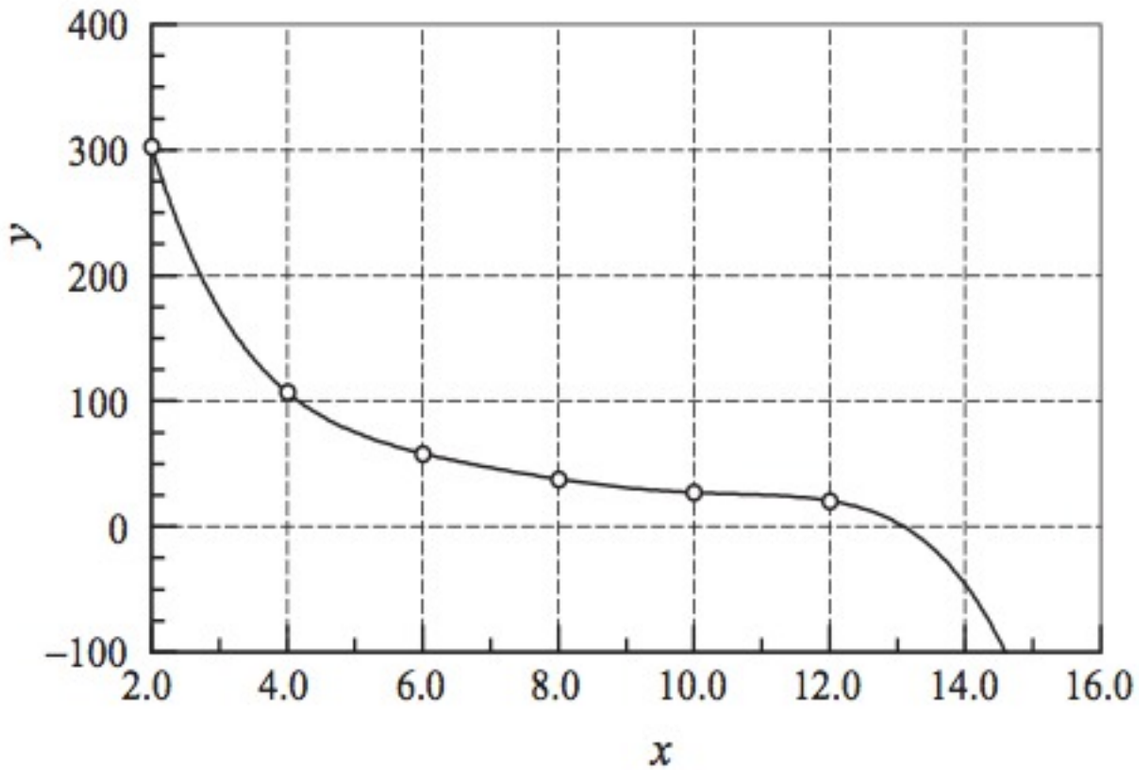
x_1	y_1				
x_2	y_2	∇y_2			
x_3	y_3	∇y_3	$\nabla^2 y_3$		
x_4	y_4	∇y_4	$\nabla^2 y_4$	$\nabla^3 y_4$	
x_5	y_5	∇y_5	$\nabla^2 y_5$	$\nabla^3 y_5$	$\nabla^4 y_5$

Limitations of Interpolation with polynomials

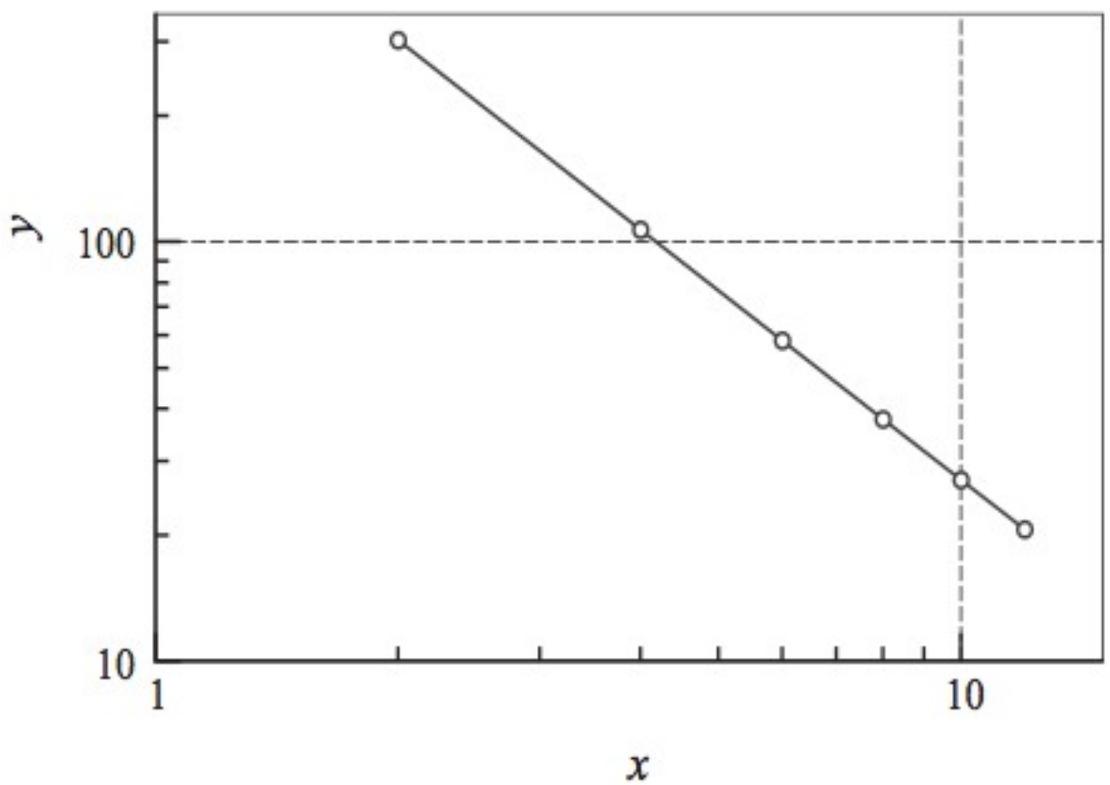
1. Works well when the total number of points used is few.
2. If many close points: linear interpolation is good.
3. Interpolation between 3 to 6 points is ok.
4. Too many points are neither useful (possibly dangerous)
5. Polynomial extrapolation outside the data points is dangerous. Better to use low order nearest neighbor interpolation (linear or quadratic). Can work with plot of log(x) vs log(y), which is generally a straight line.



dangers of using too many points (oscillatory behavior)
Runge Phenomenon



absurd extrapolation beyond 12.0



linear extrapolation on log-log plot

Given the data points

x	0	2	3
y	7	11	28

use Lagrange's method to determine y at $x = 1$.

Solution

$$\ell_1 = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = \frac{(1-2)(1-3)}{(0-2)(0-3)} = \frac{1}{3}$$

$$\ell_2 = \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} = \frac{(1-0)(1-3)}{(2-0)(2-3)} = 1$$

$$\ell_3 = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} = \frac{(1-0)(1-2)}{(3-0)(3-2)} = -\frac{1}{3}$$

$$y = y_1\ell_1 + y_2\ell_2 + y_3\ell_3 = \frac{7}{3} + 11 - \frac{28}{3} = 4$$

The data points in the table lie on the plot of $f(x) = 4.8\cos\pi x/20$. Interpolate this data by Newton's method at $x = 0, 0.5, 1.0, \ldots, 8.0$ and compare the results with the "exact" values given by $y = f(x)$.

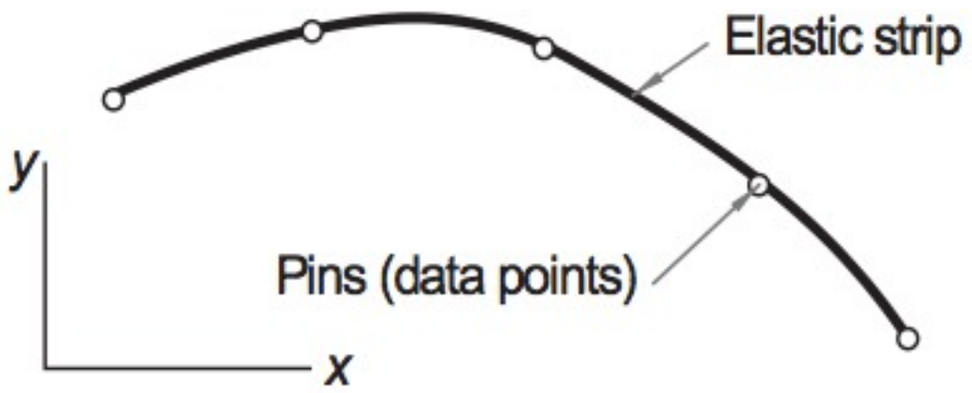
x	0.15	2.30	3.15	4.85	6.25	7.95
y	4.79867	4.49013	4.2243	3.47313	2.66674	1.51909

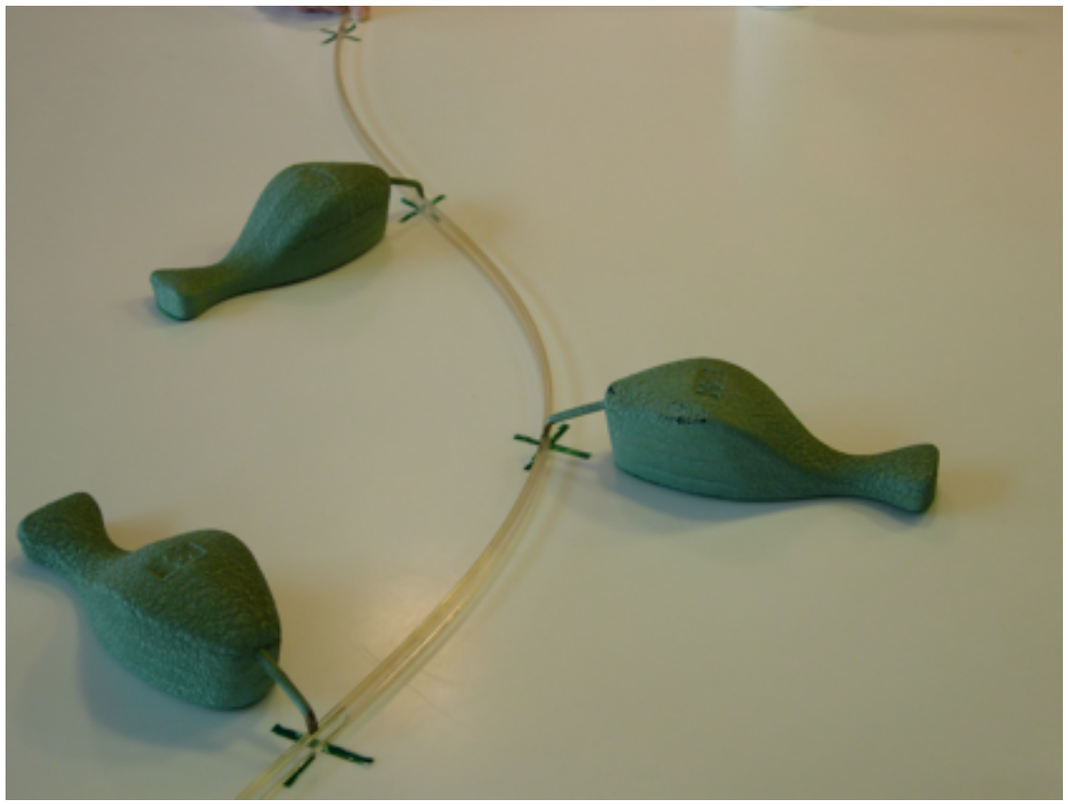
```
% Example 3.4 (Newton's interpolation)
xData = [0.15; 2.3; 3.15; 4.85; 6.25; 7.95];
yData = [4.79867; 4.49013; 4.22430; 3.47313;...
         2.66674; 1.51909];
a = newtonCoeff(xData,yData);
'       x       yInterp    yExact'
for x = 0: 0.5: 8
    y = newtonPoly(a,xData,x);
    yExact = 4.8*cos(pi*x/20);
    fprintf('%10.5f',x,y,yExact)
    fprintf('\n')
end
```

The results are:

```
ans =
      x       yInterp    yExact
   0.00000    4.80003    4.80000
   0.50000    4.78518    4.78520
   1.00000    4.74088    4.74090
   1.50000    4.66736    4.66738
   2.00000    4.56507    4.56507
   2.50000    4.43462    4.43462
   3.00000    4.27683    4.27683
   3.50000    4.09267    4.09267
   4.00000    3.88327    3.88328
   4.50000    3.64994    3.64995
   5.00000    3.39411    3.39411
   5.50000    3.11735    3.11735
   6.00000    2.82137    2.82137
   6.50000    2.50799    2.50799
   7.00000    2.17915    2.17915
   7.50000    1.83687    1.83688
   8.00000    1.48329    1.48328
```

Spline Interpolation

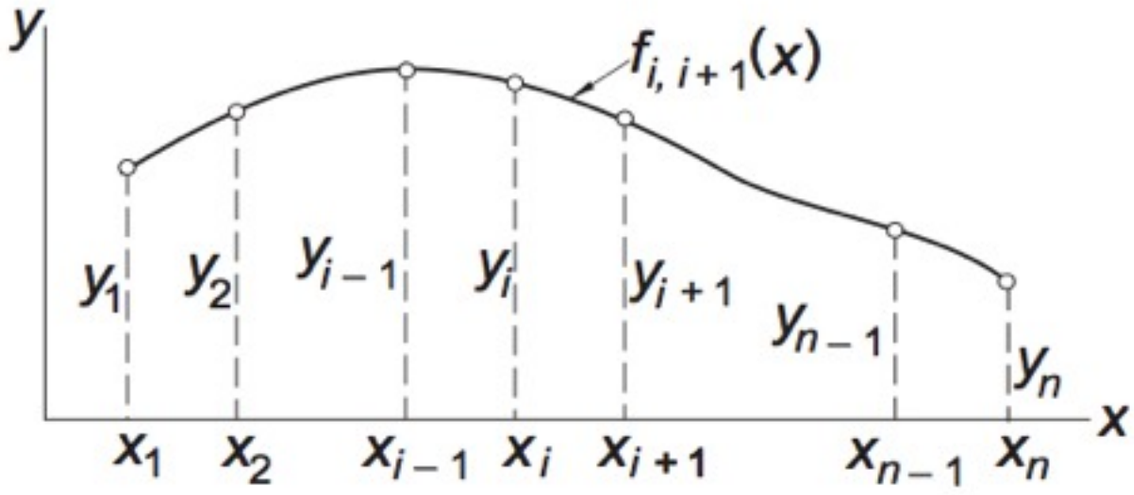




$$d^4y/dx^4 = q/(EI)$$

Since q = 0, the solution is cubic (at most)
Slope and bending moment and thus curvature is continuous.
At the ends curvature is zero.
This is called **natural cubic spline**
Data points are called *knots*.

Mathematical Description



put together from n-1 cubics

$$f_{1,2}(x),\ f_{2,3}(x),\ \ldots,\ f_{n-1,n}(x)$$

continuity of second derivative

$$f''_{i-1,i}(x_i) = f''_{i,i+1}(x_i) = k_i$$

$$k_1 = k_n = 0$$

The second derivatives are linear

$$f''_{i,i+1}(x) = k_i \ell_i(x) + k_{i+1}\ell_{i+1}(x)$$

$$\ell_i(x) = \frac{x - x_{i+1}}{x_i - x_{i+1}} \qquad \ell_{i+1}(x) = \frac{x - x_i}{x_{i+1} - x_i}$$

therefore

$$f''_{i,i+1}(x) = \frac{k_i(x - x_{i+1}) - k_{i+1}(x - x_i)}{x_i - x_{i+1}}$$

Integrating

$$f_{i,i+1}(x) = \frac{k_i(x - x_{i+1})^3 - k_{i+1}(x - x_i)^3}{6(x_i - x_{i+1})} + A(x - x_{i+1}) - B(x - x_i)$$

Using

$$f_{i,i+1}(x_i) = y_i, \qquad f_{i,i+1}(x_{i+1}) = y_{i+1}$$

$$A = \frac{y_i}{x_i - x_{i+1}} - \frac{k_i}{6}(x_i - x_{i+1})$$

$$B = \frac{y_{i+1}}{x_i - x_{i+1}} - \frac{k_{i+1}}{6}(x_i - x_{i+1})$$

Thus

$$f_{i,i+1}(x) = \frac{k_i}{6}\left[\frac{(x - x_{i+1})^3}{x_i - x_{i+1}} - (x - x_{i+1})(x_i - x_{i+1})\right]$$
$$- \frac{k_{i+1}}{6}\left[\frac{(x - x_i)^3}{x_i - x_{i+1}} - (x - x_i)(x_i - x_{i+1})\right]$$
$$+ \frac{y_i(x - x_{i+1}) - y_{i+1}(x - x_i)}{x_i - x_{i+1}}$$

to obtain *k's*

$$f'_{i-1,i}(x_i) = f'_{i,i+1}(x_i) \qquad i = 2, 3, \ldots, n-1.$$

results in simultaneous equations, which can be solved very easily because of the tri-diagonal nature of the matrix of *k's*

$$k_{i-1}(x_{i-1} - x_i) + 2k_i(x_{i-1} - x_{i+1}) + k_{i+1}(x_i - x_{i+1})$$
$$= 6\left(\frac{y_{i-1} - y_i}{x_{i-1} - x_i} - \frac{y_i - y_{i+1}}{x_i - x_{i+1}}\right), \qquad i = 2, 3, \ldots, n-1$$

Given data points: Matlab function to obtain *k's*

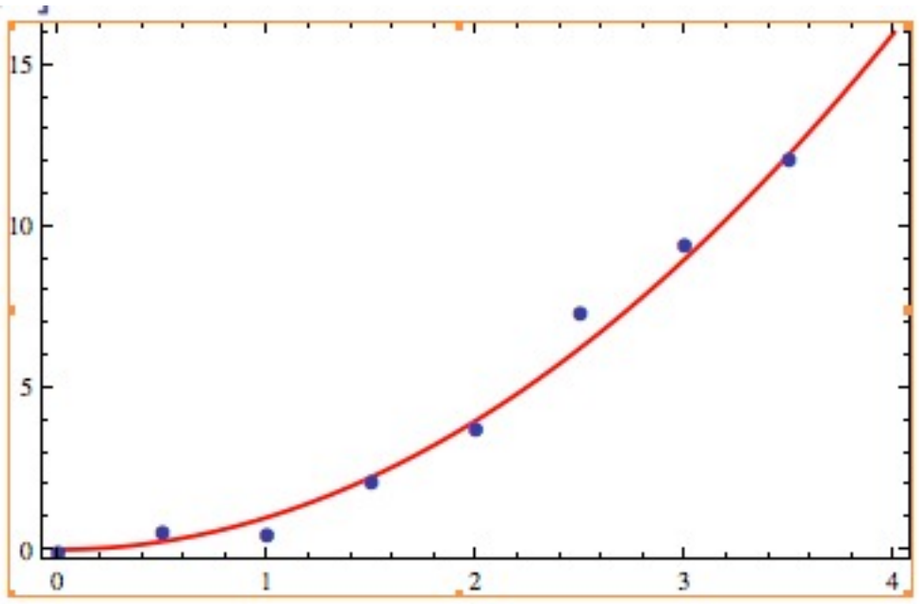
```
function k = splineCurv(xData,yData)
% Returns curvatures of a cubic spline at the
% USAGE: k = splineCurv(xData,yData)
% xData = x-coordinates of data points.
% yData = y-coordinates of data points.

n = length(xData);
c = zeros(n-1,1); d = ones(n,1);
e = zeros(n-1,1); k = zeros(n,1);
c(1:n-2) = xData(1:n-2) - xData(2:n-1);
d(2:n-1) = 2*(xData(1:n-2) - xData(3:n));
e(2:n-1) = xData(2:n-1) - xData(3:n);
k(2:n-1) = 6*(yData(1:n-2) - yData(2:n-1))...
           ./(xData(1:n-2) - xData(2:n-1))...
         - 6*(yData(2:n-1) - yData(3:n))...
           ./(xData(2:n-1) - xData(3:n));
[c,d,e] = LUdec3(c,d,e);
k = LUsol3(c,d,e,k);
```

Finally: Obtain the values of the function at any *x*

```
function y = splineEval(xData,yData,k,x)
% Returns value of cubic spline interpolant at x.
% USAGE: y = splineEval(xData,yData,k,x)
% xData = x-coordinates of data points.
% yData = y-coordinates of data points.
% k     = curvatures of spline at the knots;
%         returned by the function splineCurv.

i = findSeg(xData,x);
h = xData(i) - xData(i+1);
y = ((x - xData(i+1))^3/h - (x - xData(i+1))*h)*k(i)/6.0...
  - ((x - xData(i))^3/h - (x - xData(i))*h)*k(i+1)/6.0...
  + yData(i)*(x - xData(i+1))/h...
  - yData(i+1)*(x - xData(i))/h;
```

function to obtain in which *i,i+1* does *x* fall

```
function i = findSeg(xData,x)
% Returns index of segment containing x.
iLeft = 1; iRight = length(xData);
while 1
    if(iRight - iLeft) <= 1
        i = iLeft; return
    end
    i = fix((iLeft + iRight)/2);
    if x < xData(i)
        iRight = i;
    else
        iLeft = i;
    end
end
```



$$f(x) = f(x; a_1, a_2, \ldots, a_m)$$

Least Square Fit

n: no of data points

m: number of fitting parameters (n > m)

Smooth curve that fits data points on average

The form of the smooth curve is generally known or at least assumed.

Best fit is the one which minimizes the residual

$$r_i = y_i - f(x_i)$$

$$S(a_1, a_2, \ldots, a_m) = \sum_{i=1}^{n} [y_i - f(x_i)]^2$$

$$\frac{\partial S}{\partial a_k} = 0, \quad k = 1, 2, \ldots, m$$

The spread of the curve is denoted by

$$\sigma = \sqrt{\frac{S}{n-m}}$$

Fitting a Straight Line

Fitting a straight line

$$f(x) = a + bx \tag{3.16}$$

to data is also known as *linear regression*. In this case the function to be minimized is

$$S(a, b) = \sum_{i=1}^{n} (y_i - a - bx_i)^2$$

Equations (3.14) now become

$$\frac{\partial S}{\partial a} = \sum_{i=1}^{n} -2(y_i - a - bx_i) = 2\left(-\sum_{i=1}^{n} y_i + na + b\sum_{i=1}^{n} x_i\right) = 0$$

$$\frac{\partial S}{\partial b} = \sum_{i=1}^{n} -2(y_i - a - bx_i)x_i = 2\left(-\sum_{i=1}^{n} x_i y_i + a\sum_{i=1}^{n} x_i + b\sum_{i=1}^{n} x_i^2\right) = 0$$

Dividing both equations by $2n$ and rearranging terms, we get

$$a + b\bar{x} = \bar{y} \qquad a\bar{x} + \left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right) b = \frac{1}{n}\sum_{i=1}^{n} x_i y_i$$

where

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

are the mean values of the x and y data. The solution for the parameters is

$$b = \frac{\sum y_i(x_i - \bar{x})}{\sum x_i(x_i - \bar{x})} \qquad a = \bar{y} - \bar{x}b \qquad a = \frac{\bar{y}\sum x_i^2 - \bar{x}\sum x_i y_i}{\sum x_i^2 - n\bar{x}^2} \qquad b = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sum x_i^2 - n\bar{x}^2}$$

Fitting Linear Forms

Consider the least-squares fit of the *linear form*

$$f(x) = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_m f_m(x) = \sum_{j=1}^{m} a_j f_j(x)$$

$$S = \sum_{i=1}^{n} \left[y_i - \sum_{j=1}^{m} a_j f_j(x_i) \right]^2$$

$$\frac{\partial S}{\partial a_k} = -2 \left\{ \sum_{i=1}^{n} \left[y_i - \sum_{j=1}^{m} a_j f_j(x_i) \right] f_k(x_i) \right\} = 0, \quad k = 1, 2, \ldots, m$$

Dropping the constant (-2) and interchanging the order of summation, we get

$$\sum_{j=1}^{m} \left[\sum_{i=1}^{n} f_j(x_i) f_k(x_i) \right] a_j = \sum_{i=1}^{n} f_k(x_i) y_i, \quad k = 1, 2, \ldots, m$$

In matrix notation these equations are

$$\mathbf{Aa} = \mathbf{b} \qquad (3$$

$$A_{kj} = \sum_{i=1}^{n} f_j(x_i) f_k(x_i) \qquad b_k = \sum_{i=1}^{n} f_k(x_i) y_i$$

: $f(x) = \sum_{j=1}^{m} a_j x^{j-1}$.

$$A_{kj} = \sum_{i=1}^{n} x_i^{j+k-2} \qquad b_k = \sum_{i=1}^{n} x_i^{k-1} y_i$$

or

$$A_{kj} = \sum_{i=1}^{n} x_i^{j+k-2} \qquad b_k = \sum_{i=1}^{n} x_i^{k-1} y_i$$

$$\mathbf{A} = \begin{bmatrix} n & \sum x_i & \sum x_i^2 & \cdots & \sum x_i^m \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \cdots & \sum x_i^{m+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum x_i^{m-1} & \sum x_i^m & \sum x_i^{m+1} & \cdots & \sum x_i^{2m-2} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \vdots \\ \sum x_i^{m-1} y_i \end{bmatrix}$$

`polynFit`: Function to obtain coefficients of polynomial

```
function coeff = polynFit(xData,yData,m)
% Returns the coefficients of the polynomial

% a(1)*x^(m-1) + a(2)*x^(m-2) + ... + a(m)
% that fits the data points in the least squares sense.
% USAGE: coeff = polynFit(xData,yData,m)
% xData = x-coordinates of data points.
% yData = y-coordinates of data points.

A = zeros(m); b = zeros(m,1); s = zeros(2*m-1,1);
for i = 1:length(xData)
    temp = yData(i);
    for j = 1:m
        b(j) = b(j) + temp;
        temp = temp*xData(i);
    end
    temp = 1;
    for j = 1:2*m-1
        s(j) = s(j) + temp;
        temp = temp*xData(i);
    end
end
for i = 1:m
    for j = 1:m
        A(i,j) = s(i+j-1);
    end
end
% Rearrange coefficients so that coefficient
%  of x^(m-1) is first
coeff = flipdim(gaussPiv(A,b),1);
```

`stdDev`: To obtain the standard deviation

```
function sigma = stdDev(coeff,xData,yData)
% Returns the standard deviation between data
% points and the polynomial
% a(1)*x^(m-1) + a(2)*x^(m-2) + ... + a(m)
% USAGE: sigma = stdDev(coeff,xData,yData)
% coeff = coefficients of the polynomial.
% xData = x-coordinates of data points.

% yData = y-coordinates of data points.

m = length(coeff); n = length(xData);
sigma = 0;
for i =1:n
    y = polyEval(coeff,xData(i));
    sigma = sigma + (yData(i) - y)^2;
end
sigma =sqrt(sigma/(n - m));

function y = polyEval(coeff,x)
% Returns the value of the polynomial at x.
m = length(coeff);
y = coeff(1);
for j = 1:m-1
    y = y*x + coeff(j+1);
end
```

$$S(a_1, a_2, \ldots, a_m) = \sum^{n} W_i^2 \left[y_i - f(x_i)\right]^2$$

Weighted Linear Regression

If the fitting function is the straight line $f(x) = a + bx$, Eq. (3.24) becomes

$$S(a, b) = \sum_{i=1}^{n} W_i^2 (y_i - a - bx_i)^2$$

The conditions for minimizing S are

$$\frac{\partial S}{\partial a} = -2 \sum_{i=1}^{n} W_i^2 (y_i - a - bx_i) = 0$$

$$\frac{\partial S}{\partial b} = -2 \sum_{i=1}^{n} W_i^2 (y_i - a - bx_i) x_i = 0$$

$$a \sum_{i=1}^{n} W_i^2 + b \sum_{i=1}^{n} W_i^2 x_i = \sum_{i=1}^{n} W_i^2 y_i$$

$$a \sum_{i=1}^{n} W_i^2 x_i + b \sum_{i=1}^{n} W_i^2 x_i^2 = \sum_{i=1}^{n} W_i^2 x_i y_i$$

$$\hat{x} = \frac{\sum W_i^2 x_i}{\sum W_i^2} \qquad \hat{y} = \frac{\sum W_i^2 y_i}{\sum W_i^2}$$

$$a = \hat{y} - b\hat{x}$$

$$b = \frac{\sum_{i=1}^{n} W_i^2 y_i (x_i - \hat{x})}{\sum_{i=1}^{n} W_i^2 x_i (x_i - \hat{x})}$$

Fitting Exponential Functions

A special application of weighted linear regression arises in fitting exponential functions to data. Consider as an example the fitting function

$$f(x) = ae^{bx}$$

$$F(x) = \ln f(x) = \ln a + bx$$

to the data points $(x_i, \ln y_i)$, $i = 1, 2, \ldots, n$. This simplification comes at a price: least-squares fit to the logarithm of the data is not the same as least-squares fit to the original data. The residuals of the logarithmic fit are

$$R_i = \ln y_i - F(x_i) = \ln y_i - \ln a - bx_i$$

whereas the residuals used in fitting the original data are

$$r_i = y_i - f(x_i) = y_i - ae^{bx_i}$$

$$\therefore \ln(r_i - y_i) = \ln(ae^{bx_i}) = \ln a + bx_i$$

$$R_i = \ln y_i - \ln(r_i - y_i) = \ln\left(1 - \frac{r_i}{y_i}\right)$$

If the residuals r_i are sufficiently small ($r_i \ll y_i$), we can use the approximation $\ln(1 - r_i/y_i) \approx r_i/y_i$, so that

$$R_i \approx r_i/y_i$$

Fit a straight line to the data shown and compute the standard deviation.

x	0.0	1.0	2.0	2.5	3.0
y	2.9	3.7	4.1	4.4	5.0

Solution The averages of the data are

$$\bar{x} = \frac{1}{5}\sum x_i = \frac{0.0+1.0+2.0+2.5+3.0}{5} = 1.7$$

$$\bar{y} = \frac{1}{5}\sum y_i = \frac{2.9+3.7+4.1+4.4+5.0}{5} = 4.02$$

The intercept a and slope b of the interpolant can now be determined from

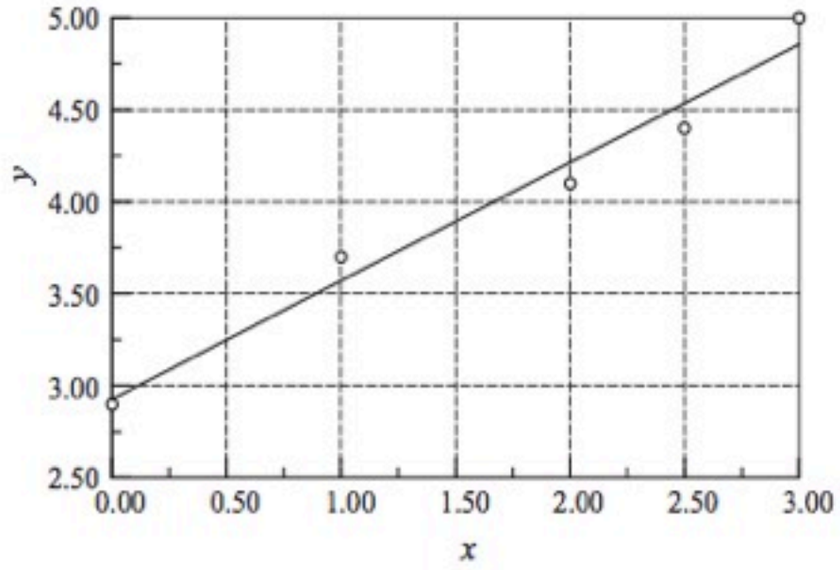
$$b = \frac{\sum y_i(x_i - \bar{x})}{\sum x_i(x_i - \bar{x})}$$

$$= \frac{2.9(-1.7)+3.7(-0.7)+4.1(0.3)+4.4(0.8)+5.0(1.3)}{0.0(-1.7)+1.0(-0.7)+2.0(0.3)+2.5(0.8)+3.0(1.3)}$$

$$= \frac{3.73}{5.8} = 0.6431$$

$$a = \bar{y} - \bar{x}b = 4.02 - 1.7(0.6431) = 2.927$$

Therefore, the regression line is $f(x) = 2.927 + 0.6431x$, which is shown in the figure together with the data points.



We start the evaluation of the standard deviation by computing the residuals:

y	2.900	3.700	4.100	4.400	5.000
$f(x)$	2.927	3.570	4.213	4.535	4.856
$y - f(x)$	−0.027	0.130	−0.113	−0.135	0.144

The sum of the squares of the residuals is

$$S = \sum [y_i - f(x_i)]^2$$

$$= (-0.027)^2 + (0.130)^2 + (-0.113)^2 + (-0.135)^2 + (0.144)^2 = 0.06936$$

so that the standard deviation in Eq. (3.15) becomes

$$\sigma = \sqrt{\frac{S}{5-2}} = \sqrt{\frac{0.06936}{3}} = 0.1520$$

Determine the parameters a and b so that $f(x) = ae^{bx}$ fits the following data in the least-squares sense.

x	1.2	2.8	4.3	5.4	6.8	7.9
y	7.5	16.1	38.9	67.0	146.6	266.2

Use two different methods: (1) fit $\ln y_i$ and (2) fit $\ln y_i$ with weights $W_i = y_i$. Compute the standard deviation in each case.

Solution of Part (1) The problem is to fit the function $\ln(ae^{bx}) = \ln a + bx$ to the data

x	1.2	2.8	4.3	5.4	6.8	7.9
$z = \ln y$	2.015	2.779	3.661	4.205	4.988	5.584

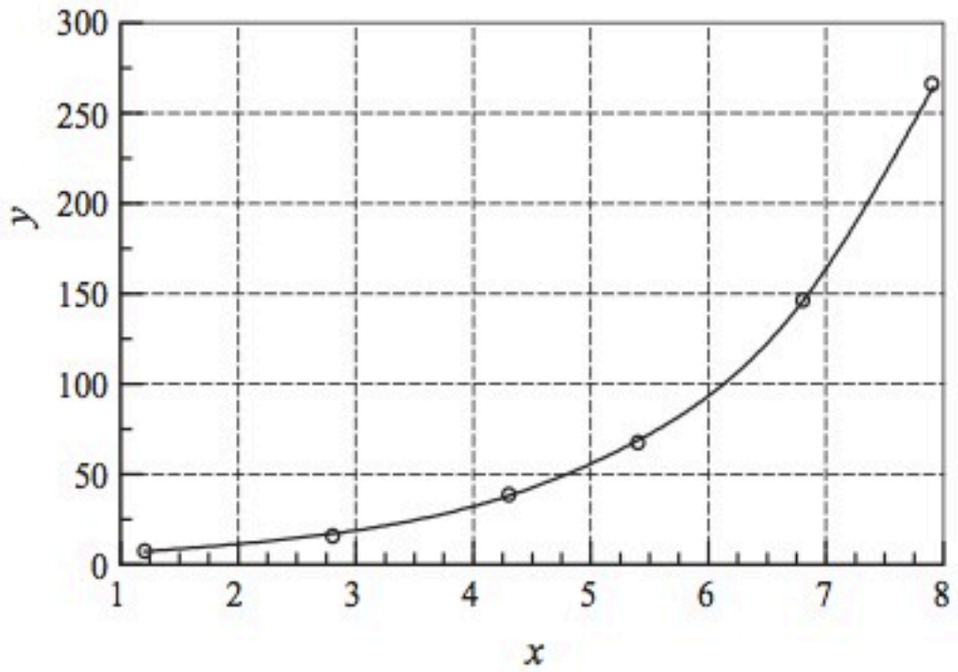
We are now dealing with linear regression, where the parameters to be found are $A = \ln a$ and b. Following the steps in Example 3.10, we get (skipping some of the

arithmetic details)

$$\bar{x} = \frac{1}{6}\sum x_i = 4.733 \qquad \bar{z} = \frac{1}{6}\sum z_i = 3.872$$

$$b = \frac{\sum z_i(x_i - \bar{x})}{\sum x_i(x_i - \bar{x})} = \frac{16.716}{31.153} = 0.5366 \qquad A = \bar{z} - \bar{x}b = 1.3323$$

Therefore, $a = e^A = 3.790$ and the fitting function becomes $f(x) = 3.790e^{0.5366}$. The plots of $f(x)$ and the data points are shown in the figure.



Here is the computation of standard deviation:

y	7.50	16.10	38.90	67.00	146.60	266.20
$f(x)$	7.21	17.02	38.07	68.69	145.60	262.72
$y - f(x)$	0.29	−0.92	0.83	−1.69	1.00	3.48

$$S = \sum [y_i - f(x_i)]^2 = 17.59$$

$$\sigma = \sqrt{\frac{S}{6-2}} = 2.10$$

As pointed out before, this is an approximate solution of the stated problem, since we did not fit y_i, but $\ln y_i$. Judging by the plot, the fit seems to be good.

Solution of Part (2) We again fit $\ln(ae^{bx}) = \ln a + bx$ to $z = \ln y$, but this time the weights $W_i = y_i$ are used. From Eqs. (3.27) the weighted averages of the data are (recall that we fit $z = \ln y$)

$$\hat{x} = \frac{\sum y_i^2 x_i}{\sum y_i^2} = \frac{737.5 \times 10^3}{98.67 \times 10^3} = 7.474$$

$$\hat{z} = \frac{\sum y_i^2 z_i}{\sum y_i^2} = \frac{528.2 \times 10^3}{98.67 \times 10^3} = 5.353$$

$$b = \frac{\sum y_i^2 z_i (x_i - \hat{x})}{\sum y_i^2 x_i (x_i - \hat{x})} = \frac{35.39 \times 10^3}{65.05 \times 10^3} = 0.5440$$

$$\ln a = \hat{z} - b\hat{x} = 5.353 - 0.5440(7.474) = 1.287$$

Therefore,

$$a = e^{\ln a} = e^{1.287} = 3.622$$

so that the fitting function is $f(x) = 3.622e^{0.5440x}$. As expected, this result is somewhat different from that obtained in Part (1).

The computations of the residuals and standard deviation are as follows:

y	7.50	16.10	38.90	67.00	146.60	266.20
$f(x)$	6.96	16.61	37.56	68.33	146.33	266.20
$y - f(x)$	0.54	−0.51	1.34	−1.33	0.267	0.00

$$S = \sum [y_i - f(x_i)]^2 = 4.186$$

$$\sigma = \sqrt{\frac{S}{6-2}} = 1.023$$

Observe that the residuals and standard deviation are smaller than that in Part (1), indicating a better fit, as expected.

It can be shown that fitting y_i directly (which involves the solution of a transcendental equation) results in $f(x) = 3.614e^{0.5442}$. The corresponding standard deviation is $\sigma = 1.022$, which is very close to the result in Part (2).

Write a program that fits a polynomial of arbitrary degree k to the data points shown below. Use the program to determine k that best fits these data in the least-squares sense.

x	−0.04	0.93	1.95	2.90	3.83	5.00
y	−8.66	−6.44	−4.36	−3.27	−0.88	0.87
x	5.98	7.05	8.21	9.08	10.09	
y	3.31	4.63	6.19	7.40	8.85	

Solution The following program prompts for m. Execution is terminated by pressing "return."

```
% Example 3.12 (Polynomial curve fitting)
xData = [-0.04,0.93,1.95,2.90,3.83,5.0,...
                 5.98,7.05,8.21,9.08,10.09]';
yData = [-8.66,-6.44,-4.36,-3.27,-0.88,0.87,...
                 3.31,4.63,6.19,7.4,8.85]';

format short e
while 1
    k = input('degree of polynomial = ');
    if isempty(k)         % Loop is terminated
        fprintf('Done')   % by pressing ''return''
        break
    end
    coeff = polynFit(xData,yData,k+1)
    sigma = stdDev(coeff,xData,yData)
    fprintf('\n')
end
```

```
Degree of polynomial = 1
coeff =
  1.7286e+000
 -7.9453e+000
sigma =
  5.1128e-001

degree of polynomial = 2
coeff =
 -4.1971e-002
  2.1512e+000
 -8.5701e+000
sigma =
  3.1099e-001

degree of polynomial = 3
coeff =
 -2.9852e-003
  2.8845e-003
  1.9810e+000
 -8.4660e+000
sigma =
  3.1948e-001

degree of polynomial =
Done
```