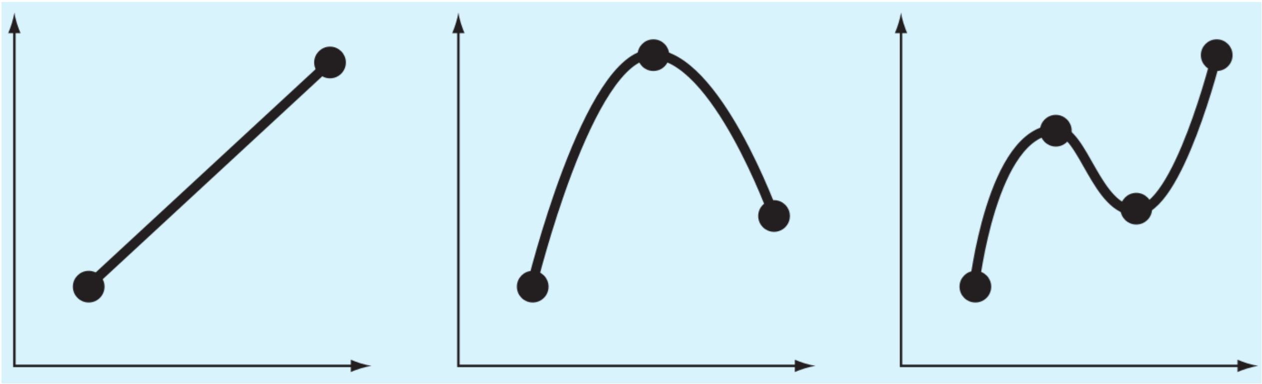
9.1 Polynomial Interpolation

- Previously we found a best fitting line to certain data.

- We now look at finding a curve that exactly passes through each of the data points that we have collected in order to estimate a function value in between the points.

- This is known as **interpolation**. The problem of finding coefficients for an interpolating polynomial is solved using linear algebra.



$$f(x) = p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x + p_n$$

- If we have $n + 1$ data points we can fit an nth degree polynomial to the data by setting up a system of linear equations (where the coefficients are the variables) then solving using standard techniques.

EXAMPLE 1 Find an interpolating polynomial of degree 2 that fits the following data then estimate the density at 350ºC.

$$f(x) = p_1x^2 + p_2x + p_3$$

2nd degree polynomial

T, °C	ρ, kg/m^3
300	0.616
400	0.525
500	0.457

System of equations

$$0.616 = p_1(300)^2 + p_2(300) + p_3$$
$$0.525 = p_1(400)^2 + p_2(400) + p_3$$
$$0.457 = p_1(500)^2 + p_2(500) + p_3$$

$$x_1 = 300 \quad f(x_1) = 0.616$$
$$x_2 = 400 \quad f(x_2) = 0.525$$
$$x_3 = 500 \quad f(x_3) = 0.457$$

Matrix form

$$\begin{bmatrix} 90{,}000 & 300 & 1 \\ 160{,}000 & 400 & 1 \\ 250{,}000 & 500 & 1 \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \end{Bmatrix} = \begin{Bmatrix} 0.616 \\ 0.525 \\ 0.457 \end{Bmatrix}$$

- Solving in Matlab:

```
>> format long
>> A = [90000 300 1; 160000 400 1; 250000 500 1];
>> b = [0.616 0.525 0.457]';
>> p = A\b
p =
    0.00000115000000
   -0.00171500000000
    1.02699999999999
```

$$\longrightarrow \quad f(x) = 0.00000115x^2 - 0.001715x + 1.027$$

$$\longrightarrow \quad f(350) = 0.00000115(350)^2 - 0.001715(350) + 1.027 = 0.567625$$

Weakness of the Method

- The previous method is very easy to implement however it does have a weakness.
- Our matrix has rows that are geometric series.

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \end{Bmatrix} = \begin{Bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{Bmatrix}$$

- These are known as Vandermonde Matrices and are renowned as being **ill-conditioned**.
- Ill-conditioned matrices basically mean that the system is **highly sensitive to errors in data**. This means that when we use the backslash operator to solve it we might get inaccurate results if the data has small errors.

EXAMPLE 2 Observe how the solution to the linear system changes by a large amount when the data in the matrix changes only by a small amount.

- Create an ill-conditioned matrix:

```
>> format long
>> A = [1 1-1e-8; 1 1+1e-8]
A =
         1          0.99999999
         1          1.00000001
```

- Let's pick a vector ***x*** = [1;3] to obtain ***b*** in the matrix equation ***Ax*** = ***b***.

```
>> x = [1;3]
x =
     1
     3
>> b = A*x
b =
        3.99999997
        4.00000003
```

- Now let's use the backslash operator to solve the linear system for ***x*** using the ***b*** we just calculated:

```
>> x2 = A \ b
x2 =
     0.999999994448885
     3.000000000555112
```

- So solving the system didn't quite bring us back to our original values of ***x*** but still pretty close.

- Now look what happens if the measured data, ***b***, has a small error of 1e-7 = 0.0000001:

```
>> x2 = A \ (b + [1e-7;-1e-7])
x2 =
   11.000000001165734
   -7.000000011657342
```

The solution for *x* has become totally corrupted from a small in change in *b*

- We can check the **condition number** of a matrix in Matlab to see how well/badly conditioned it is.

```
>> format shortg
>> cond(A)
ans =
        2e+08
```

- This tells us that we may lose up to 8 significant figures when we try to solve it.

- The larger the number, the more badly conditioned it is. In **Example 1** the condition of the matrix was also high from our interpolating polynomial system:

```
>> A = [90000 300 1;160000 400 1;250000 500 1];
>> cond(A)
ans =
   5.8932e+06
```

- The condition number is defined to be:

$$\boxed{\kappa(A) = \|A\|_2 \|A^{-1}\|_2}$$

where, $$\|A\|_2 = \max_{1 \le j \le n} \sigma_{max}$$

and the σ_i are the square roots of the Eigenvalues of $\boldsymbol{A^T A}$.

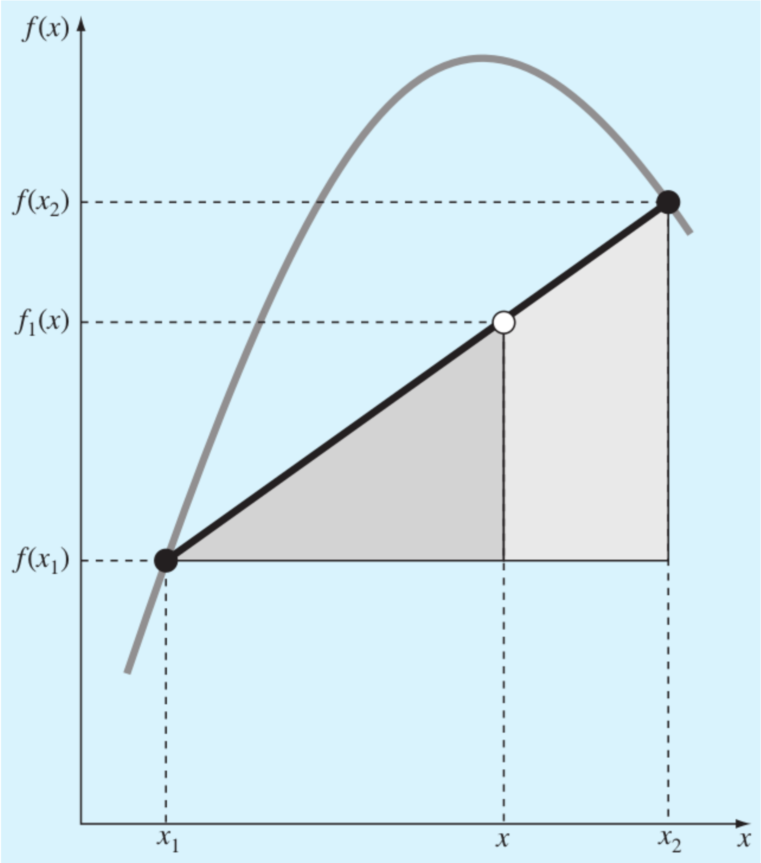
- The $||\boldsymbol{A}||_2$ represents the 2-norm of a matrix and norms are used in linear algebra as a metric that indicates a relative "size".

- The 2-norm used above is the matrix equivalent of the Euclidean norm (extension of Pythagoras' Theorem for vector lengths).

- When $\boldsymbol{\kappa(A)}$ is close to 1 the matrix is well-conditioned.

9.2 Newton Interpolating Polynomial

- When we want to interpolate many data points we get extremely large systems.
- Since the simple method from section **9.1** is highly sensitive to errors, these get magnified for larger systems and we must have other methods available to us.
- We begin by doing a linear interpolation (straight line) between 2 data points.



- Similar triangles in the previous figure gives:

$$\frac{f_1(x) - f(x_1)}{x - x_1} = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

Secant line joining 2 points →

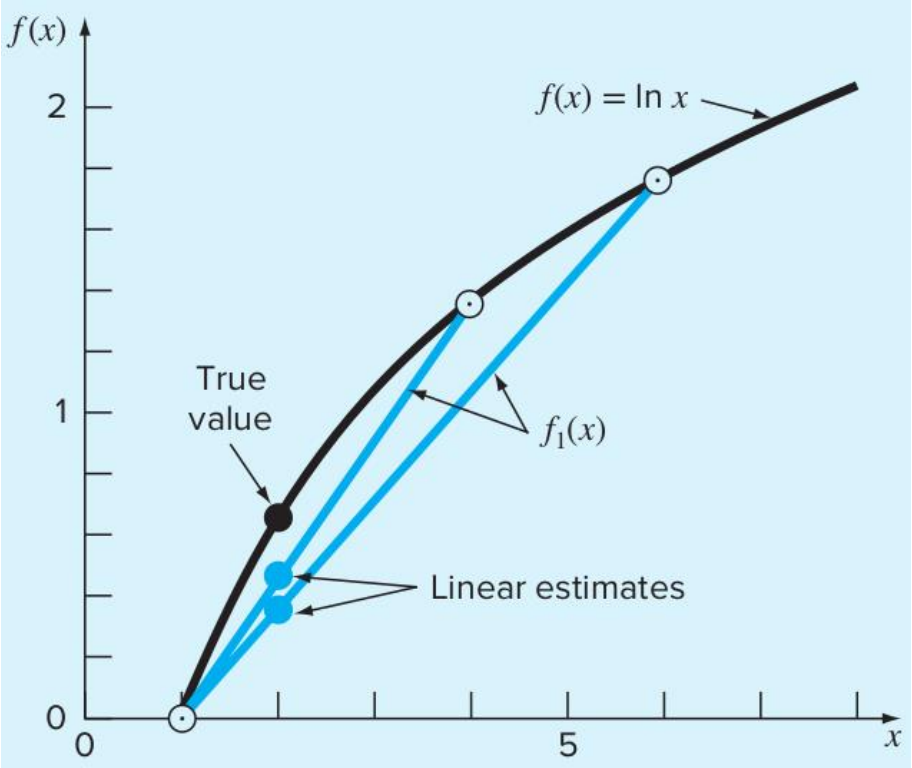
$$f_1(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1)$$

EXAMPLE 3

The graph of $f(x) = \ln(x)$ is shown on the right.

We can estimate
the value of $f(2)$ with a linear interpolation between 2 points.

The closer the points, the closer our estimation.



Quadratic Newton Interpolation

- To get a better approximation of nonlinear data we can use a quadratic of the following form:

Interpolating quadratic

$$f(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2)$$

- Evaluating at $x = x_1$ gives: $$b_1 = f(x_1)$$

- Substituting for b_1 then evaluating at $x = x_2$ gives:

$$b_2 = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

- Substituting for b_1 and b_2 then evaluating at $x = x_3$ gives:

$$b_3 = \frac{\dfrac{f(x_3) - f(x_2)}{x_3 - x_2} - \dfrac{f(x_2) - f(x_1)}{x_2 - x_1}}{x_3 - x_1}$$

EXAMPLE 4 Use a quadratic polynomial to interpolate the following data:

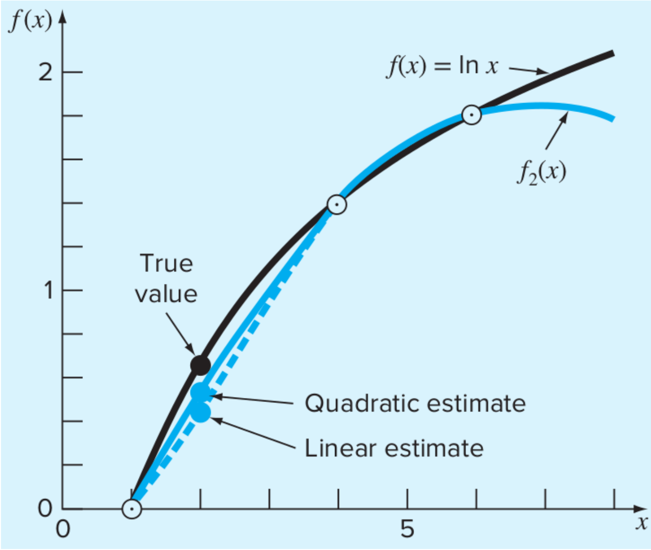
x	$f(x) = \ln(x)$
1	0
4	1.386294
6	1.791759

$$x_1 = 1 \qquad f(x_1) = 0$$
$$x_2 = 4 \qquad f(x_2) = 1.386294$$
$$x_3 = 6 \qquad f(x_3) = 1.791759$$

$$b_1 = 0 \qquad b_2 = \frac{1.386294 - 0}{4 - 1} = 0.4620981$$

$$b_3 = \frac{\dfrac{1.791759 - 1.386294}{6 - 4} - 0.4620981}{6 - 1} = -0.0518731$$

$$f(x) = 0 + 0.4620981(x - 1) - 0.0518731(x - 1)(x - 4)$$



(n-1)th Degree Newton Polynomial Interpolation

$$f(x) = b_1 + b_2(x - x_1) + \cdots + b_n(x - x_1)(x - x_2)\cdots(x - x_{n-1})$$

- Defining the **finite divided differences** as:

Notice the recursive nature which makes it ideal for programming into a loop

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

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

.

.

.

$$f[x_n, x_{n-1}, \ldots, x_2, x_1] = \frac{f[x_n, x_{n-1}, \ldots, x_2] - f[x_{n-1}, x_{n-2}, \ldots, x_1]}{x_n - x_1}$$

- We can extend the formulas from linear and quadratic interpolation to the (n-1)th degree as follows:

$$b_1 = f(x_1)$$

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

$$b_3 = f[x_3, x_2, x_1]$$

$$\vdots$$

$$b_n = f[x_n, x_{n-1}, \ldots, x_2, x_1]$$

$$\downarrow$$

$$f(x) = f(x_1) + (x - x_1) f[x_2, x_1] + (x - x_1)(x - x_2) f[x_3, x_2, x_1] + \cdots + (x - x_1)(x - x_2) \cdots (x - x_{n-1}) f[x_n, x_{n-1}, \ldots, x_2, x_1]$$

- Each coefficient is calculated from previous coefficients:

x_i	$f(x_i)$	First	Second	Third
x_1	$f(x_1)$	$f[x_2, x_1]$	$f[x_3, x_2, x_1]$	$f[x_4, x_3, x_2, x_1]$
x_2	$f(x_2)$	$f[x_3, x_2]$	$f[x_4, x_3, x_2]$	
x_3	$f(x_3)$	$f[x_4, x_3]$		
x_4	$f(x_4)$			

EXAMPLE 5 Use a cubic polynomial to interpolate the following data:

x	$f(x) = \ln(x)$
1	0
4	1.386294
6	1.791759
5	1.609438

Notice that the data doesn't have to be in ascending order for the method to work

$$f_3(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2) + b_4(x - x_1)(x - x_2)(x - x_3)$$

First Divided Differences

$$f[x_2, x_1] = \frac{1.386294 - 0}{4 - 1} = 0.4620981$$

$$f[x_3, x_2] = \frac{1.791759 - 1.386294}{6 - 4} = 0.2027326$$

$$f[x_4, x_3] = \frac{1.609438 - 1.791759}{5 - 6} = 0.1823216$$

Second Divided Differences

$$f[x_3, x_2, x_1] = \frac{0.2027326 - 0.4620981}{6 - 1} = -0.05187311$$

$$f[x_4, x_3, x_2] = \frac{0.1823216 - 0.2027326}{5 - 4} = -0.02041100$$

Third Divided Difference

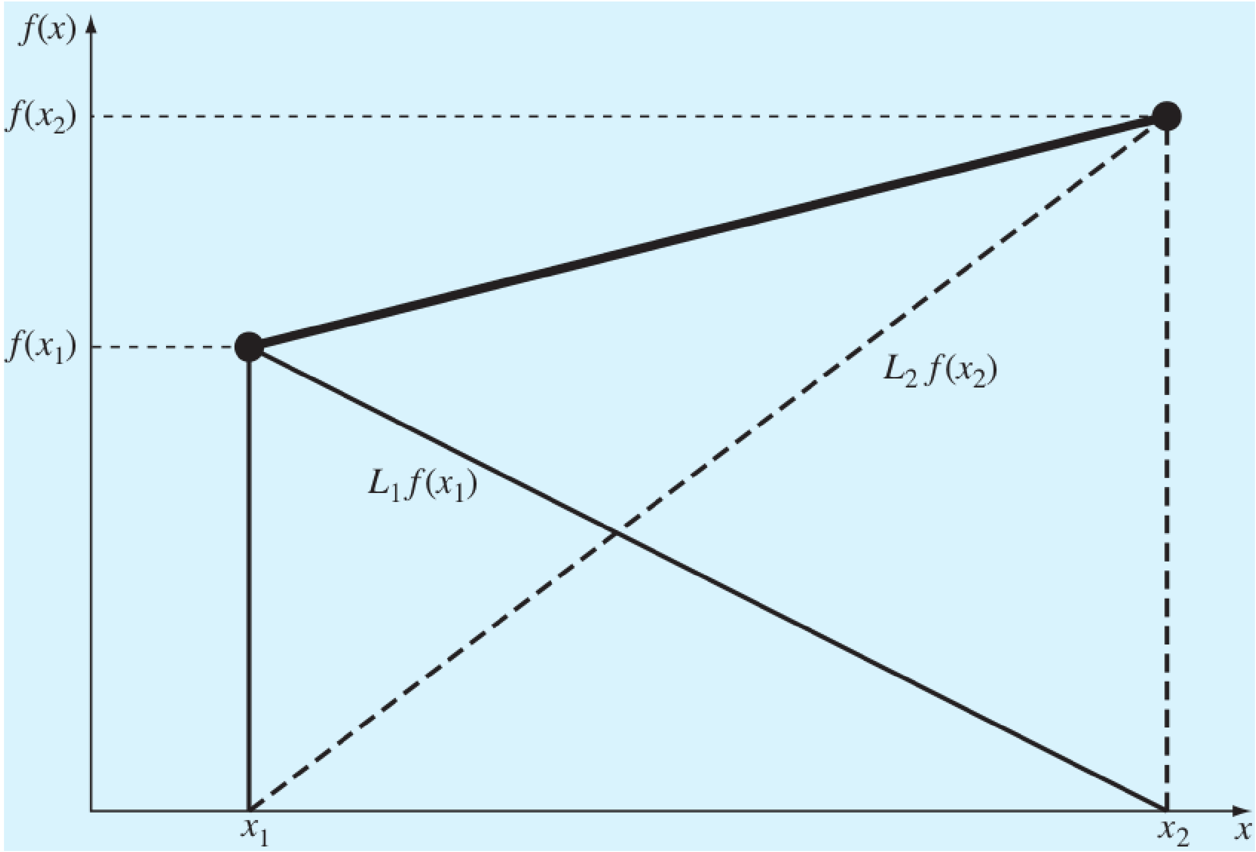
$$f[x_4, x_3, x_2, x_1] = \frac{-0.02041100 - (-0.05187311)}{5 - 1} = 0.007865529$$

x_i	$f(x_i)$	First	Second	Third
1	0	0.4620981	−0.05187311	0.007865529
4	1.386294	0.2027326	−0.02041100	
6	1.791759	0.1823216		
5	1.609438			

$$f(x) = 0 + 0.4620981(x-1) - 0.05187311(x-1)(x-4) + 0.007865529(x-1)(x-4)(x-6)$$

9.3 Lagrange Interpolating Polynomial

- Another way to look at linear interpolation is by considering the sum of the following straight lines:



- This leads to the following formula for the linear interpolating line

$$f(x) = L_1 f(x_1) + L_2 f(x_2)$$

where, $L_1 = \dfrac{x - x_2}{x_1 - x_2}$ $\qquad$ $L_2 = \dfrac{x - x_1}{x_2 - x_1}$

$$\longrightarrow \quad f(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2)$$

- This can be extended to a quadratic as:

$$f(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f(x_2)$$

$$+ \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f(x_3)$$

- The quadratic coefficients come from summing the 3 quadratic curves that pass through 1 point each and are equal to 0 at the other 2 points.

- The general *n*th degree Lagrange interpolating polynomial is then,

$$f(x) = \sum_{i=1}^{n+1} L_i(x) f(x_i)$$

where,

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{x - x_j}{x_i - x_j}$$

Product notation

EXAMPLE 6 Use a Lagrange interpolating polynomial to estimate the function value when $x = 15$.

$$x_1 = 0 \qquad f(x_1) = 3.85$$

$$x_2 = 20 \qquad f(x_2) = 0.800$$

$$x_3 = 40 \qquad f(x_3) = 0.212$$

Linear Lagrange

$$f_1(x) = \frac{15 - 20}{0 - 20} 3.85 + \frac{15 - 0}{20 - 0} 0.800 = 1.5625$$

Quadratic Lagrange

$$f_2(x) = \frac{(15 - 20)(15 - 40)}{(0 - 20)(0 - 40)} 3.85 + \frac{(15 - 0)(15 - 40)}{(20 - 0)(20 - 40)} 0.800$$

$$+ \frac{(15 - 0)(15 - 20)}{(40 - 0)(40 - 20)} 0.212 = 1.3316875$$

9.4 Inverse Interpolation

- If we find an interpolating polynomial for some data then inverse interpolation is the method of using it to find inverse values.

- This is essentially a combination of an interpolation problem with a root finding problem.

EXAMPLE 7 For the following data estimate which value of x would correspond with $f(x) = 0.3$.

x	1	2	3	4	5	6	7
$f(x)$	1	0.5	0.3333	0.25	0.2	0.1667	0.1429

- We start by finding an interpolating polynomial by any means (for example Lagrange) for some data points:

$$(2, 0.5), (3, 0.3333), \text{ and } (4, 0.25)$$

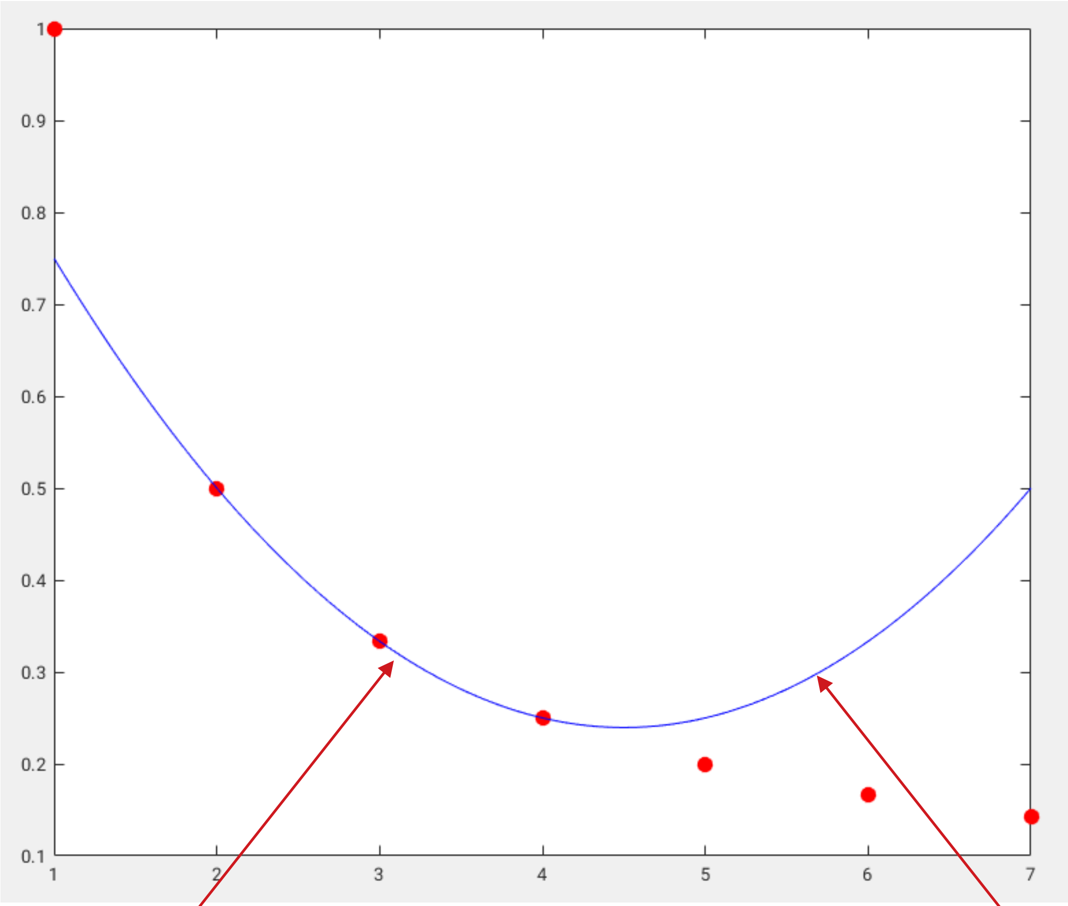
$$f_2(x) = 0.041667x^2 - 0.375x + 1.08333$$

- Notice we picked 3 data points where the endpoint function values bracket the value (0.3) of interest. We could have picked more data points for a higher order polynomial.

- Finally we solve the polynomial for the function value. Since ours is quadratic we can use the standard formula, however a root finding algorithm may be necessary.

$$x = \frac{0.375 \pm \sqrt{(-0.375)^2 - 4(0.041667)0.78333}}{2(0.041667)} = \begin{matrix} 5.704158 \\ \boxed{3.295842} \end{matrix}$$

- We can check which root to use by plotting:



We should use this root

This root is an artefact of the inaccurate interpolation process

9.5 Using Matlab Built-In Functions

- We can use the same functions in Matlab for interpolation as we did in regression as long as the number of data points is the same as the degree of the polynomial that we are fitting.

- In this case, regression becomes the same as interpolation.

EXAMPLE 8 Obtain the polynomial to fit the data in **Example 7** using Matlab built-in functions.

```
>> x = 1:7;
>> y = 1./x;
```

Full data set

```
>> xx = 2:4;
```

Data to interpolate

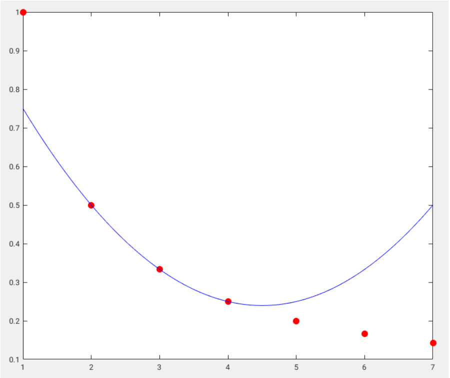
```
>> yy = 1./xx;
>> a=polyfit(xx,yy,2)
a =
      0.041667        -0.375        1.0833
```

Fit the data (this can be used for regression too)

```
>> xxx = linspace(1,7);
>> yyy = polyval(a,xxx);
```

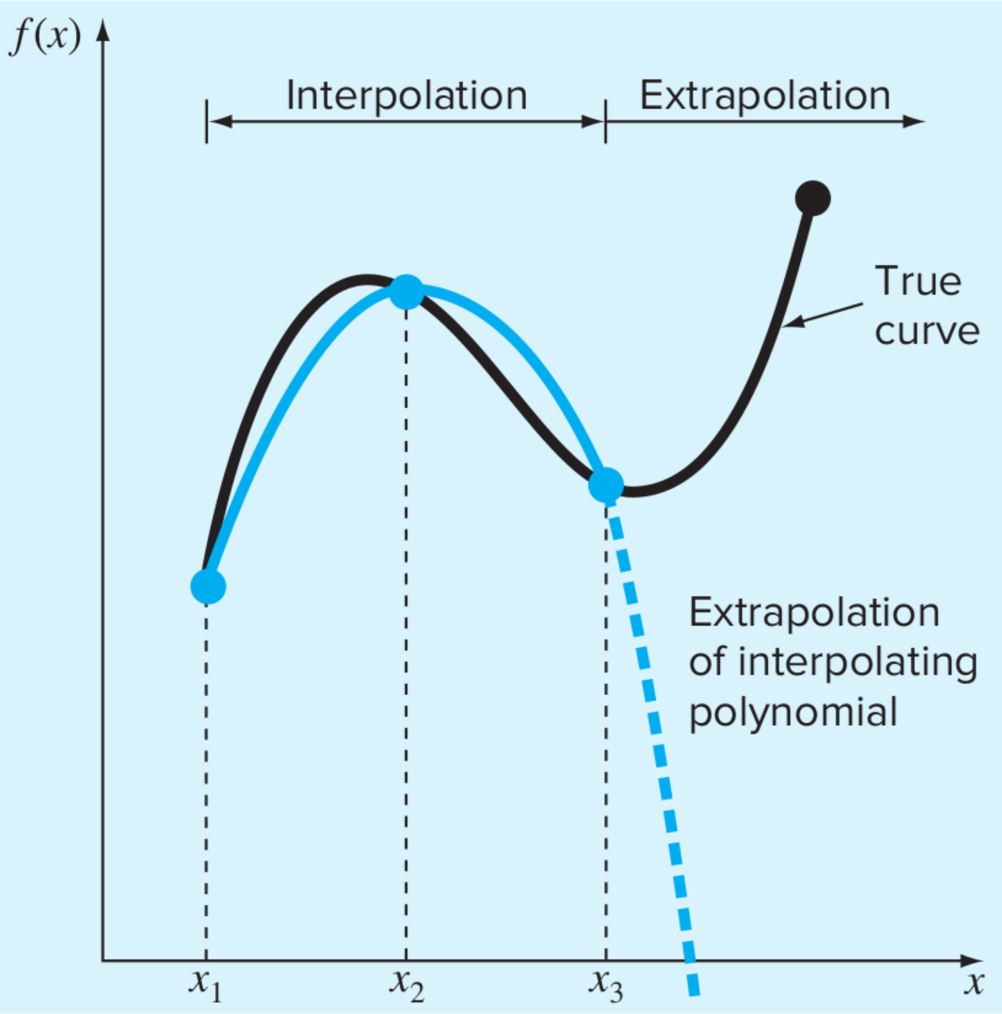
Plot interpolating function against data

```
>> plot(x,y,'or','markersize',8,'markerfacecolor','r')
>> hold on
>> plot(xxx,yyy,'b')
```



9.6 Extrapolation

- It is important to recognise that extending your interpolating polynomial beyond the data points is a generally bad a idea.
- The approximations are safer to use at data points within the interval of the original measurements



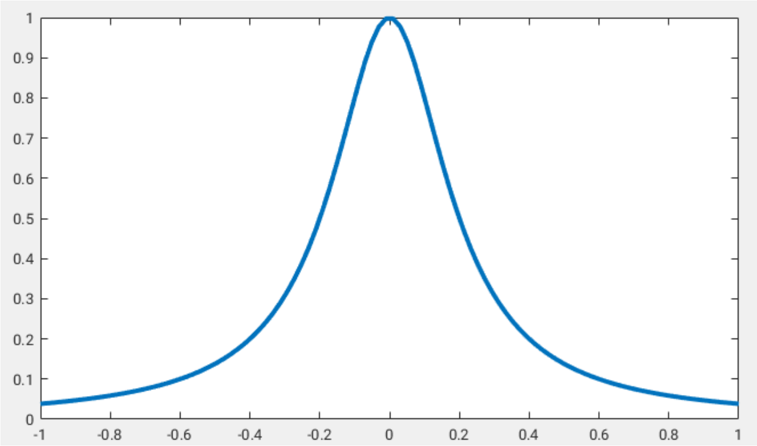
9.7 Oscillations

- It's also possible that some interpolating polynomials do not match the data well even though they pass through the data points.

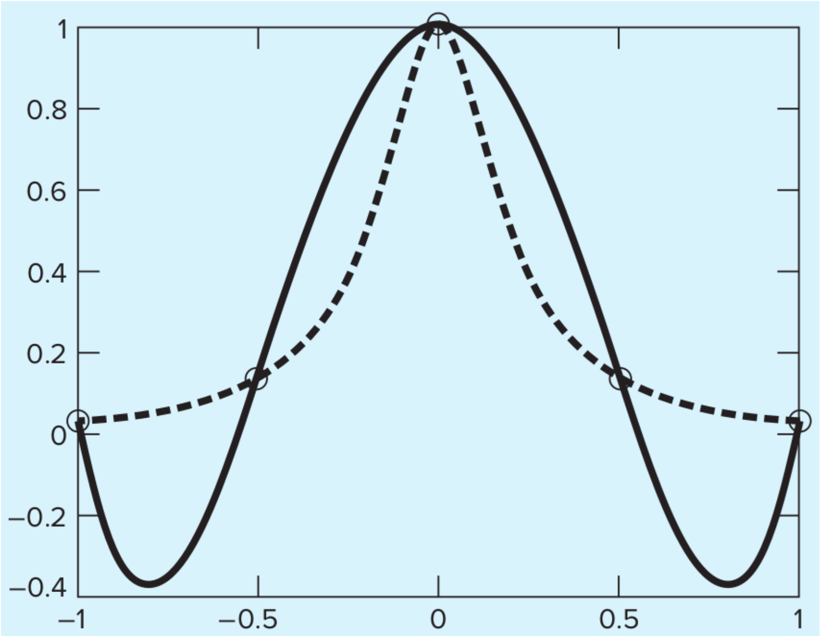
- One way that this can happen is by oscillations.

- To demonstrate we apply the interpolating methods to the following function:

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

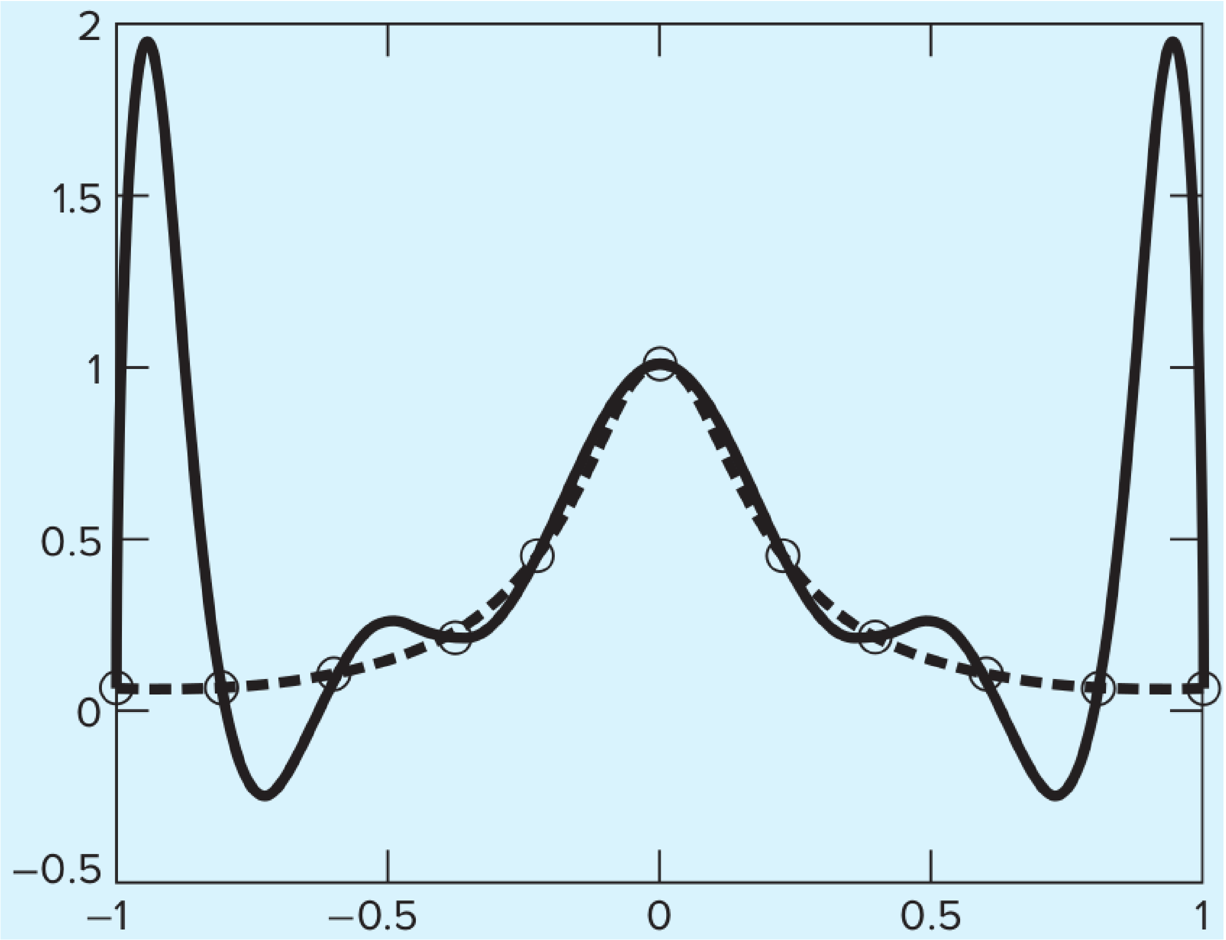


- Sampling data points along the interval [–1,1] we can interpolate. Sampling 5 data points gives us a 4th order polynomial:



- Notice that the interpolating polynomial (solid line) does not represent the true data at all even though it passes through the sampled data points.

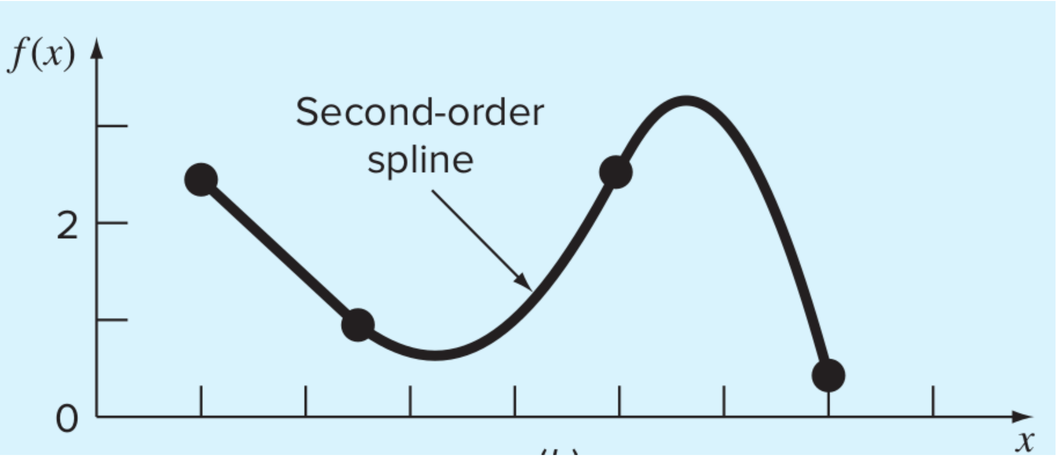
- Normally increasing the number of data points and taking a higher order polynomial gives more accurate results, however this is not the case for this function:



9.8 Splines

- The problems with higher order polynomials as we saw in the last section can be overcome using **splines**.

- It essentially uses cubic (or lower degree) interpolating polynomials over subsets of data and joins them at "**knots**".



Quadratic interpolating splines joined at the knots $2^{nd}/3^{rd}$ data points

- The method for deriving cubic splines is very similar to how we derived interpolating polynomials (choose the form of the equation and find the coefficients by substitution).

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

- However we must add in the knot conditions.
- Each s_i is one section of the interpolating curve.
- For n data points we will have $n - 1$ intervals and $4(n - 1)$ coefficients to solve for (4 for each interval).
- The conditions for the knots are that the first/second derivatives are equal for the adjoining curves.

- We define the interval widths of the data points to be:

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

- We also will use the following finite differences:

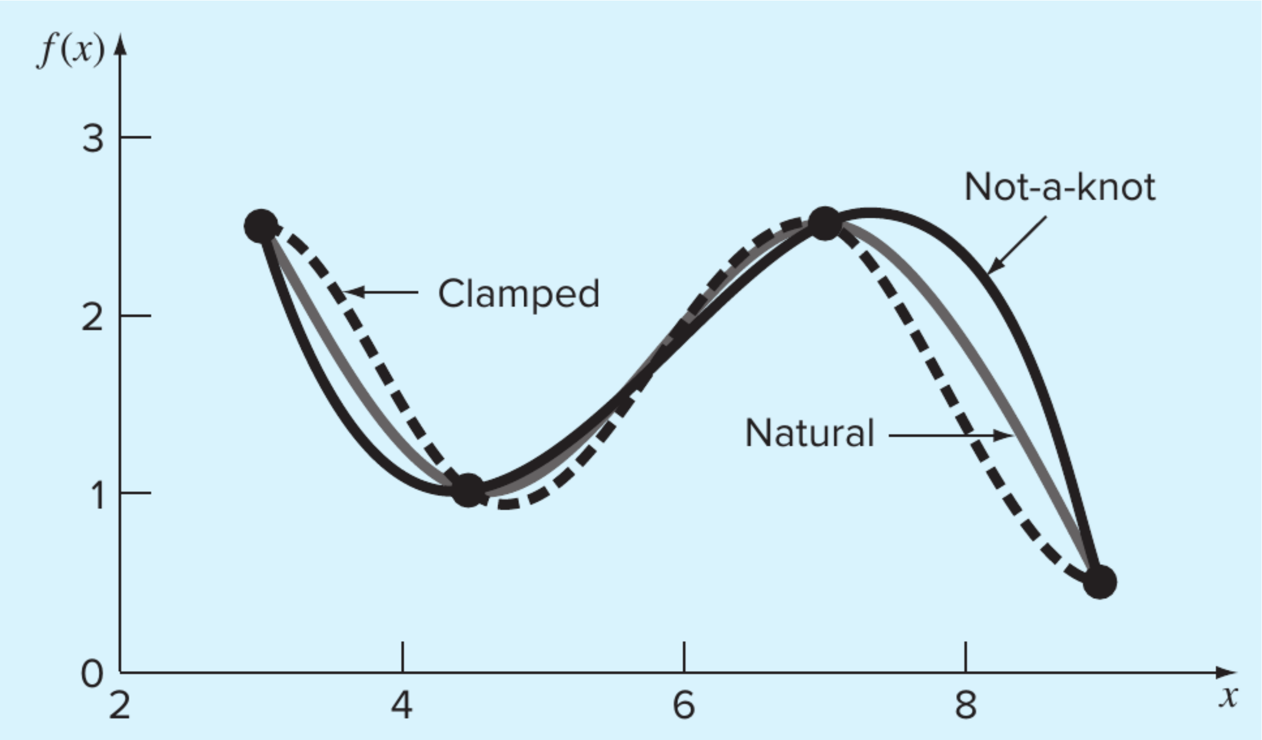
$$f[x_i, x_j] = \frac{f_i - f_j}{x_i - x_j}$$

- The endpoints of the data will have the slightly different condition that the 2nd derivatives are set to 0. This is a choice that produces what is called the **natural spline**.

- If we know the first derivatives at the endpoints we can input them as a condition instead. This is known as a **clamped** end condition.

- If we make the 3rd order derivatives continuous at the 2nd and 2nd-to-last points then the same cubic functions are used in the first and last adjacent segments.

- This is known as a **not-a-knot** end condition and has the effect of producing more curvature.



$$\begin{bmatrix} 1 & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & \ddots & \ddots & \ddots & \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ & & & & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{Bmatrix} = \begin{Bmatrix} 0 \\ 3(f[x_3, x_2] - f[x_2, x_1]) \\ \vdots \\ 3(f[x_n, x_{n-1}] - f[x_{n-1}, x_{n-2}]) \\ 0 \end{Bmatrix}$$

Solve the c_i coefficients then use the formulas below to get the others

$$a_i = f_i \qquad b_i = \frac{f_{i+1} - f_i}{h_i} - \frac{h_i}{3}(2c_i + c_{i+1}) \qquad d_i = \frac{c_{i+1} - c_i}{3h_i}$$

EXAMPLE 9 Fit the following data with cubic splines.

i	x_i	f_i
1	3.0	2.5
2	4.5	1.0
3	7.0	2.5
4	9.0	0.5

$$\begin{bmatrix} 1 & & & \\ h_1 & 2(h_1+h_2) & h_2 & \\ & h_2 & 2(h_2+h_3) & h_3 \\ & & & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 3(f[x_3,x_2]-f[x_2,x_1]) \\ 3(f[x_4,x_3]-f[x_3,x_2]) \\ 0 \end{Bmatrix}$$

$$h_1 = 4.5 - 3.0 = 1.5$$
$$h_2 = 7.0 - 4.5 = 2.5$$
$$h_3 = 9.0 - 7.0 = 2.0$$

$$f_1 = 2.5 \qquad f_3 = 2.5$$
$$f_2 = 1.0 \qquad f_4 = 0.5$$

$$\begin{bmatrix} 1 & & & \\ 1.5 & 8 & 2.5 & \\ & 2.5 & 9 & 2 \\ & & & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 4.8 \\ -4.8 \\ 0 \end{Bmatrix}$$

- Solving in Matlab gives:

$$c_1 = 0 \qquad c_2 = 0.839543726$$
$$c_3 = -0.766539924 \qquad c_4 = 0$$

- Using the equations for the coefficients:

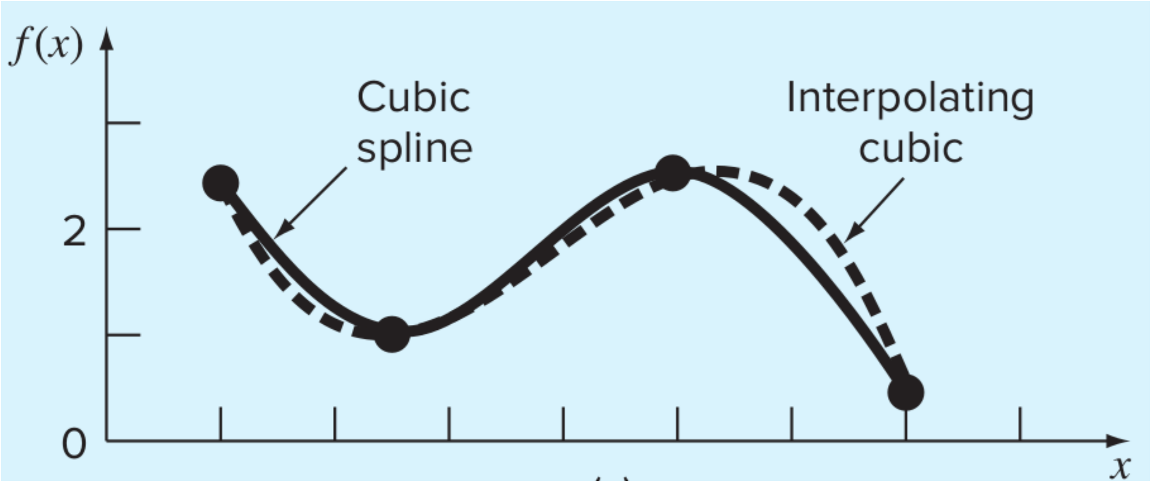
$$b_1 = -1.419771863 \qquad d_1 = 0.186565272$$
$$b_2 = -0.160456274 \qquad d_2 = -0.214144487$$
$$b_3 = 0.022053232 \qquad d_3 = 0.127756654$$

- The splines for each subinterval are:

$$s_1(x) = 2.5 - 1.419771863(x-3) + 0.186565272(x-3)^3$$

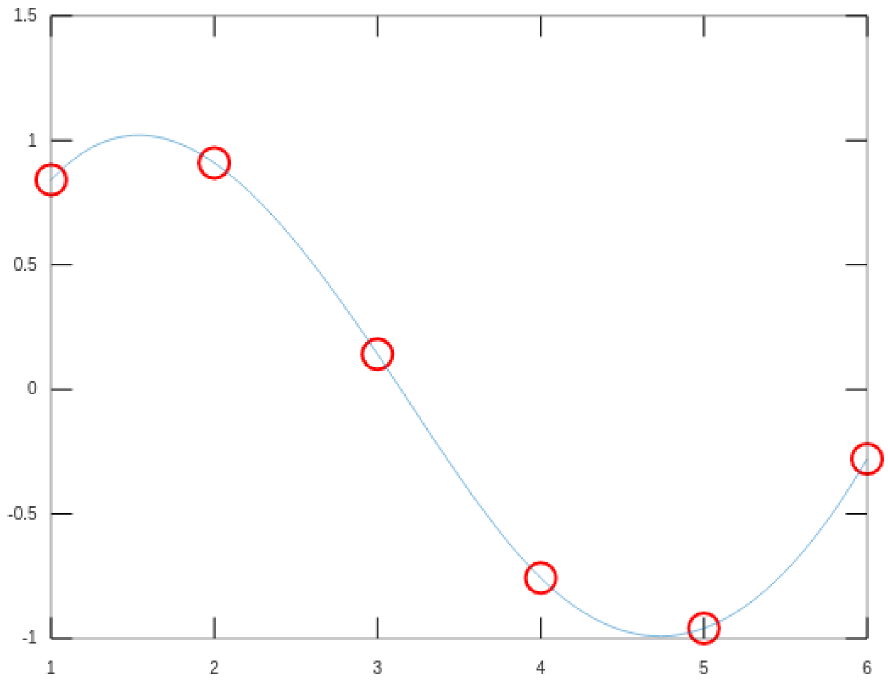
$$s_2(x) = 1.0 - 0.160456274(x-4.5) + 0.839543726(x-4.5)^2 - 0.214144487(x-4.5)^3$$

$$s_3(x) = 2.5 + 0.022053232(x-7.0) - 0.766539924(x-7.0)^2 + 0.127756654(x-7.0)^3$$



- To estimate a function value at an x we must choose the correct spline function.

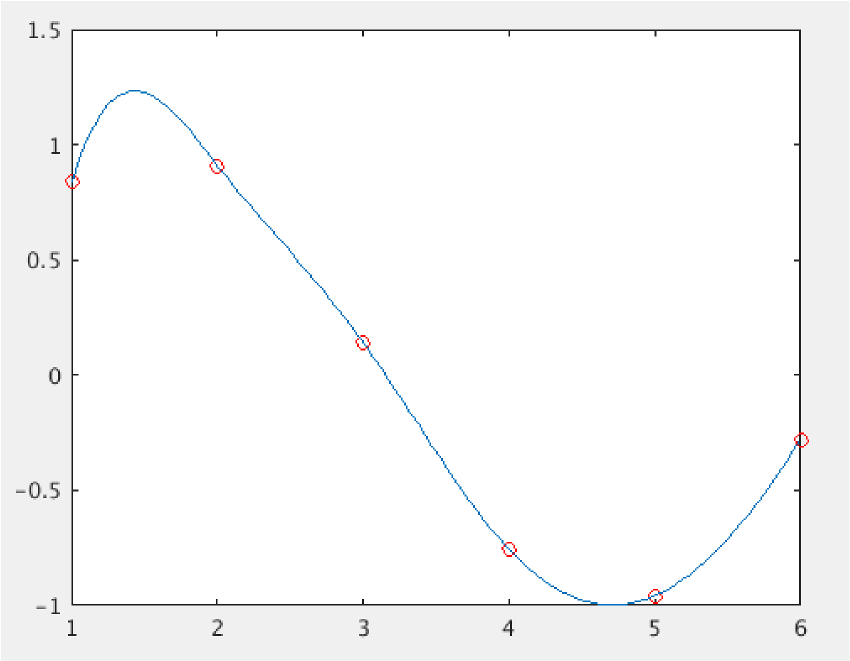
- Splines can be implemented in Matlab using the **spline** function:




```
>> x = 1:6;
>> y = sin(x);

>> xfit = linspace(1,6);
>> yfit = spline(x,y,xfit);

>> plot(x,y,'or','MarkerSize',6,xfit,yfit)
```

- By default Matlab will use not-a-knot end condition however the clamped end condition can be used by specifying the first derivatives of the end points as the first and last values of the data vector.

```
>> x = 1:6;
>> y = [2 sin(x) 1];

>> xfit = linspace(1,6);
>> yfit = spline(x,y,xfit);

>> plot(x,y,'or','MarkerSize',6)
>> hold on
>> plot(xfit,yfit)
```



9.9 Matlab’s interp Function

- Using the **interp1** function you can select a suitable method of interpolation for your data.

- There are 9 methods available with the main 4 variants being *linear*, *nearest neighbour*, *spline*, and *pchip*.

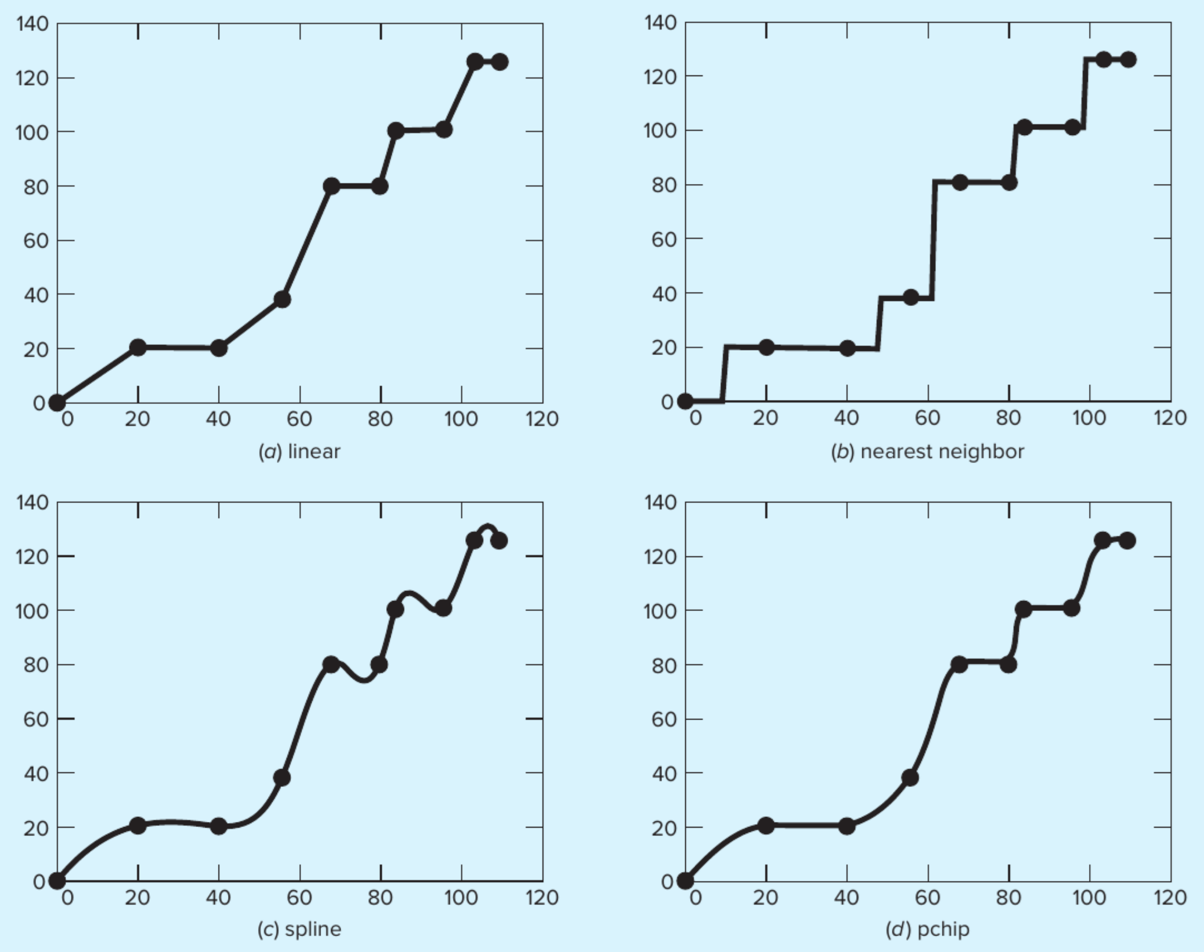
- Nearest neighbour interpolates the value to the nearest data point in the sample. Pchip flattens the curve when cubic splines causes overshoot.

- The syntax is as follows:

```
>> interp1(x,y,xx,'method')
```

EXAMPLE 10 Compare the 4 main methods of interp to the following car velocity data. Note there is no deceleration.

t	0	20	40	56	68	80	84	96	104	110
v	0	20	20	38	80	80	100	100	125	125



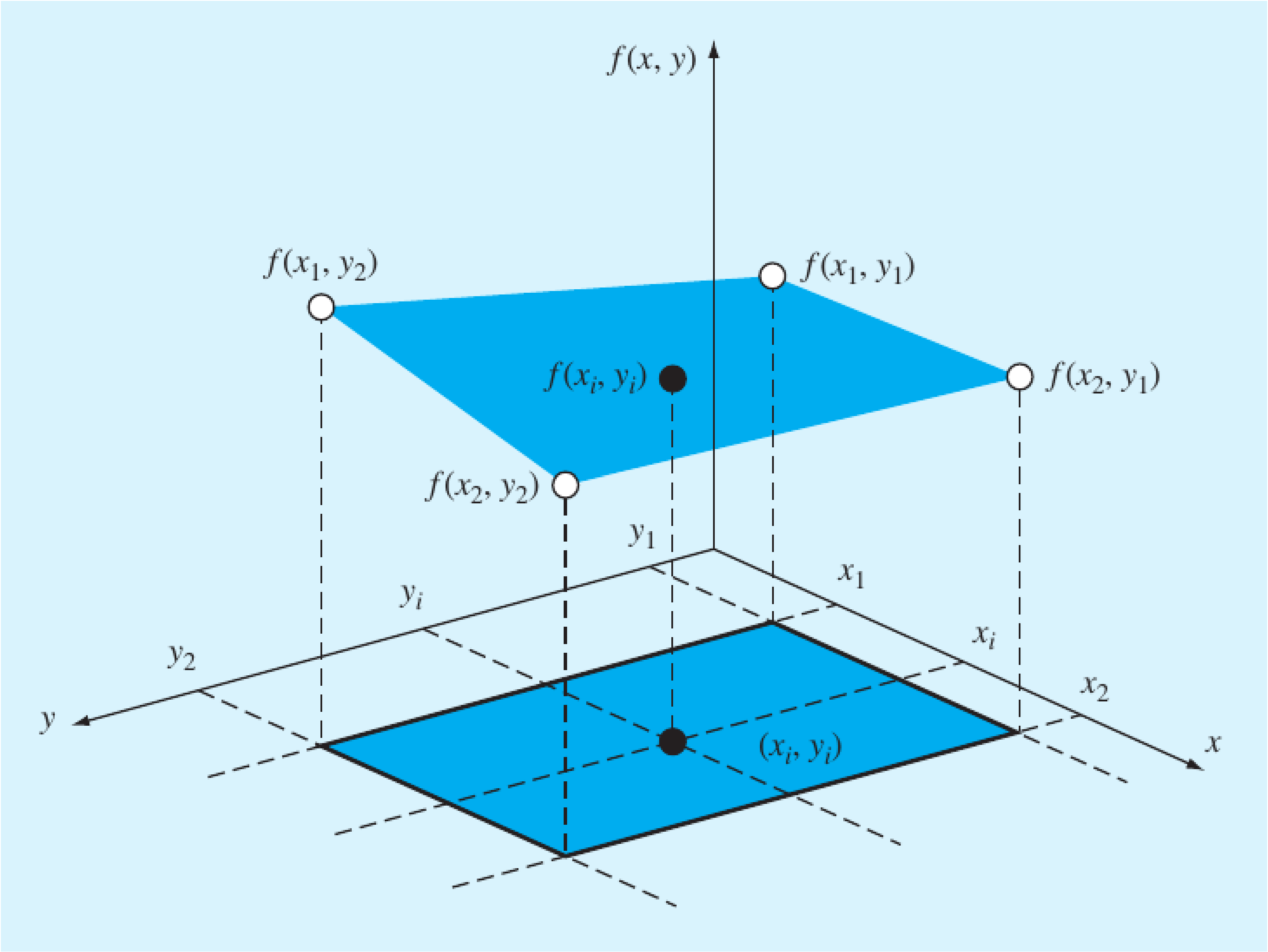
- Note the overshoot for the cubic spline interpolation between 60 and 110 seconds.

9.10 Multidimensional Interpolation

- To do this we hold all variables fixed except 1, then do 1-dimensional interpolation along the direction of this single variable.

- We then use those points to interpolate in another direction while holding all other variables fixed.

- The **interp2** and **interp3** functions do this for 2 and 3 variables respectively using the same methods as **interp1**.



$f(x, y)$
$f(x_1, y_2)$
$f(x_1, y_1)$
$f(x_i, y_i)$
$f(x_2, y_1)$
$f(x_2, y_2)$
y_1
x_1
y_i
x_i
y_2
x_2
y
(x_i, y_i)
x

EXAMPLE 11 The temperature at a number of coordinates on the surface of a rectangular heated plate are measured as follows. Estimate the temperature at x = 5.25 and y = 4.8.

$$T(2, 1) = 60 \qquad T(9, 1) = 57.5$$

$$T(2, 6) = 55 \qquad T(9, 6) = 70$$

```
>>  x = [ 2 9 ];
>>  y = [ 1 6 ];

>>  T = [ 60 57.5 ; 55 70 ];

>>  interp2( x , y , T , 5.25 , 4.8 )

ans =

      61.2143
```