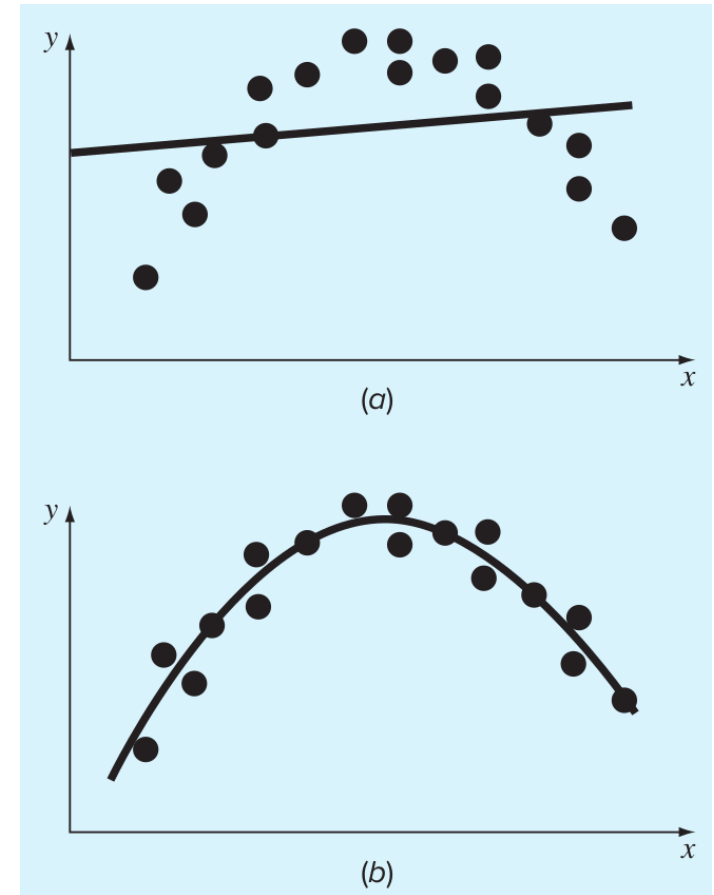


8.1 Polynomial Regression

- ❑ Instead of fitting a straight line to data, or using a nonlinear relationship with linearisation techniques, we can choose to fit an n th degree polynomial to a data set.
- ❑ A 2nd degree polynomial would have the following equation where e is the error (**residual**):

$$y = a_0 + a_1x + a_2x^2 + e$$



- Summing the squares of the residuals gives:

$$S_r = \sum_{i=1}^n (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$$

- Setting the partial derivatives equal to 0 yields the formulae for the constants a_0 , a_1 and a_2 .

$$\frac{\partial S_r}{\partial a_0} = -2 \sum (y_i - a_0 - a_1 x_i - a_2 x_i^2) = 0$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum x_i (y_i - a_0 - a_1 x_i - a_2 x_i^2) = 0$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum x_i^2 (y_i - a_0 - a_1 x_i - a_2 x_i^2) = 0$$

2nd degree polynomial regression equations

$$(n)a_0 + (\sum x_i)a_1 + (\sum x_i^2)a_2 = \sum y_i$$

$$(\sum x_i)a_0 + (\sum x_i^2)a_1 + (\sum x_i^3)a_2 = \sum x_i y_i$$

$$(\sum x_i^2)a_0 + (\sum x_i^3)a_1 + (\sum x_i^4)a_2 = \sum x_i^2 y_i$$

- Note that this is simply a system of **3 linear equations** where a_0 , a_1 and a_2 are the variables.
- Note also that **if there are less than 3 data points we cannot solve the system** to get a unique solution.
- Instead of rearranging to find explicit formulae (which can get complicated) we can simply calculate the sums then **solve the linear system** using any method from linear algebra (using Matlab functions to speed up this process).

EXAMPLE 1 Fit a 2nd degree polynomial to the following data.

x_i	y_i	$(y_i - \bar{y})^2$	$(y_i - a_0 - a_1x_i - a_2x_i^2)^2$
0	2.1	544.44	0.14332
1	7.7	314.47	1.00286
2	13.6	140.03	1.08160
3	27.2	3.12	0.80487
4	40.9	239.22	0.61959
5	61.1	1272.11	0.09434
Σ	152.6	2513.39	3.74657

$$\Sigma x_i = 15 \quad \Sigma x_i^4 = 979 \quad \Sigma x_i^2 = 55 \quad \Sigma x_i^2 y_i = 2488.8$$

$$\Sigma y_i = 152.6 \quad \Sigma x_i y_i = 585.6 \quad \Sigma x_i^3 = 225 \quad n = 6$$

$$\longrightarrow \begin{bmatrix} 6 & 15 & 55 \\ 15 & 55 & 225 \\ 55 & 225 & 979 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} 152.6 \\ 585.6 \\ 2488.8 \end{Bmatrix}$$

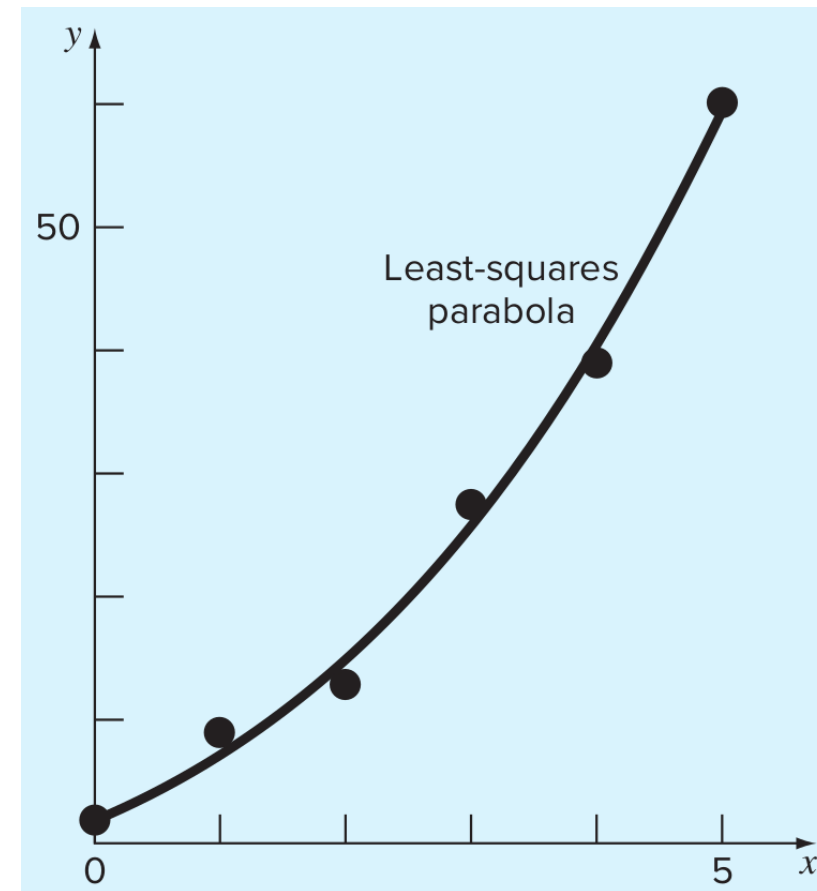
- Solving using Matlab's left-division operator we have:

```
>> N = [6 15 55;15 55 225;55 225 979];  
>> r = [152.6 585.6 2488.8];  
>> a = N\r
```

```
a =  
    2.4786  
    2.3593  
    1.8607
```

- This gives the fitted quadratic curve as:

$$y = 2.4786 + 2.3593x + 1.8607x^2$$



- ❑ The correlation coefficient is calculated in similar way to how we did it for linear regression.
- ❑ We use the relative comparison of the spread from the mean:

$$r^2 = \frac{S_t - S_r}{S_t}$$

where,
$$S_r = \sum_{i=1}^n (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$$

$$S_t = \sum (y_i - \bar{y})^2$$

→
$$r^2 = \frac{2513.39 - 3.74657}{2513.39} = 0.99851$$

Polynomial of Degree m

- We can use the same idea to fit any degree polynomial we like to our data provided we have enough data points (no. of data points must be at least $m + 1$).

$$y = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m + e$$

$$\longrightarrow S_r = \sum_{i=1}^n (y_i - (a_0 + a_1x_i + a_2x_i^2 + \cdots + a_mx_i^m))^2$$

- From differentiating with respect to the $m + 1$ coefficients and setting equal to 0 we obtain $m + 1$ linear equations that can be solved using the Matlab left-division operator: $0 \leq k \leq m$

$$a_0 \sum x^k + a_1 \sum x^{k+1} + a_2 \sum x^{k+2} + \cdots + a_m \sum x^{k+m} = \sum x^k y$$

Standard Error

- We can compute a **standard error** (analogous to standard deviation of data) from the following equation:

$$s_{y/x} = \sqrt{\frac{S_r}{n - (m + 1)}}$$

where n is the number of data points and m is the degree of the polynomial. For linear regression with degree 1, we have:

$$s_{y/x} = \sqrt{\frac{S_r}{n - 2}}$$

- The denominator is equivalent to Bessel's correction in the standard deviation formula in which the sample spread is calculated by dividing by the number of degrees of freedom of the system.

Correlation Coefficient

- We use the same idea to calculate the correlation coefficient:

$$S_t = \sum (y_i - \bar{y})^2$$
$$\longrightarrow r^2 = \frac{S_t - S_r}{S_t}$$

EXAMPLE 2 Calculate the standard error and correlation coefficient for the data in **Example 1**.

x_i	y_i	$(y_i - \bar{y})^2$	$(y_i - a_0 - a_1x_i - a_2x_i^2)^2$
0	2.1	544.44	0.14332
1	7.7	314.47	1.00286
2	13.6	140.03	1.08160
3	27.2	3.12	0.80487
4	40.9	239.22	0.61959
5	61.1	1272.11	0.09434
Σ	152.6	2513.39	3.74657

$$\bar{y} = 25.433$$

$$m = 2$$

$$s_{y/x} = \sqrt{\frac{3.74657}{6 - (2 + 1)}} = 1.1175$$

$$n = 6$$

$$r^2 = \frac{2513.39 - 3.74657}{2513.39} = 0.99851 \longrightarrow r = 0.99925$$

$S_{y/x}$ vs. r^2

- ❑ The **regression coefficient is a relative measure** which helps us compare best fit lines of different data sets with each other.
- ❑ The **standard error gives us an absolute measure** that is useful in calculations that are specific to one particular data set.
- ❑ Both are useful depending on the context.

EXAMPLE 3 Consider the following 3 data sets for hours spent studying vs. exam scores and their respective regression coefficients and standard errors.

Class 1

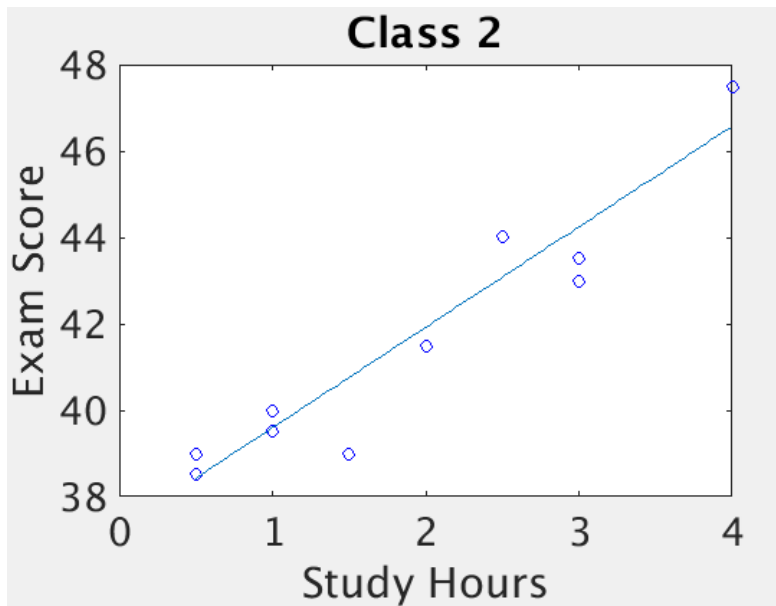
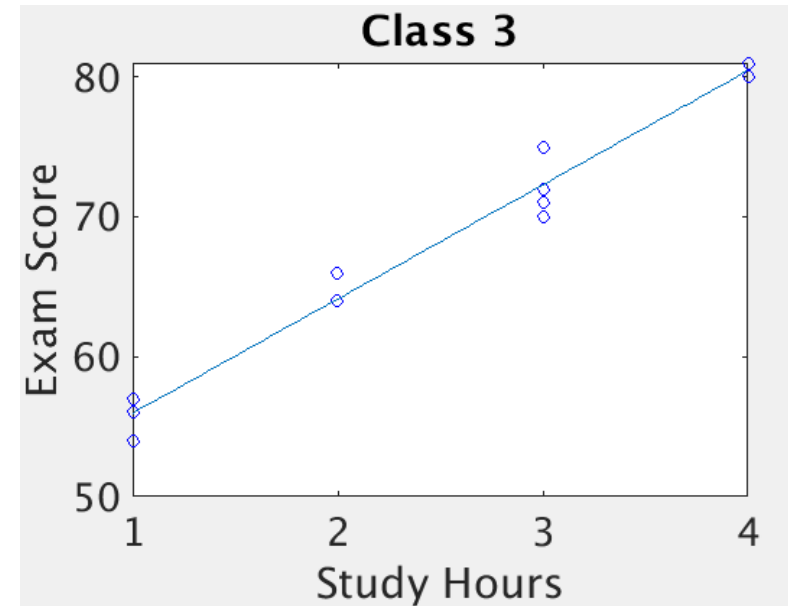
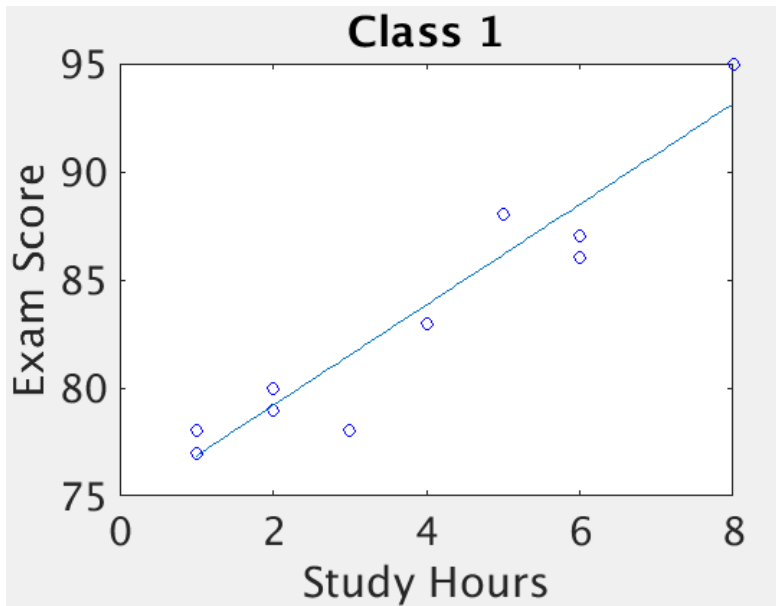
Study Hours	Exam Score
1	78
1	77
2	80
2	79
2	80
3	78
4	83
5	88
5	88
6	87
6	86
8	95

Class 2

Study Hours	Exam Score
0.5	39
0.5	38.5
1	40
1	39.5
1	40
1.5	39
2	41.5
2.5	44
2.5	44
3	43.5
3	43
4	47.5

Class 3

Study Hours	Exam Score
1	56
1	57
1	54
1	56
2	64
2	66
3	71
3	70
3	72
3	75
4	81
4	80



Class 1

$$S_{y/x} = 4.1430 \quad r = 0.9479$$

Class 2

$$S_{y/x} = 2.0715 \quad r = 0.9479$$

Class 3

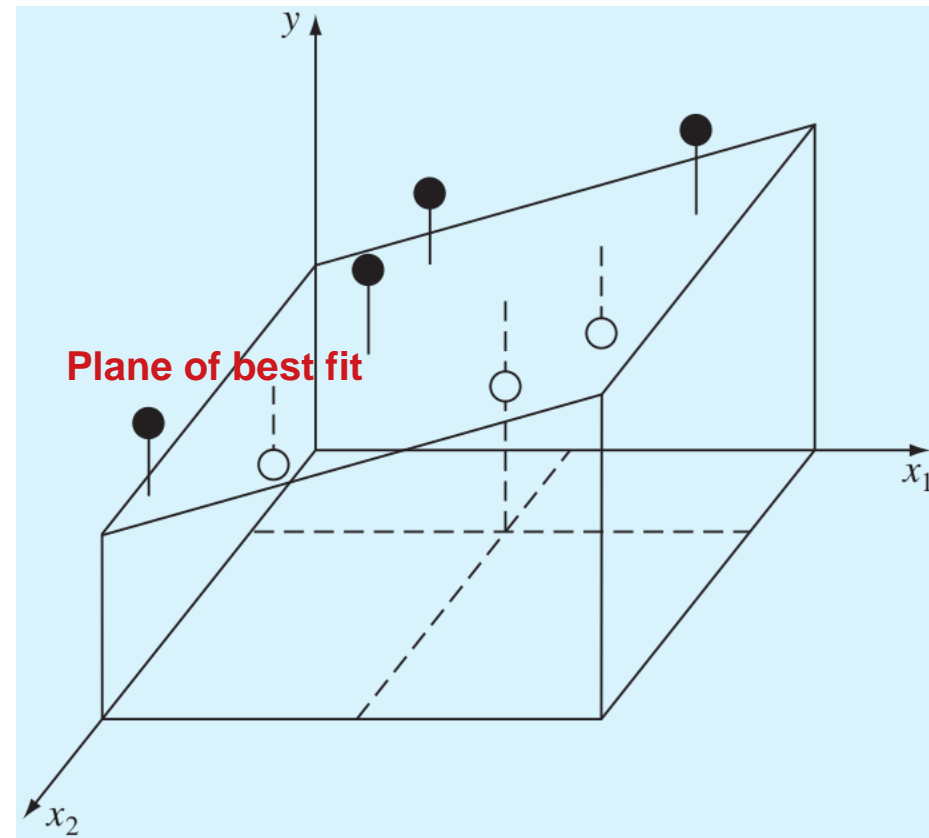
$$S_{y/x} = 3.4129 \quad r = 0.9883$$

- ❑ The regression coefficients for Classes 1 & 2 are the same.
- ❑ This indicates that the spread of each data set from the regression lines is the same. However the standard error for Class 2 is half of Class 1 which means it gives a more precise prediction.
- ❑ The standard error for Class 3 is larger than Class 2 which means the regression line will not give as precise a prediction.
- ❑ However when we compare the regression coefficients the Class 3 regression line is a better fit to the data than the Class 2 regression line.

8.2 Multiple Linear Regression

- If we observe that some data has a linear relationship with **more than one variable** we can do multiple linear regression.
- The result is a “hyperplane” of best fit. This is best visualised for some observed data, y , that has a suspected linear relationship with 2 variables, x_1 and x_2 .

$$y = a_0 + a_1x_1 + a_2x_2 + e$$



- We proceed in the usual way, by doing partial differentiation with respect to each coefficient and setting equal to 0 to obtain the equations that we must solve in order to get the regression plane.

$$\frac{\partial S_r}{\partial a_0} = -2 \sum (y_i - a_0 - a_1 x_{1,i} - a_2 x_{2,i}) = 0$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum x_{1,i} (y_i - a_0 - a_1 x_{1,i} - a_2 x_{2,i}) = 0$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum x_{2,i} (y_i - a_0 - a_1 x_{1,i} - a_2 x_{2,i}) = 0$$

- We can solve these equations using the following linear system.

System to Solve for Regression in 2 Variables

$$\begin{bmatrix} n & \sum x_{1,i} & \sum x_{2,i} \\ \sum x_{1,i} & \sum x_{1,i}^2 & \sum x_{1,i} x_{2,i} \\ \sum x_{2,i} & \sum x_{1,i} x_{2,i} & \sum x_{2,i}^2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} \sum y_i \\ \sum x_{1,i} y_i \\ \sum x_{2,i} y_i \end{Bmatrix}$$


EXAMPLE 4 Find the best fitting linear function of 2 variables for the data shown on the left.

x_1	x_2	y
0	0	5
2	1	10
2.5	2	9
1	3	0
4	6	3
7	2	27

$$\begin{bmatrix} 6 & 16.5 & 14 \\ 16.5 & 76.25 & 48 \\ 14 & 48 & 54 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} 54 \\ 243.5 \\ 100 \end{Bmatrix}$$

$$a_0 = 5 \quad a_1 = 4 \quad a_2 = -3$$

$$y = 5 + 4x_1 - 3x_2$$



y	x_1	x_2	x_1^2	x_2^2	x_1x_2	x_1y	x_2y
5	0	0	0	0	0	0	0
10	2	1	4	1	2	20	10
9	2.5	2	6.25	4	5	22.5	18
0	1	3	1	9	3	0	0
3	4	6	16	36	24	12	18
27	7	2	49	4	14	189	54
<u>54</u>	<u>16.5</u>	<u>14</u>	<u>76.25</u>	<u>54</u>	<u>48</u>	<u>243.5</u>	<u>100</u>

Multiple Linear Regression For m Variables

- We assume a linear relationship between the data and m variables now:

$$y = a_0 + a_1x_1 + a_2x_2 + \cdots + a_mx_m + e$$

- We must take $m + 1$ partial derivatives and set them equal to 0 as before then solve for the constants.

$$\begin{bmatrix} n & \sum x_{1,i} & \sum x_{2,i} & \cdots & \sum x_{m,i} \\ \sum x_{1,i} & \sum x_{1,i}^2 & \sum x_{1,i}x_{2,i} & \cdots & \sum x_{1,i}x_{m,i} \\ \vdots & \vdots & \vdots & & \vdots \\ \sum x_{m,i} & \sum x_{m,i}x_{1,i} & \sum x_{m,i}x_{2,i} & \cdots & \sum x_{m,i}^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_{1,i}y_i \\ \vdots \\ \sum x_{m,i}y_i \end{bmatrix}$$

- We compute the standard error and correlation coefficient using the same formulae as before.

Application to Power Equations

- A common engineering fitting equation for multiple variables is the following power equation:

$$y = a_0 x_1^{a_1} x_2^{a_2} \cdots x_m^{a_m}$$

- Taking logs yields a linearisation that can be solved for the coefficients using multiple linear regression analysis.

$$\log y = \log a_0 + a_1 \log x_1 + a_2 \log x_2 + \cdots + a_m \log x_m$$

8.3 General Linear Least Squares

- We can now generalise the method as follows:

$$y = a_0 z_0 + a_1 z_1 + a_2 z_2 + \cdots + a_m z_m + e$$

- The linear part of the method refers to the coefficients a_0, a_1 etc. so the **z-functions can be any functions that form a basis** for the fitted function.

Polynomial regression

$$z_0 = 1, z_1 = x, z_2 = x^2, \dots, z_m = x^m$$

Sinusoidal regression

$$z_0 = \sin(\omega x), \quad z_1 = \sin(\omega x), \quad \dots, z_m = \sin(m\omega x)$$

- The sinusoidal case and other related cases are used in **Fourier analysis**.

- We can express the problem in matrix notation as:

$$\{y\} = [Z]\{a\} + \{e\}$$

$$[Z] = \begin{bmatrix} z_{01} & z_{11} & \cdots & z_{m1} \\ z_{02} & z_{12} & \cdots & z_{m2} \\ \vdots & \vdots & & \vdots \\ z_{0n} & z_{1n} & \cdots & z_{mn} \end{bmatrix}$$

Value of basis functions at the measured values

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad e = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

Data values and residuals at the measured values

$$a = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix}$$

Unknown coefficients

- The sum of the square of the residuals is:

$$S_r = \sum_{i=1}^n \left(y_i - \sum_{j=0}^m a_j z_{ji} \right)^2$$

- Setting the partial derivatives with respect to each coefficient, a_k , to 0:

$$\begin{aligned} & \frac{\partial}{\partial a_k} \sum_{i=1}^n (y_i - a_0 z_{0i} - a_1 z_{1i} - \cdots - a_m z_{mi})^2 \\ &= -2 \sum_{i=1}^n (y_i - a_0 z_{0i} - a_1 z_{1i} - \cdots - a_m z_{mi}) z_{ki} \\ &= -2 \sum_{i=1}^n \left(y_i z_{ki} - \sum_{j=0}^m a_j z_{ji} z_{ki} \right) = 0 \end{aligned}$$

- Putting the y_i sums on the right hand side we have:

$$\sum_{i=1}^n \sum_{j=0}^m a_j z_{ji} z_{ki} = \sum_{i=1}^n y_i z_{ki}$$

- We can also write this in matrix notation:

$$\mathbf{Z}^T \mathbf{Z} \mathbf{a} = \mathbf{Z}^T \mathbf{y}$$

- This can be solved for \mathbf{a} using the Matlab left-division operator:

$$>> \mathbf{a} = (\mathbf{Z}' * \mathbf{Z}) \setminus (\mathbf{Z}' * \mathbf{y})$$

- The standard error and correlation coefficient are computed using the same formulae as before.

EXAMPLE 5 Repeat **Example 1** using the matrix notation defined for general linear regressions.

```
>> x = [0 1 2 3 4 5]';  
>> y = [2.1 7.7 13.6 27.2 40.9 61.1]';
```

Enter the data

```
>> Z = [ones(size(x)) x x.^2]
```

Calculate Z matrix
($z_0 = 1$, $z_1 = x$, $z_2 = x^2$)

```
Z =  
    1     0     0  
    1     1     1  
    1     2     4  
    1     3     9  
    1     4    16  
    1     5    25
```

Calculate the coefficients

```
>> a = (Z'*Z)\(Z'*y)
```

```
ans =  
    2.4786  
    2.3593  
    1.8607
```

$$y = 2.4786 + 2.3593x + 1.8607x^2$$



Or go to www.pollev.com/jsands601

Which of the following equations cannot be fitted using general linear least squares?

$$y \approx a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3$$

$$y \approx a_0 e^{x^2} + a_1 e^{x^2} + a_2 e^{x^2} + a_3 e^{x^2}$$

Tc



0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

8.4 Nonlinear Regression

- Sometimes we want to fit an equation to some data and the relationship with the unknown coefficients is nonlinear, for example:

$$y = a_0(1 - e^{-a_1x}) + \text{error}$$

- Now we can compute the sums of the squares of the residuals as:

$$S_r = \sum_{i=1}^n [y_i - a_0(1 - e^{-a_1x_i})]^2$$

- We obtain a system of equations, $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, where each F_k is given by the the partial derivative of S_r with respect to a_k .

- The resulting system of nonlinear equations can be solved using Newton's method of root finding for nonlinear systems:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}^{-1}(\mathbf{x}_n) \mathbf{F}(\mathbf{x}_n)$$

- Alternatively Matlab's built-in function **fminsearch** can be used to find the values of the a 's that minimise the function (**Nelder-Mead algorithm**).

EXAMPLE 6

Fit the data from the wind tunnel example from Lecture 7 to a power equation using nonlinear regression instead of transforming to a linear model and using linear least squares.

$$F = \alpha v^\beta$$

i	x_i	y_i
1	10	25
2	20	70
3	30	380
4	40	550
5	50	610
6	60	1,220
7	70	830
8	80	1,450
Σ	360	5,135

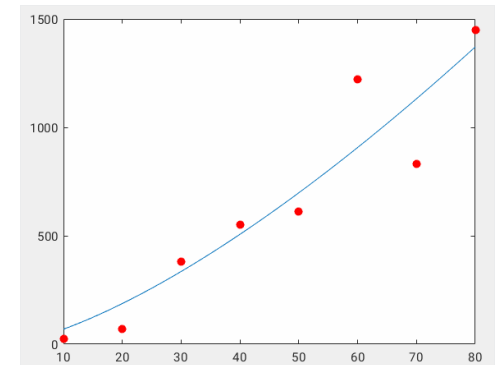
`[x, fval] = fminsearch(fun,x0,options,p1,p2,...)`

- ❑ Create a function file to hold the error function.

```
function Sr = fit_power(a,x_data,y_data)
% a = [alpha, beta]
F_power = a(1)*x_data.^a(2);
Sr = sum((y_data - F_power).^2);
```

- ❑ Use **fminsearch** to find values of α and β that minimise the function.

```
>> x = [10 20 30 40 50 60 70 80];
>> y = [25 70 380 550 610 1220 830 1450];
>> A = fminsearch(@fit_power,[1,1],[],x,y)
A =
    2.5384    1.4359
>> F = @(v) A(1)*v.^(A(2));
>> plot(x,y,'or','MarkerSize',6,'Markerfacecolor','r')
>> hold on
>> fplot(F,[10,80])
```



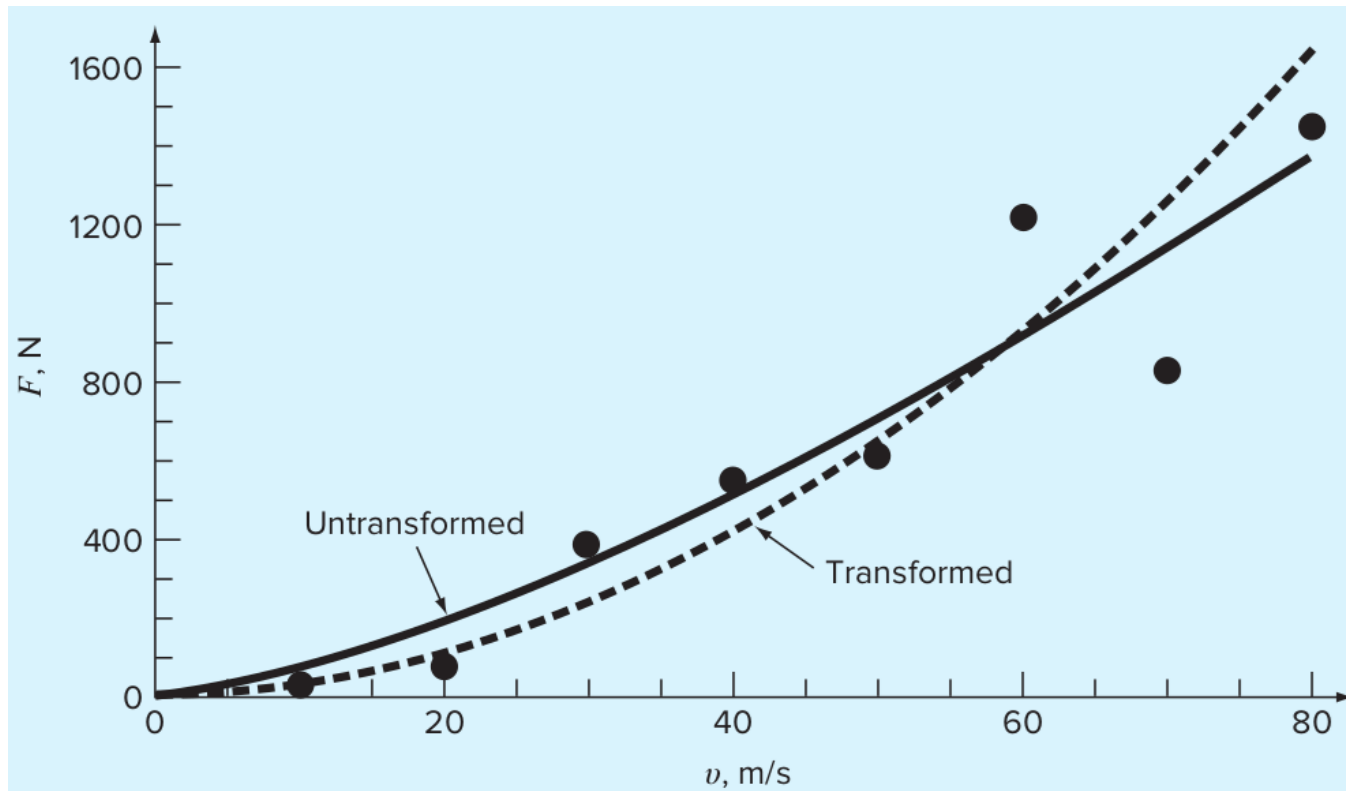
- We got different “best fit” parameters using nonlinear regression than when we linearised the power equation and did linear least squares. Both seem to give a reasonable fit.

Nonlinear regression

$$F = 2.5384v^{1.4359}$$

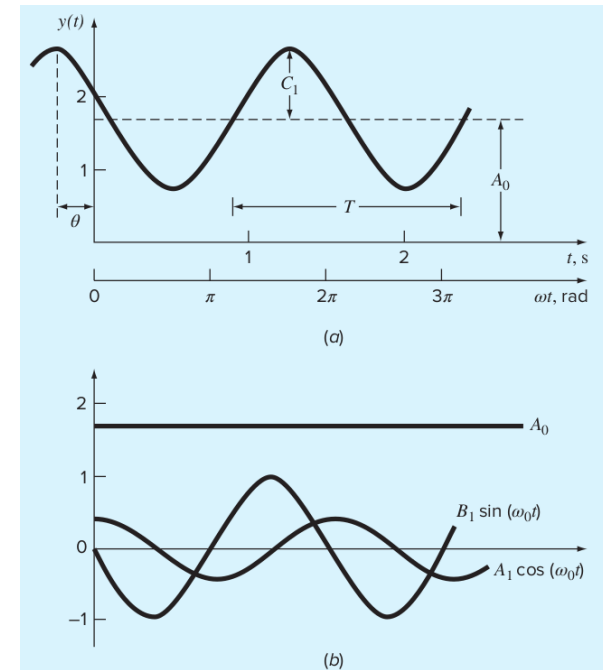
Linearised power equation

$$F = 0.2741v^{1.9842}$$



8.5 Fitting Sinusoids

- If we have periodic functions Fourier showed that we can write them as a linear combination of sines and cosines. In other words the basis functions in this nonlinear regression are trig functions.



$$f(t) = a_0 + a_1 \cos(\omega_0 t) + b_1 \sin(\omega_0 t) + a_2 \cos(2\omega_0 t) + b_2 \sin(2\omega_0 t) + \dots$$

- We wish to minimise the square of the residuals as usual:

$$S_r = \sum_{i=1}^N \left[y_i - \sum_{j=0}^m f(t_j) \right]^2$$

- In general, the more sines/cosines we use the more accurate our fit is.

EXAMPLE 7 Fit the following data to the simplified case of a Fourier series given with $\omega_0 = 4.189$.

$$f(t) = A_0 + A_1 \cos(\omega_0 t) + B_1 \sin(\omega_0 t)$$

- ❑ The basis functions are $z_0 = 1$, $z_2 = \cos(\omega_0 t)$, $z_3 = \sin(\omega_0 t)$.
- ❑ We have the following from the multiple linear regression form:

t	y
0	2.200
0.15	1.595
0.30	1.031
0.45	0.722
0.60	0.786
0.75	1.200
0.90	1.805
1.05	2.369
1.20	2.678
1.35	2.614
$\Sigma =$	17.000

$$\begin{bmatrix} N & \sum \cos(\omega_0 t) & \sum \sin(\omega_0 t) \\ \sum \cos(\omega_0 t) & \sum \cos^2(\omega_0 t) & \sum \cos(\omega_0 t) \sin(\omega_0 t) \\ \sum \sin(\omega_0 t) & \sum \cos(\omega_0 t) \sin(\omega_0 t) & \sum \sin^2(\omega_0 t) \end{bmatrix} \begin{Bmatrix} A_0 \\ B_1 \\ B_1 \end{Bmatrix} = \begin{Bmatrix} \sum y \\ \sum y \cos(\omega_0 t) \\ \sum y \sin(\omega_0 t) \end{Bmatrix}$$

- ❑ We calculate the values of the coefficient matrix:

```
>> w = 4.189;  
>> c1 = cos(w*t);  
>> c2 = sin(w*t);  
>> c3 = cos(w*t).^2;  
>> c4 = sin(w*t).^2;  
>> c5 = cos(w*t).*sin(w*t);  
>> c6 = y.*cos(w*t);  
>> c7 = y.*sin(w*t);
```

- ❑ Calculate the matrix and Y vector:

```
>> A = [length(t) sum(c1) sum(c2); sum(c1) sum(c3) sum(c5);  
sum(c2) sum(c5) sum(c4)]
```

```
A =  
    10.0000    0.0005   -0.0002  
    0.0005    5.0002   -0.0002  
   -0.0002   -0.0002    4.9998
```

```
>> b = [sum(y); sum(c6); sum(c7)]
```

```
b =  
    17.0000  
     2.5015  
    -4.3305
```

❑ Solve the system: `>> a = A\b`

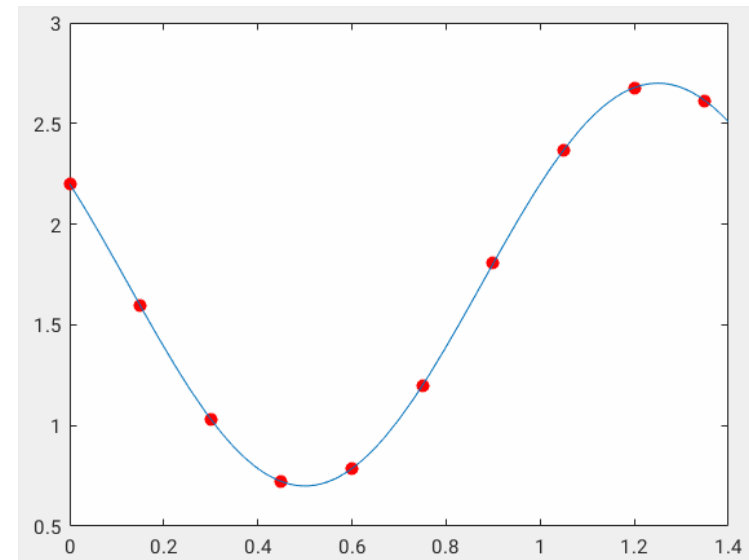
```
a =  
    1.7000  
    0.5001  
   -0.8661
```

❑ So the best fit is:

$$y = 1.7 + 0.500 \cos(\omega_0 t) - 0.866 \sin(\omega_0 t)$$

❑ Plot:

```
>> f = @(t) a(1) + a(2)*cos(w*t) + a(3)*sin(w*t);  
>> plot(t,y,'or','markersize',6,  
        'markerfacecolor','r')  
>> hold on  
>> fplot(f,[0,1.4])
```



- Alternatively we can use the Z-matrix formulation:

```
>> Z = [ ones(length(t),1) cos(w*t) sin(w*t) ];
```

- Solve the system:

```
>> a = (Z' * Z) \ (Z' * y)
```

```
a =
```

```
1.7000  
0.5001  
-0.8661
```

- So the best fit is:

$$y = 1.7 + 0.500 \cos(\omega_0 t) - 0.866 \sin(\omega_0 t)$$

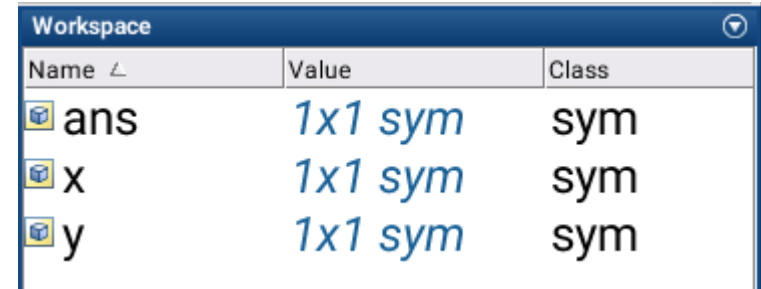
8.6 Symbolic Toolbox in Matlab

- ❑ The symbolic toolbox in Matlab is extremely useful for doing basic algebra and calculus.
- ❑ It can help to generalise your functions when you implement numerical methods that require calculus (Newton-Raphson for example).
- ❑ Keywords:

`sym syms vpa solve diff int
simplify expand factor`

- We can create symbolic variables to do algebra with:

```
>> x = sym(pi)
x =
pi
>> y = 2*x
y =
2*pi
```



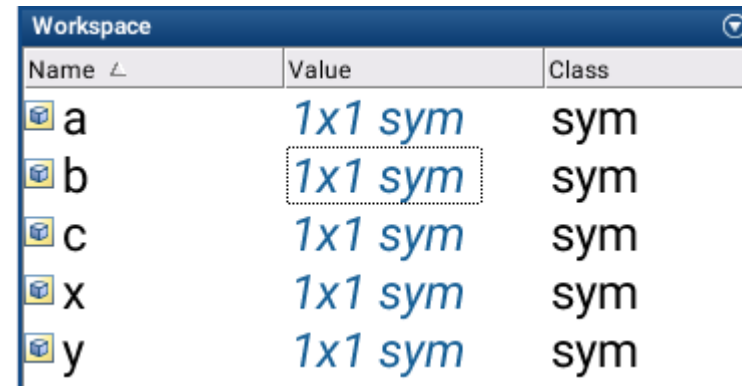
Name	Value	Class
ans	1x1 sym	sym
x	1x1 sym	sym
y	1x1 sym	sym

- We can evaluate the symbolic variable using **vpa** in order to do calculations with it.

```
>> vpa(y)
ans =
6.283185307179586476925286766559
```

- Use **syms** to create multiple symbolic variables at once.

```
>> syms x y a b c
```

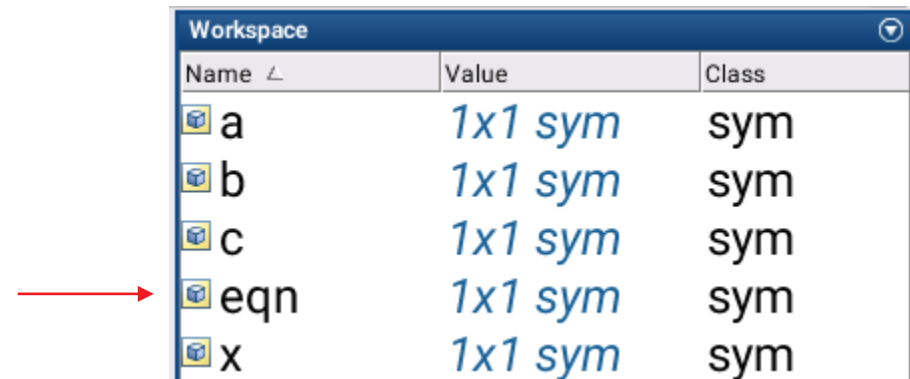


Name	Value	Class
a	1x1 sym	sym
b	1x1 sym	sym
c	1x1 sym	sym
x	1x1 sym	sym
y	1x1 sym	sym

8.7 Solving Symbolic Equations

- If we define a new function or variable using previously defined symbolic variables then Matlab **automatically creates a new symbolic variable** for us:

```
>> eqn = x^2 + y^2 == 4  
eqn =  
x^2 + y^2 == 4
```



Name	Value	Class
a	1x1 sym	sym
b	1x1 sym	sym
c	1x1 sym	sym
eqn	1x1 sym	sym
x	1x1 sym	sym

- We can then **solve** the equation with respect to any variable we like:

```
>> X = solve(eqn, x)  
X =
```

$$(2 - x)^{1/2} * (x + 2)^{1/2} = \sqrt{4 - x^2}$$

$$\begin{aligned} &(2 - y)^{1/2} * (y + 2)^{1/2} \\ &-(2 - y)^{1/2} * (y + 2)^{1/2} \end{aligned}$$

□ Or with respect to the other variable:

```
>> Y = solve(eqn, y)
Y =
    (2 - x)^(1/2)*(x + 2)^(1/2)
   -(2 - x)^(1/2)*(x + 2)^(1/2)
```

EXAMPLE 8 Obtain the quadratic formula in Matlab.

```
>> syms x y a b c
>> quad = a*x^2 + b*x + c == 0
```

```
quad =
```

```
a*x^2 + b*x + c == 0
```

```
>> x = solve(quad, x)
```

```
x =
```

```
-(b + (b^2 - 4*a*c)^(1/2))/(2*a)
-(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```


Calculus Using Symbolic Functions

❑ Define a symbolic function:

```
>> syms x y
>> f = x^2*exp(x)
f =
x^2*exp(x)
```

❑ **Differentiate** it:

```
>> df = diff(f)
df =
x^2*exp(x) + 2*x*exp(x)
```

❑ **Simplify** the answer:

```
>> df = simplify(df)
df =
x*exp(x)*(x + 2)
```

❑ **Integrate** the function:

```
>> F = int(f)
F =
exp(x)*(x^2 - 2*x + 2)
```

- ❑ Take partial derivatives of a multivariable function:

```
>> g = sin(x*y);
>> dgdx = diff(g,x)
dgdx =
y*cos(x*y)
>> dgdy = diff(g,y)
dgdy =
x*cos(x*y)
```

- ❑ Do partial integration:

```
>> Gx = int(g,x)
Gx =
-cos(x*y)/y
>> Gy = int(g,y)
Gy =
-cos(x*y)/x
```

Expansion point

- ❑ **Taylor** series:

```
>> exp_Taylor = taylor(exp(x),x,1,'Order',3)
exp_Taylor =
exp(1) + exp(1)*(x - 1) + (exp(1)*(x - 1)^2)/2
```

- ❑ **Expand** brackets:

```
>> expand(exp_Taylor)
ans =
(exp(1)*x^2)/2 + exp(1)/2
```

❑ Convert a symbolic function back into a **matlabFunction**.

```
>> syms x
>> f = x^2*exp(x);
>> f(2)
```

Doesn't work as a function

Index exceeds array bounds.

Error in sym/subsref (line 859)

```
R_tilde = builtin('subsref',L_tilde,Idx);
```

```
>> H = matlabFunction(f)
```

```
H =
```

```
function_handle with value:
```

```
@(x)x.^2.*exp(x)
```

```
>> H(2)
```

```
ans =
```

```
29.5562
```

EXAMPLE 9 Implement a basic Newton-Raphson method for a general function input.

```
function r = NR_general(f,x0)
G = sym(f);
dG = diff(G);
df = matlabFunction(dG);
x(1) = x0+1;
x(2) = x0;
while abs(x(end)-x(end-1)) > 0.01
    x(end+1) = x(end) - f(x(end))/df(x(end));
    if length(x) > 1002
        disp('Stopping. More than 1000 iterations reached.')
        break
    end
end
r = x(end);
disp(['Root found: ', num2str(r)])
```

```
>> f = @(x) x.^2 - 4*x + 2;
>> R = NR_general(f,1)
Root found: 0.58578
R =
    0.5858
>> f(R)
ans =
    6.0073e-06
```