1. (a) The objective function is

$$F(a_0, a_1) = (-1 - (a_0 - 4a_1 + 16a_2))^2 + (0 - (a_0 - 2a_1 + 4a_2))^2 + (2 - (a_0 + 2a_1 + 4a_2))^2 + (3 - (a_0 + 5a_1 + 25a_2))^2$$

(b) Rather than implementing the objective function manually, we have implemented the sum of squared residuals in a more general manner. In the file `objective.m`:

```
function f = objective(a)
x = [-4 -2 2 5];
y = [-1 0 2 3];
a0 = a(1);
a1 = a(2);
a2 = a(3);
f = sum((y - (a0 + a1*x + a2*x.^2)).^2);
end
```

Then running `fminsearch(@objective, [0 0 0])` results in $a_0 \approx 1.0690$, $a_1 \approx 0.4664$, $a_2 \approx -0.0152$.

(c) The normal equations are

$$\begin{aligned} na_0 + a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 &= \sum_{i=1}^{n} y_i, \\ a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 &= \sum_{i=1}^{n} x_i y_i, \\ a_0 \sum_{i=1}^{n} x_i^2 + a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum_{i=1}^{n} x_i^4 &= \sum_{i=1}^{n} x_i^2 y_i \end{aligned}$$

We use the following code to compute the coefficients:

```
x = [-4 -2 2 5];
y = [-1 0 2 3];
n = size(x,2);
LHS = [n sum(x) sum(x.^2);
       sum(x) sum(x.^2) sum(x.^3);
       sum(x.^2) sum(x.^3) sum(x.^4)];
RHS = [sum(y); sum(x.*y); sum(x.^2.*y)];
```

The equations are then

$$\begin{aligned} 4a_0 + a_1 + 49a_2 &= 4 \\ a_0 + 49a_1 + 61a_2 &= 23 \\ 49a_0 + 61a_1 + 913a_2 &= 67, \end{aligned}$$

which can be solved by running `inv(LHS)*RHS` to give $a_0 \approx 1.0690$, $a_1 \approx 0.4664$, $a_2 \approx -0.0152$.

(d) The matrices are

$$\mathbf{A} = \begin{pmatrix} 1 & -4 & 16 \\ 1 & -2 & 4 \\ 1 & 2 & 4 \\ 1 & 5 & 25 \end{pmatrix}, \quad \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} -1 \\ 0 \\ 2 \\ 3 \end{pmatrix}.$$

Then running the folllowing code will find the coefficients:

```
x = [-4 -2 2 5]';
y = [-1 0 2 3]';
A = [ones([4 1]) x x.^2];
inv(A'*A)*A'*y
```

This also gives $a_0 \approx 1.0690$, $a_1 \approx 0.4664$, $a_2 \approx -0.0152$.

2. We have

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} \mathbf{1}^T \\ \mathbf{x}^T \\ \vdots \\ (\mathbf{x}^m)^T \end{pmatrix} \begin{pmatrix} \mathbf{1} & \mathbf{x} & \dots & \mathbf{x}^m \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_1^m & x_2^m & \dots & x_n^m \end{pmatrix} \begin{pmatrix} 1 & x_1 & \dots & x_1^m \\ 1 & x_2 & \dots & x_2^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \dots & x_n^m \end{pmatrix}$$

$$= \begin{pmatrix} n & \sum_{i=1}^n x_i & \dots & \sum_{i=1}^n x_i^m \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 & \dots & \sum_{i=1}^n x_i^{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_i^m & \sum_{i=1}^n x_i^{m+1} & \dots & \sum_{i=1}^n x_i^{m+m} \end{pmatrix}$$

Then, with $\mathbf{a} = \begin{pmatrix} a_0 & a_1 & \dots & a_m \end{pmatrix}^T$, we have

$$\mathbf{A}^T\mathbf{A}\mathbf{a} = \begin{pmatrix} n & \sum_{i=1}^n x_i & \dots & \sum_{i=1}^n x_i^m \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 & \dots & \sum_{i=1}^n x_i^{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_i^m & \sum_{i=1}^n x_i^{m+1} & \dots & \sum_{i=1}^n x_i^{m+m} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{pmatrix}$$

$$= \begin{pmatrix} a_0 n + a_1 \sum_{i=1}^n x_i + \ldots + a_m \sum_{i=1}^n x_i^m \\ a_0 \sum_{i=1}^n x_i + a_1 \sum_{i=1}^n x_i^2 + \ldots + a_m \sum_{i=1}^n x_i^{m+1} \\ \vdots \\ a_0 \sum_{i=1}^n x_i^m + a_1 \sum_{i=1}^n x_i^{m+1} + \ldots + a_m \sum_{i=1}^n x_i^{m+m} \end{pmatrix} = \begin{pmatrix} \sum_{j=0}^m \left(a_j \sum_{i=1}^n x_i^h \right) \\ \sum_{j=0}^m \left(a_j \sum_{i=1}^n x_i^{j+1} \right) \\ \vdots \\ \sum_{j=0}^m \left(a_j \sum_{i=1}^n x_i^{j+m} \right) \end{pmatrix}.$$

This gives the left-hand side of each equation.

For the right-hand side, we have

$$\mathbf{A}^T\mathbf{y} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_1^m & x_2^m & \dots & x_n^m \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \\ \vdots \\ \sum_{i=1}^n x_i^m y_i \end{pmatrix}$$

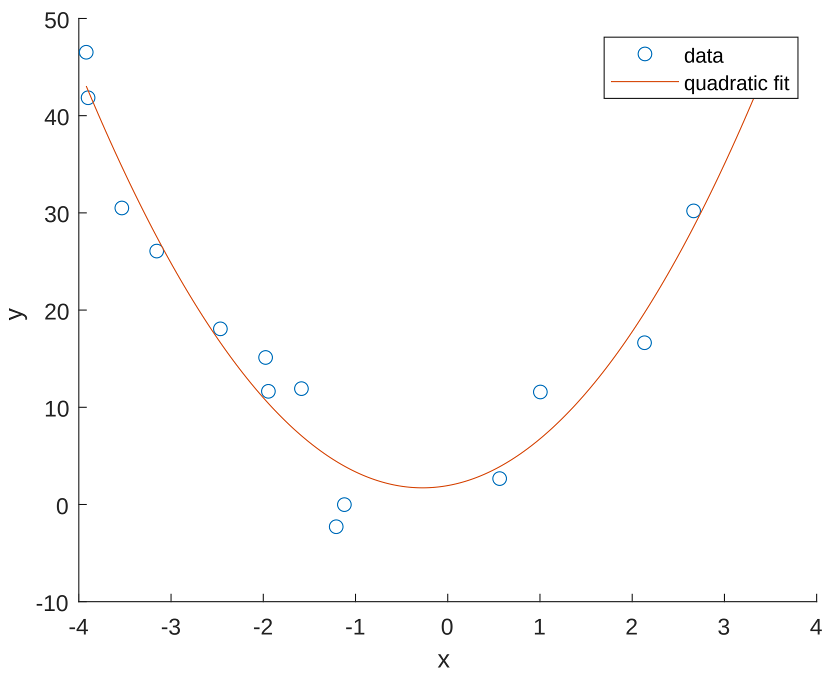
3. The following variables will be used for Questions 3–7.

```
dat = readmatrix('workshop4_data');
x = dat(:,1);
y = dat(:,2);
q = dat(:,3);
r = dat(:,4);
s = dat(:,5);
```

To fit the data to $y = a_0 + a_1 x + a_2 x^2$ and construct the plot we use the following:

```
A = [ones(size(x)) x x.^2];
a = inv(A'*A)*A'*y;
hold on;
scatter(x,y);
t = linspace(min(x), max(x), 300);
plot(t, a(1) + a(2)*t + a(3)*t.^2);
legend('data', 'quadratic fit');
xlabel('x');
ylabel('y');
hold off
```

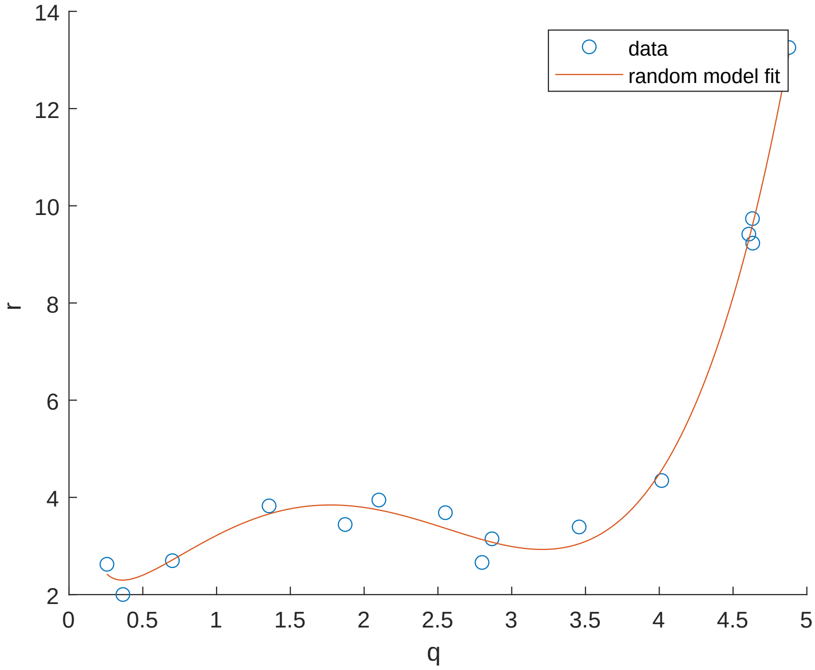
The following plot is obtained:



4. To fit the data to $r = a_0 \exp(q) + a_1 \sin(q) + a_2 \frac{1}{q}$ and construct the plot we use the following:

```
A = [exp(q) sin(q) 1./q];
a = inv(A'*A)*A'*r;
hold on;
scatter(q,r);
t = linspace(min(q), max(q), 300);
plot(t, a(1)*exp(t) + a(2)*sin(t) + a(3)./t);
legend('data', 'random model fit');
xlabel('q');
ylabel('r');
hold off
```

The plot is shown on the next page.

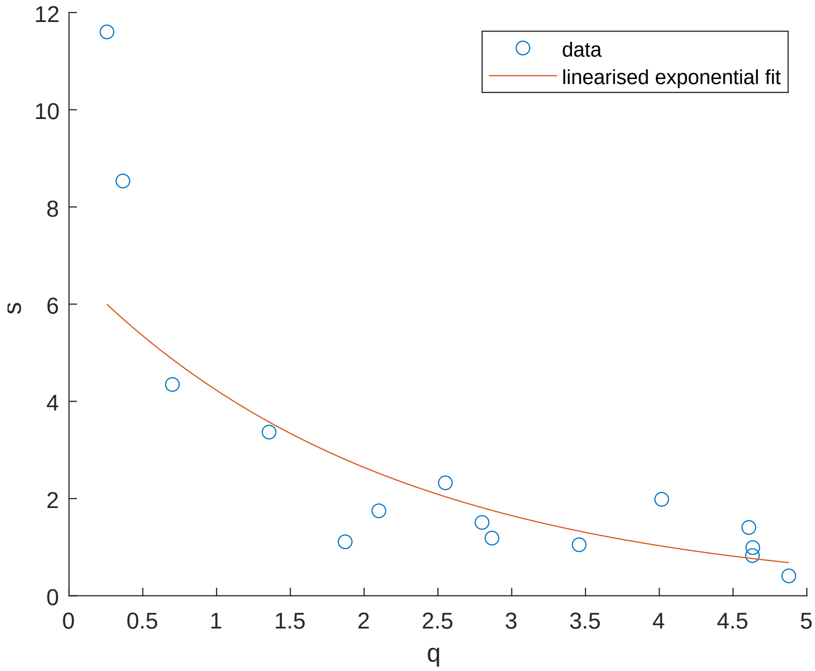


5. Applying log to both sides of $y = ae^bx$ results in $\log(y) = \log(a) + b\log(x)$, which gives $b = m$ and $\log(a) = c$.

6. Note that this results in the linearised coefficients, so to construct the function we must delinearise. To fit the linearised data $(q_i, \log(s_i))$ to a linear function $l(q) = mq + c$ and construct the plot, we use the following:

```
A = [ones(size(q)) q];
logged = inv(A'*A)*A'*log(s); % note that c = logged(1), m = logged(2)
a = exp(logged(1));
b = logged(2);
hold on;
scatter(q,s);
t = linspace(min(q), max(q), 300);
plot(t, a*exp(t*b));
legend('data', 'linearised exponential fit');
xlabel('q');
ylabel('s');
hold off
```

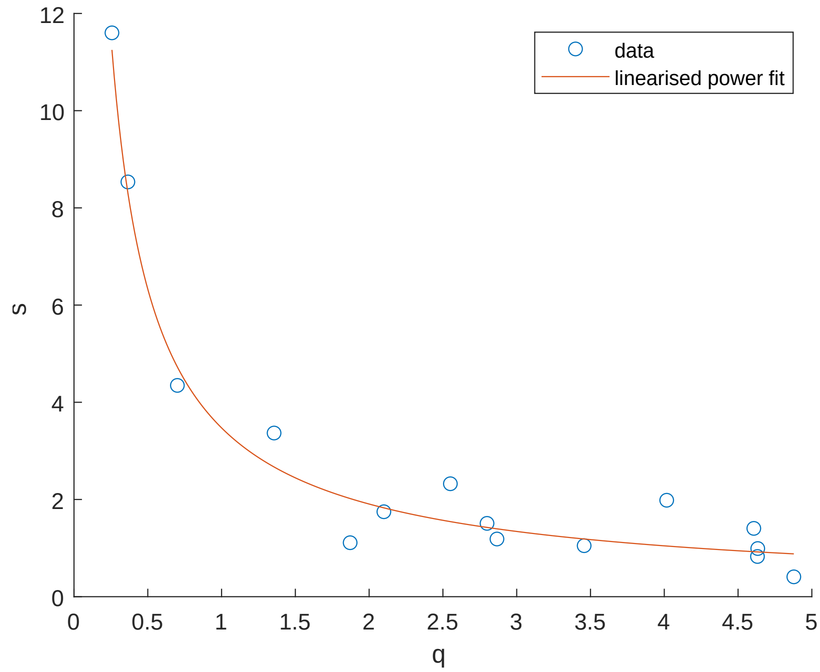
The following plot is obtained:



7. Applying log to both sides of $s = aq^b$ results in $\log(s) = \log(a) + b\log(q)$. So we will fit $(\log(q_i), \log(s_i))$ to a linear function $l(x) = mx + c$, with $x = \log(q)$, $b = m$ and $a = e^c$.

```
A = [ones(size(q)) log(q)];
logged = inv(A'*A)*A'*log(s); % note that c = logged(1), m = logged(2)
a = exp(logged(1));
b = logged(2);
hold on;
scatter(q,s);
t = linspace(min(q), max(q), 300);
plot(t, a*t.^b);
legend('data', 'linearised power fit');
xlabel('q');
ylabel('s');
hold off
```

The following plot is obtained:



8. (a) The objective function is

$$\begin{aligned} F_1(a,b) &= \sum_{i=1}^{7} (y_i - ae^{bx_i})^2 \\ &= (1.62 - ae^{0.28b})^2 + (1.22 - ae^{0.76b})^2 + \ldots + (0.22 - ae^{4.90b})^2 \end{aligned}$$

In the file `objective.m`, we define

```
function f = objective(vars)
x = [0.28 0.76 0.93 1.88 3.03 4.73 4.90];
y = [1.62 1.22 1.80 1.03 1.17 0.70 0.22];
a = vars(1);
b = vars(2);
f = sum((y - (a*exp(b*x))).^2);
end
```

Then `fminsearch(@objective, [0 0])` finds $a \approx 1.7883$, $b \approx -0.2344$.

(b) The objective function is

$$\begin{aligned} F_2(a,b) &= \sum_{i=1}^{7} (\log(y_i) - ax_i - b)^2 \\ &= (\log(1.62) - 0.28a - b)^2 + (\log(1.22) - 0.76a - b)^2 + \ldots + (\log(0.22) - 4.90a - b)^2 \end{aligned}$$

The minimiser is found using the methods from earlier:

```
x = [0.28 0.76 0.93 1.88 3.03 4.73 4.90]';
y = [1.62 1.22 1.80 1.03 1.17 0.70 0.22]';
A = [ones(size(x)) x];
logged = inv(A'*A)*A'*log(y);
a = exp(logged(1))
b = logged(2)
```

This gives $a \approx 1.9669$ and $b \approx -0.3120$.

(c) A self-contained snippet of code to do this is

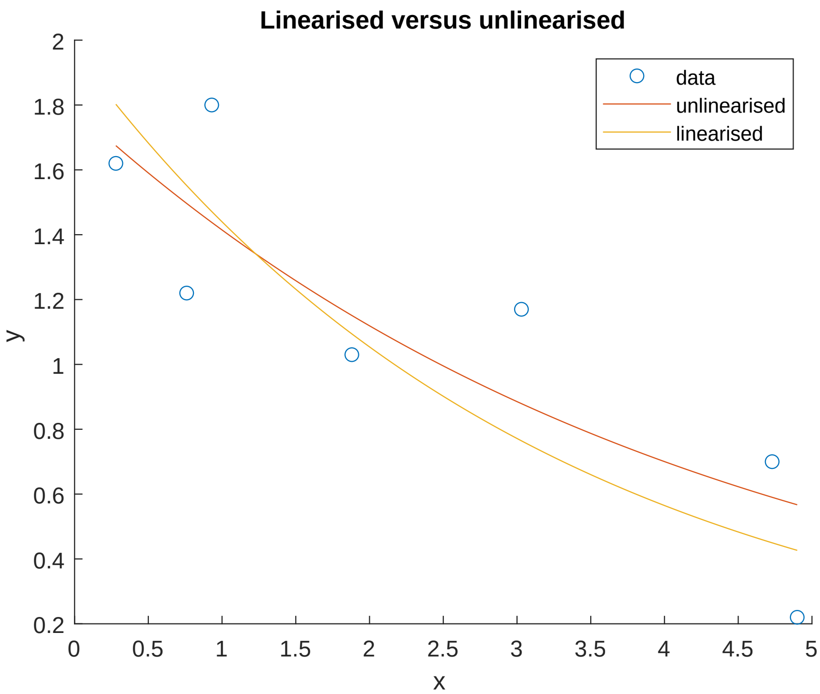
```
x = [0.28 0.76 0.93 1.88 3.03 4.73 4.90]';
y = [1.62 1.22 1.80 1.03 1.17 0.70 0.22]';

f = @(v) sum((y - (v(1)*exp(v(2)*x))).^2);
coeffs1 = fminsearch(f, [0 0]);

A = [ones(size(x)) x];
logged = inv(A'*A)*A'*log(y);
coeffs2 = [exp(logged(1)) logged(2)];

hold on
scatter(x,y)
t = linspace(min(x), max(x), 300);
plot(t, coeffs1(1)*exp(t*coeffs1(2)));
plot(t, coeffs2(1)*exp(t*coeffs2(2)));
legend('data', 'unlinearised', 'linearised');
xlabel('x');
ylabel('y');
title('Linearised versus unlinearised');
hold off
```

The following plot is obtained:



9. Placing the following code in `solveQuadratics.m` will do the job:

```
function roots = solveQuadratics(coeffs)
a = coeffs(:,1);
b = coeffs(:,2);
c = coeffs(:,3);

positions = (a < 0);
a(positions) = a(positions)*-1;
b(positions) = b(positions)*-1;
c(positions) = c(positions)*-1;

discriminant = b.^2 - 4.*a.*c;
roots = ((-b + [-1,1].*sqrt(discriminant))./(2*a));
end
```

Note that this also accounts for the ordering that was specified in Workshop 1. As this was not a requirement here, the following code is also suitable:

```
function [roots, discriminant] = solveQuadratics(coeffs)
a = coeffs(:,1);
b = coeffs(:,2);
c = coeffs(:,3);
discriminant = b.^2 - 4.*a.*c;
roots = ((-b + [-1,1].*sqrt(discriminant))./(2*a));
end
```

10. (a)
```
function M = swapRows(M, i, j)
M([i j],:) = M([j i], :);
end
```

(b)
```
function M = multiplyRow(M, i, c)
M(i,:) = c*M(i,:);
end
```

(c)
```
function M = addRow(M, i, j, c)
M(j,:) = M(j,:) + c.*M(i,:);
end
```

(d)
```
function column = rowLeader(M, i)
nonzero = find(M(i,:) ~= 0);
if isempty(nonzero)
    column = 0;
else
    column = nonzero(1);
end
```

(e) A useful trick for this one is that the `addRow` function can actually add multiple rows by using matrix operations.

```
function M = pivot(M, i, j)
entry = M(i,j);
if (entry ~= 0)
    M(i,:) = M(i,:)/entry;
    [numrows, ~] = size(M);
    column = -M(:,j);
    column(i) = 0;
    M = addRow(M, i, 1:numrows, column);
end
end
```

11. For $\mathbf{A} = \left(\begin{array}{cc|c} 3 & -1 & 4 \\ 3 & -5 & 2 \end{array}\right)$, do the following:

```
A = [3 -1 4; 3 -5 2];
A = pivot(A,1);
A = pivot(A,2);
```

For $\mathbf{A} = \left(\begin{array}{cccc|c} 1 & 2 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 2 & 1 & 0 & 0 & 0 \end{array}\right)$, do the following:

```
A = [1 2 3 1 0; 0 1 2 3 0; 2 1 0 0 0];
A = pivot(A, 1);
A = pivot(A, 2);
A = pivot(A, 3);
```

12. Theoretically, the following code should work:

```
function M = gauss(M)
[numrows, numcols] = size(M);
currentRow = 1;
for c = 1:numcols
    % Find a row with a row leader in column c.
    % If there are no row leaders in this column, ignore this column.
    % If there is a row leader in this column,
    %   swap that row with currentRow and pivot on currentRow.
    %   Then increment currentRow.
    for r = 1:numrows
        leader = rowLeader(M,r);
        if (leader == c)
            M = swapRows(M, currentRow, r);
            M = pivot(M, currentRow);
            currentRow = currentRow + 1;
            break;
        end
    end
end
end
```

However, in some cases, due to imprecision in floating point arithmetic, it is possible to get incorrect results. A more robust implementation might implement tolerances when checking if an entry is zero, as is done by the built in `rref` function using the `tol` argument.