Question 1 –

First we use polyfit function of MATLAB to calculate the coefficient G.

```
load('data1.mat');
P = polyfit(v,d,1);
```

Which results in:

P = 0.1556 -0.0976

So the coefficient G is $0.1556\ \mu m/v$.

To plot the fitted function, we evaluate the function for the required range $v \in [0,10]$ as follows,

```
v1 = linspace(0,100,1000);
d1 = polyval(P,v1);
```

and then plot the fitted function and the dataset (Fig. 1) as follows,

```
plot(v,d,'*');
hold on;
plot(v1,d1,'.');
xlabel('Voltage (volts)');
ylabel('Displacement (µm)');
```

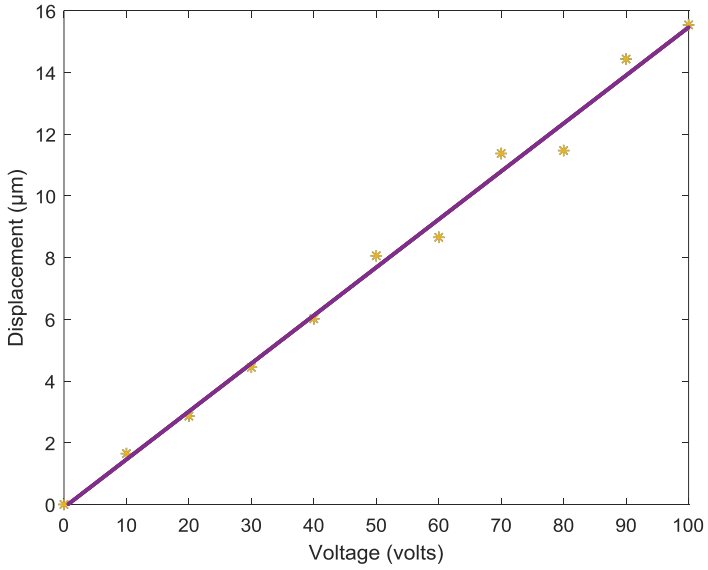

Figure 1 – Fitted line to data

We then calculate the displacement for v = 75 volts using polyval function.

```
d_75 = polyval(P,75);
```

Which results in:

d_75 = 11.5731

Question 2 –

a)

The results of surface fitting to the given data (x,y,u) is shown in Fig. 2.

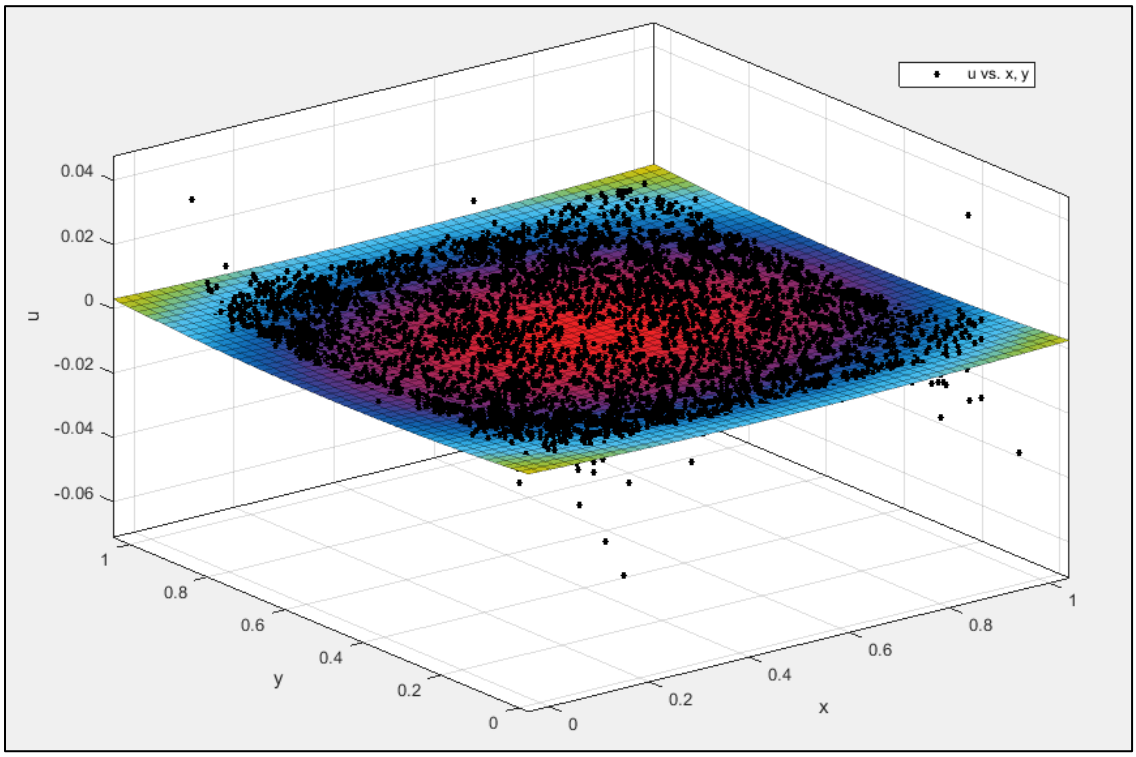

Figure 2 - Second order polynomial surface fits ($u = f(x, y)$)

A function is generated using the curve fitting toolbox of MATLAB and exported to MATLAB (attached to this document "createFit.m"). The generated code is presented below.

```
function [fitresult, gof] = createFit(x, y, u)

    %% Fit: 'untitled fit 1'.
    [xData, yData, zData] = prepareSurfaceData( x, y, u );

    % Set up fittype and options.
    ft = fittype( 'poly22' );

    % Fit model to data.
    [fitresult, gof] = fit( [xData, yData], zData, ft );

end
```

b)

A MATLAB script is written to plot the Quiver plot of the data. The Quiver plot of the data and the fitting function is given in Fig. 3.

```
% Load data
load('data2.mat');

% make grids
[gridx,gridy] = meshgrid(0:0.05:1,0:0.05:1);

% Call createFit1c and Calculate the functions
[ufit, gofu] = createFit(x, y, u);
[vfit, gofv] = createFit(x, y, v);

% Quiver plot
quiver(gridx, gridy, ufit(gridx, gridy), vfit(gridx, gridy));
axis equal
```

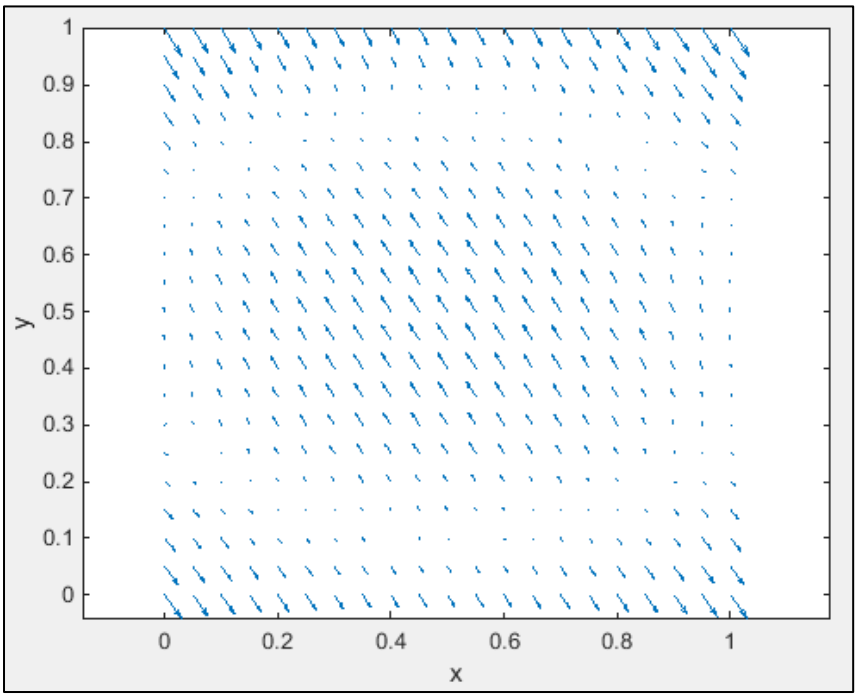

Figure 3 – Quiver plot for the given functions

Question 3-

a)

The function is nonlinear, however, this can be modified to a linear function as below.

$$y = k_1 x^{k_2}$$

$$\ln(y) = \ln\left(k_1 x^{k_2}\right) = \ln(k_1) + k_2 \ln(x)$$

We can rewrite it in the following linear format.

$$y_{new} = \theta_1 + \theta_2 x_{new}$$

Where:

$$y_{new} = \ln(y)$$

$$x_{new} = \ln(x)$$

$$\theta_1 = \ln(k_1)$$

$$\theta_2 = k_2$$

The closed form matrices will have the following form.

$$X = [1 \;\; \ln(x)]$$

$$Y = [lny]$$

$$\theta = [\theta_1;\ \theta_2]$$

And the solution, i.e. the coefficient matrix θ is computed by the following equation:

$$\theta = (X^T X)^{-1} * X^T Y$$

The obtained values for the coefficient matrix θ are:

$$\theta = [0.7260; 1.3215]$$

Accordingly, the values for k_1 and k_2 can be calculated.

$$k_1 = 2.0668; \quad k_2 = 1.3215$$

The following MATLAB code has been used in this section.

```
clear all; close all;

load data3.mat
```

```
plot(x,y,'.'); hold on;
plot(xv,yv,'.r')
legend('Training data', 'Validation data')


X = [ones(length(x),1) log(x)];
Y = log(y);

theta = inv(X'*X)*X'*Y;

k1 = exp(theta(1));
k2 = theta(2);
```

b)

The estimated function and scatter plots of the data are shown in Fig. 4.

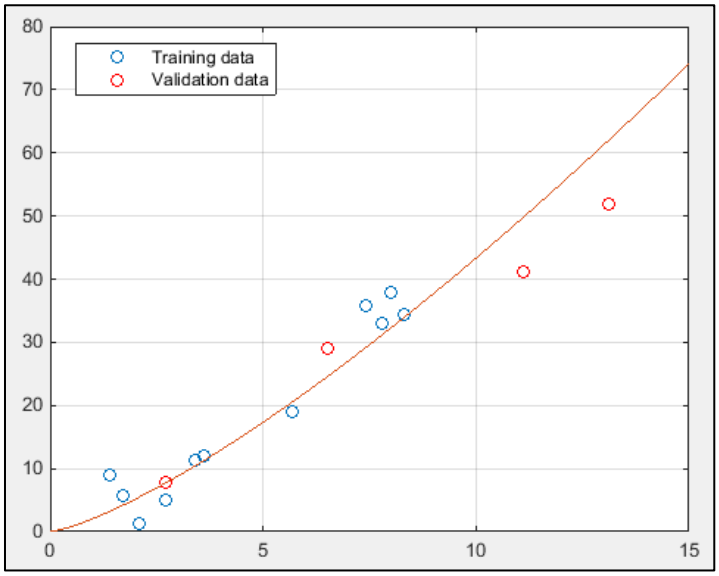

Figure 4 – Estimated function and data

The computed are as below.

SSE for training data	SSE for validation data	R^2
144	191	0.93

The following MATLAB code has been used in this part.

```
xaxis = linspace(0,15,100);
plot(xaxis, k1*xaxis.^k2);
```

```
grid on;

% Calculate SSE and R2

yest = k1*x.^k2;
SSE_train = sum((y-yest).^2);
SST_train = sum((y-mean(y)).^2);
R2 = 1-SSE_train/SST_train;

yestv = k1*xv.^k2;
SSE_validation = sum((yv-yestv).^2);
```

c)

The following MATLAB code has been used to estimate 1st, 2nd, 3rd, 4th order polynomials for the training data. To choose the best fit, we computed SSE for the validation data for each of the estimated polynomials. We chose the 1st order polynomial, since even though this model is quite simple, it captures the data quite well and has the lowest SSE for validation data.

Polynomial order	1	2	3	4
SSE for validation data (~)	145	1880	4737	437

```
SSE_valid = [];
P1 = polyfit(x,y,1);
yestv1 = polyval(P1,xv);
SSE_valid(1) = sum((yv-yestv1).^2);

P2 = polyfit(x,y,2);
yestv2 = polyval(P2,xv);
SSE_valid(2) = sum((yv-yestv2).^2);

P3 = polyfit(x,y,3);
yestv3 = polyval(P3,xv);
SSE_valid(3) = sum((yv-yestv3).^2);

P4 = polyfit(x,y,4);
yestv4 = polyval(P4,xv);
SSE_valid(4) = sum((yv-yestv4).^2);
```

Question 4 -

a)

The MATLAB function which outputs the sum-of-squared-error (SSE) for the input coefficient matrix k is as follows:

$$k = [k_1;\ k_2]$$

```
function c = computeCost(k)

    global x y

    k1 = k(1);
    k2 = k(2);

    c = sum((y - k1*(1- exp(k2*x))).^2);

end
```

b)

Using the MATLAB command fminsearch, the coefficients k_1 and k_2 were identified for which the SSE has the smallest value. The following MATLAB script has been used in this part:

```
% Load data
load data4.mat
global x y

k1 = 1;
k2 = 1;

k = [k1 k2];
k = fminsearch('computeCost', k);
```

The results are:

$k_1 = 1.4927$; $k_2 = -0.2021$.

c)

The visualization of the cost function in the neighborhood of the global minima is illustrated in Fig. 5. The following MATLAB script has been used in this part.

```
% Grid over which we will calculate J
k1_vals = linspace(1.5, 1.8, 100);
k2_vals = linspace(1.2, 1.4, 100);

% initialize J_vals to a matrix of 0's
J_vals = zeros(length(k1_vals), length(k2_vals));

% Fill out J_vals
for i = 1:length(k1_vals)
    for j = 1:length(k2_vals)
      t = [k1_vals(i); k2_vals(j)];
      J_vals(i,j) = computeCost(t);
    end
end


% Because of the way meshgrids work in the surf command, we need to
% transpose J_vals before calling surf, or else the axes will be flipped
J_vals = J_vals';
% Surface plot
figure;
surf(k1_vals, k2_vals, J_vals);
xlabel('k_1'); ylabel('k_2');
```

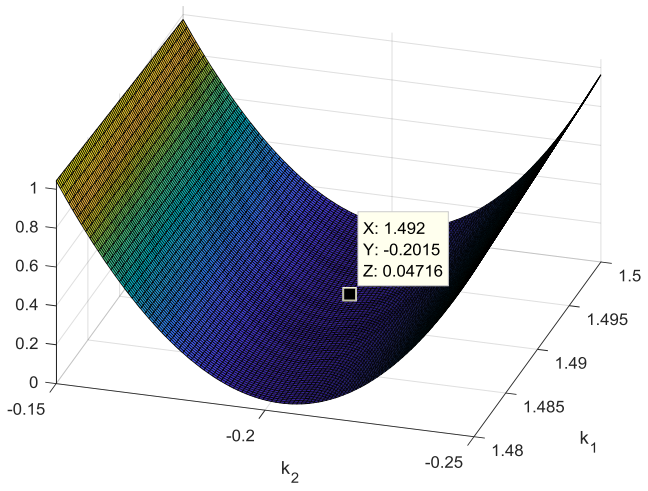

Figure 5 - 3D plot of the cost function in the neighborhood of the global minima