```
Question 1

(*Part 1*)

Ip = Range[0.5, 10, 0.5]; (*data values of current in mA*)
Vp = {0.2, 0.4, 0.6, 0.9, 1.1, 1.2, 1.4, 1.7, 1.8, 2.1, 2.2, 2.5,
   2.7, 2.9, 3.1, 3.4, 3.5, 3.9, 4.1, 4.3}; (*data values of Voltage in V*)
Vtheor =  (m * Ip ) + p; (*Theoretical relation for the Hall Voltage. This should ideally
 not have an intercept value but that is added to take in account systemic errors.*)
Resd = Vp - Vtheor;
res1 = (Resd^2) / 0.01;
sd = Total[res1]; (*Chi Square*)
Clear[c1];
c1 = ContourPlot[sd, {m, 0.42, 0.44}, {p, -0.1, 0.1}];
Show[c1] (*Plots χ^2 values against the parameter space*)


(*Part 2*)

Clear[vec1]
vec1 = {10, 0};

step = 0.000001; (*Value of β*)
Clear[m, p, i];
Clear[vecm, vecc, vecsd, vecn];
vecm = {}; (*list containing gradients to be calculated in the loop*)
vecc = {};(*list containing intercepts to be calculated in the loop*)
vecsd = {};(*list of χ^2 values *)
vecn = {}; (*list for iteration count*)

(*Finding the Gradient of the Chi-Square values analytically*)
Clear[del]
del = -Grad[sd, {m, p}]

(*Gradient Descent Method*)

For[i = 1, i < 10000, i++,
  d = -Grad[sd, {m, p}];
  m = vec1[[1]]; p = vec1[[2]];
 AppendTo[vecm, m];
 AppendTo[vecc, p];
 AppendTo[vecsd, sd];
 AppendTo[vecn, i];
 vec1 = vec1 + step * d;
 Clear[m, p]]
Print[vec1]
(*This yield an optimal value for the gradient and intercept as 0.42739,-0.043837*)

(*Part 3*)

data = Transpose@{vecm, vecc}
```

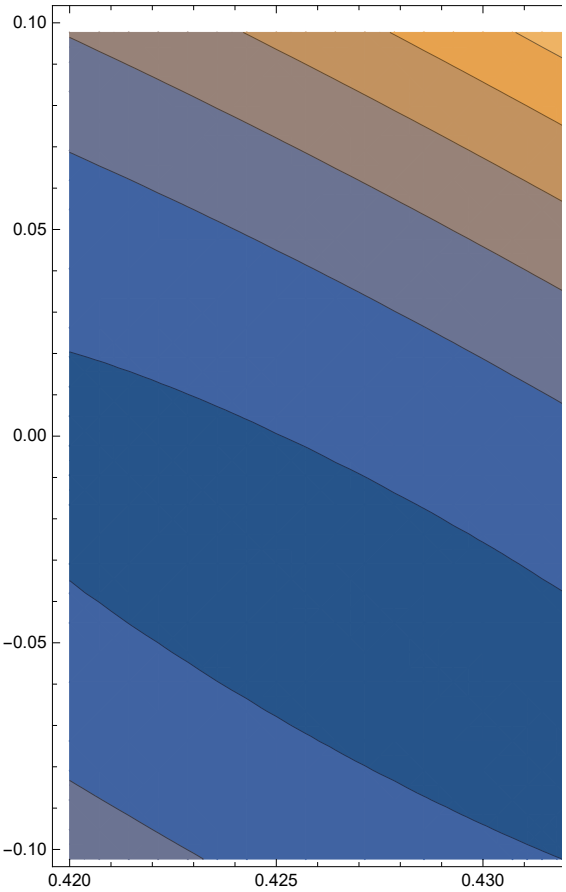
```
parameterspace1 = ListPlot[Thread[{vecm , vecc}]];
Show[c1, parameterspace1]
(*Plots the evolution of the parameters on the contour plot.*)

(*Part 4*)

data1 = Transpose@{vecn, vecsd}
ListPlot[data1] (*Plots values of χ^2  against the iteration count.*)
```

Question

Part



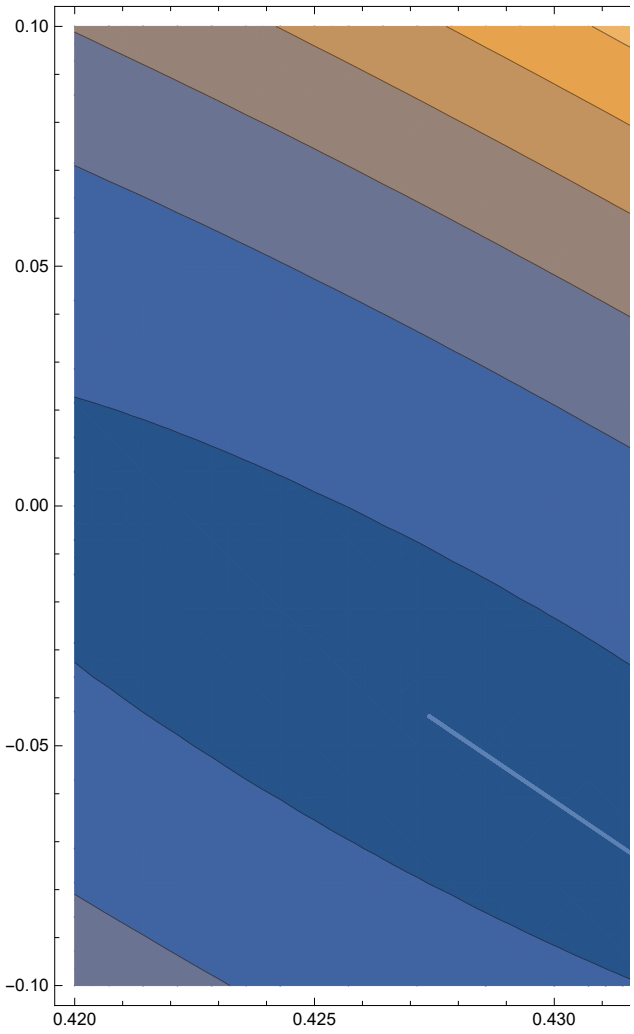

```
(*The plot above shows the Chi Square values plotted
 against the gradient on the x axis and the intercept on the y axis.*)
```

2 Part

```
{2000. (4.3 - 10. m - p) + 1900. (4.1 - 9.5 m - p) + 1800. (3.9 - 9. m - p) + 1700. (3.5 - 8.5 m - p) +
  1600. (3.4 - 8. m - p) + 1500. (3.1 - 7.5 m - p) + 1400. (2.9 - 7. m - p) + 1300. (2.7 - 6.5 m - p) +
  1200. (2.5 - 6. m - p) + 1100. (2.2 - 5.5 m - p) + 1000. (2.1 - 5. m - p) + 900. (1.8 - 4.5 m - p) +
  800. (1.7 - 4. m - p) + 700. (1.4 - 3.5 m - p) + 600. (1.2 - 3. m - p) + 500. (1.1 - 2.5 m - p) +
  400. (0.9 - 2. m - p) + 300. (0.6 - 1.5 m - p) + 200. (0.4 - 1. m - p) + 100. (0.2 - 0.5 m - p),
 200. (4.3 - 10. m - p) + 200. (4.1 - 9.5 m - p) + 200. (3.9 - 9. m - p) + 200. (3.5 - 8.5 m - p) +
  200. (3.4 - 8. m - p) + 200. (3.1 - 7.5 m - p) + 200. (2.9 - 7. m - p) + 200. (2.7 - 6.5 m - p) +
  200. (2.5 - 6. m - p) + 200. (2.2 - 5.5 m - p) + 200. (2.1 - 5. m - p) + 200. (1.8 - 4.5 m - p) +
  200. (1.7 - 4. m - p) + 200. (1.4 - 3.5 m - p) + 200. (1.2 - 3. m - p) + 200. (1.1 - 2.5 m - p) +
  200. (0.9 - 2. m - p) + 200. (0.6 - 1.5 m - p) + 200. (0.4 - 1. m - p) + 200. (0.2 - 0.5 m - p)}
```

```
{0.427391, -0.0438372}
```

```
{{10, 0}, {8.62541, -0.2012}, {7.4523, -0.372729}, {6.45113, -0.518936},
 {5.5967, -0.643534}, {4.8675, -0.749691}, {4.24517, -0.840109}, {3.71404, -0.917098},
 OutputSizeLimit`Skeleton[9983], {0.427391, -0.0438383}, {0.427391, -0.0438381},
 {0.427391, -0.043838}, {0.427391, -0.0438378}, {0.427391, -0.0438377},
 {0.427391, -0.0438376}, {0.427391, -0.0438374}, {0.427391, -0.0438373}}
```

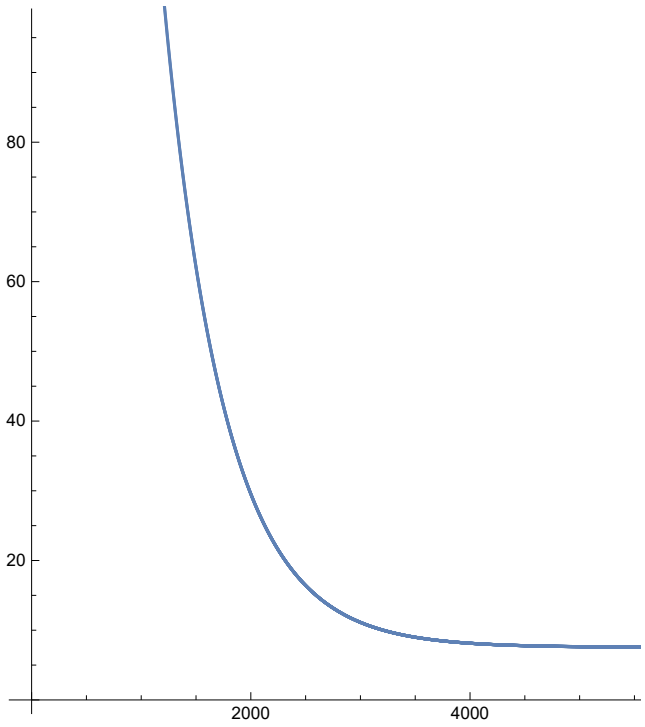


(*This plot shows the evolution of the contour plot
with the gradient and intercept. We can see that the evolution is linear.*)

$$\{\{1, 6.58362\times10^6\}, \{2, 4.79511\times10^6\}, \{3, 3.49252\times10^6\}, \{4, 2.54384\times10^6\},$$
$$\{5, 1.85291\times10^6\}, \{6, 1.34971\times10^6\}, \{7, 983\,222.\}, \{8, 716\,308.\},$$
$$\{9, 521\,913.\}, \{10, 380\,333.\}, \{11, 277\,220.\}, \{12, 202\,122.\}, \{13, 147\,428.\},$$

OutputSizeLimit`Skeleton[9974], {9988, 7.54738}, {9989, 7.54738},

{9990, 7.54738}, {9991, 7.54738}, {9992, 7.54738}, {9993, 7.54738}, {9994, 7.54738},
{9995, 7.54738}, {9996, 7.54738}, {9997, 7.54738}, {9998, 7.54738}, {9999, 7.54738}}




```
(*This plos the x axis and the iteration count on the y axis. This makes evident
 the fact that Chi Square decreases with the iteration count exponentially.*)
```

Question 1

Part 5

```
Ip = Range[0.5, 10, 0.5];
Vp = {0.2, 0.4, 0.6, 0.9, 1.1, 1.2, 1.4,
   1.7, 1.8, 2.1, 2.2, 2.5, 2.7, 2.9, 3.1, 3.4, 3.5, 3.9, 4.1, 4.3};
Vtheor = (m * Ip) + p;
Resd = Vp - Vtheor;
res1 = (Resd^2) / 0.01;
sd = Total[res1];

Clear[vec1, step, m, p, i]
vec1 = {10, 0};
step = 0.000005; (*Value of β*)


(*I have copied the above code from my part 1.
 For investigating how the step size changes the parameters,
I used the range of the step size between 0.000005 and 0.0000017 and took
 four vaues of the step size. If the range is changed significantly,
it will yield highly fluctuating parameter values. This shows that
 the paramter optimization is highly sensitive to the step size and
 that the step size is to be chosen within a reasonable range*)


For[j = 1, j < 5, j++;
 step = step + 0.000003;
 For[i = 1, i < 10000, i++,
   d = -Grad[sd, {m, p}];
   m = vec1[[1]]; p = vec1[[2]];
  vec1 = vec1 + step * d;
  Clear[m, p]];
  Print[vec1]]
```

2 Part

```
{}

{0.427368, -0.0436842}

{0.427368, -0.0436842}

{0.427368, -0.0436842}

{-2.52016921636234×10^1721, -3.71151850567703×10^1720}
```

Question 2

```
Ip = Range[0.5, 10, 0.5];
Vp = {0.2, 0.4, 0.6, 0.9, 1.1, 1.2, 1.4,
   1.7, 1.8, 2.1, 2.2, 2.5, 2.7, 2.9, 3.1, 3.4, 3.5, 3.9, 4.1, 4.3};
Vtheor = (m * Ip) + p;
Resd = Vp - Vtheor;
res1 = (Resd^2) / 0.01;

sd = Total[res1]; (*χ^2*)
Clear[vec1, i, HessianChi, A, d]
vec1 = {10, 0};
HessianChi = D[sd, {{m, p}, 2}] (*Defining Hessian*)

For[i = 1, i < 10, i++;
 A = Inverse[HessianChi];
 d = -Grad[sd, {m, p}];
 m = vec1[[1]];
 p = vec1[[2]];
 vec1 = vec1 + (A.d);
 Print[vec1];
 Clear[m, p]]

(*This yields the optimal value for the gradient and intercept as
 (0.427368) and (-0.04368). We can see from the output values that the
 gradient and intercept are not changing with the iteration number. This
 shows an exact quadratic dependence of Chi Square on the parameters. As such,
the number of iterations is inconsequential but carrying the
 process in the loops reveals the exact quadratic dependence.*)
```

2 Question

```
{{143500., 21000.}, {21000., 4000.}}

{0.427368, -0.0436842}

{0.427368, -0.0436842}

{0.427368, -0.0436842}

{0.427368, -0.0436842}

{0.427368, -0.0436842}

{0.427368, -0.0436842}

{0.427368, -0.0436842}

{0.427368, -0.0436842}

{0.427368, -0.0436842}
```

Alternate way using MATLAB

```
%declaring function

function y = f(c,x)
y = c(1)*x+c(2);
end


%----------------Question
1-----------------------------------------------------------



close all;
clc;

%Declaring arrays

I=[0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5,7.0,7.5,8.0,8.5,9.0,9.5,10.0
];
V=[0.2,0.4,0.6,0.9,1.1,1.2,1.4,1.7,1.8,2.1,2.2,2.5,2.7,2.9,3.1,3.4,3.5,3.9,4.1,4.3]
;

%Prediciting initial values

m=0.5;
c=5;

%function written in expanded form

X = @(m,c)
(sum(V.^2)-2.*m.*dot(V,I)-2.*c*sum(V)+m.^2*sum(I.^2)+2.*m.*c.*sum(I)+20.*c.^2)./(0.
1^2);

%creating grid for contour plot

[M,C] = meshgrid(-1.5:0.1:1.5,-8:0.1:8);

%Plotting the contour

z = X(M,C);
contour(M,C,z,50);
title('Contour plot of Chi-squared function linear fitting by gradiet descent
method')
```

```
xlabel('Gradient');
ylabel('Y-intercept');
hold on;

%Defining stepsize

s = 0.00001;

%initializing gradient vector so that it enters the loop for sure

v=[1000000,1000000];

%defining tolerence level

tol=1;

%defining array to save X values

val = [];
j = 1; %counter for saving X values

while norm(v)> tol

        i = 1; %counter for sum in function
        %initializing partial derivatives
        pm=0;
        pc=0;
        while (i<21)       %function has sums for each value, so I have done it
explicitly

            pm = pm  -2*I(i)*(V(i)-((m).*I(i)+c))/(0.1^2);       %partial
derivative wrt m;
            pc = pc  -2     *(V(i)-((m).*I(i)+c))/(0.1^2);       %partial
derivative wrt c;
            i  = i + 1;
        end

        v = [pm , pc];

        %updating the values according to step size

        m = m - s*pm;
        c = c - s*pc;

        %to plot at every iteration
        val(j)= X(m,c);         %store value in an array
        plot (m,c,'ok');        %show point on contour plot
        pause(1.5/(j));         %pause for animation (pause gets lesser as
```

```
iteration increases
        hold on;
        j = j + 1;
end

[m,c]
hold on;

%new figure for X vs iteration count
figure;

%skipping initial 30 values because the
%difference between them is too large making the plot ugly
k=30;

%I have split it in two so that the decay is visible for large and small
%chi-squared
while k<400
        plot(k,val(k),'ok');
        hold on;
        k=k+1;
end
xlabel('Iteration Number')
ylabel('Value of Chi-squared function')
title('Value of Chi-squared function vs Iteration number')

figure;
while k<j
        plot(k,val(k),'ok');
        hold on;
        k=k+1;
end

xlabel('Iteration Number')
ylabel('Chi-squared')
title('Value of Chi-squared function vs Iteration number')

%regular curve fitting for I vs V
figure;
plot (I,V,'ok');
hold on;

constant = lsqcurvefit(@f,[0;0],I,V);
y = constant(1)*I + constant(2);
plot (I,y);
xlabel('Current');
ylabel('Voltage');
title('Hall Voltage vs Current');
```

```
-------------------------------------Question
2-------------------------------------------------



close all;
clc;

%Declaring arrays

I=[0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5,7.0,7.5,8.0,8.5,9.0,9.5,10.0
];
V=[0.2,0.4,0.6,0.9,1.1,1.2,1.4,1.7,1.8,2.1,2.2,2.5,2.7,2.9,3.1,3.4,3.5,3.9,4.1,4.3]
;

%creating the hessian

hess = [;];

p2m  = 0; % second partial derivative of m
p2c  = 0; %secong partial derivative of c
pmpc = 0; %partial m and partial c

%numerically calulating the partial derivatives
i=1;
while (i<20)
    p2m  = p2m   + (2*sum(I(i).^3)/(0.1)^2);
    p2c  = p2c   + 20/(0.1^2);
    pmpc = pmpc  + 2*(sum(I(i))/(0.1^2));
    i = i+1;
end

%filling the hessian matrix

hess(1,1) = p2m;
hess(2,2) = p2c;
hess(1,2) = pmpc;
hess(2,1) = pmpc;

%vector that will update values of m and c
update = [];

%predicting initial values
    m=-1.25;
    c=2;
```

```
% chi squared function written in expanded form

X = @(m,c)
(sum(V.^2)-2.*m.*dot(V,I)-2.*c*sum(V)+m.^2*sum(I.^2)+2.*m.*c.*sum(I)+20.*c.^2)./(0.
1^2);

%creating grid for contour plot

[M,C] = meshgrid(-1.5:0.1:1.5,-8:0.1:8);

%Plotting the contour

z = X(M,C);
contour(M,C,z,50);
title('Contour plot of Chi-squared function for linear fitting by Newton`s method')
xlabel('Gradient');
ylabel('Y-intercept');
hold on;


%initializing gradient vector so that it enters the loop for sure

v=[1000000,1000000];

%defining tolerence level

tol=1;

%defining array to save X values

val = [];
j = 1; %counter for saving X values

while norm(v)> tol

        i = 1; %counter for sum in function

        %initializing partial derivatives
        pm=0;
        pc=0;

        while (i<21)        %function has sums for each value, so I have done it
explicitly

            pm = pm  -2*I(i)*(V(i)-((m).*I(i)+c))/(0.1^2);       %partial
derivative wrt m;
            pc = pc  -2     *(V(i)-((m).*I(i)+c))/(0.1^2);       %partial
derivative wrt c;
```

```
            i = i + 1;
        end

        v = [pm , pc];

        %updating the values according to newton`s method
        update = inv(hess)*(v');

        m = m - update(1);
        c = c - update(2);

        %to plot at every iteration
        val(j)= X(m,c);         %store value in an array
        plot (m,c,'ok');        %show point on contour plot
        pause(1.5/(j));         %pause for animation (pause gets lesser as
iteration increases)
        hold on;
        j = j + 1;
end

[m,c]

%new figure for X vs iteration count
figure;

%skipping initial 30 values because the
%difference between them is too large making the plot ugly
k=30;

%I have split it in two so that the decay is visible for large and small
%chi-squared
while k<300
        plot(k,val(k),'ok');
        hold on;
        k=k+1;
end

xlabel('Iteration Number')
ylabel('Value of Chi-squared function')
title('Value of Chi-squared function vs iteration number')
figure;

while k<j
        plot(k,val(k),'ok');
        hold on;
        k=k+1;
end
```

```
xlabel('Iteration Number')
ylabel('Chi-squared')
title('Value of Chi-squared function vs iteration number')

%regular curve fitting of I vs V
figure;
plot (I,V,'ok');
hold on;

constant = lsqcurvefit(@f,[0;0],I,V);
y = constant(1)*I + constant(2);
plot (I,y);
xlabel('Current')
ylabel('Voltage')
title('Hall Voltage vs Current')
```