ISMET OKAN CELIK

1. Solve the following using steepest descent algorithm. Start with $x0$=[1 1]T and use stopping threshold ∈=10−6.

(a) Verify that the final solution satisfies the second order necessary conditions for a minimum.

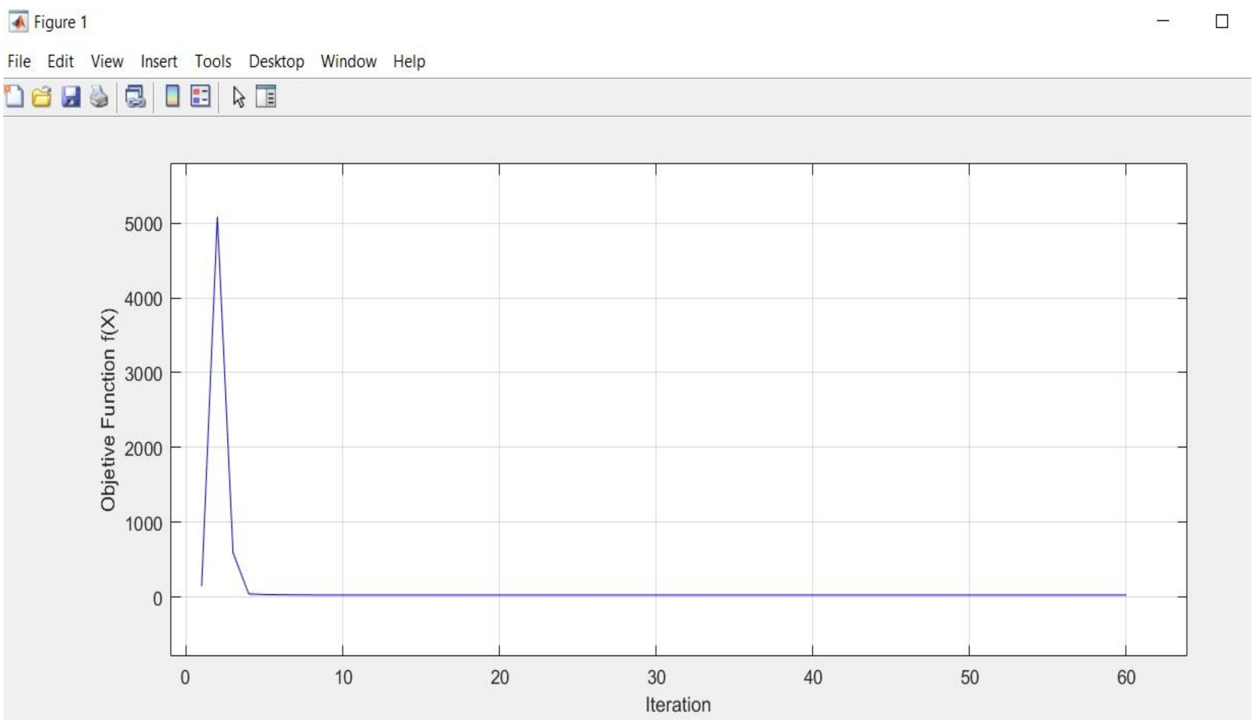
```
function [fx] = objectiveFunc(x1,x2,x3);
tic
format short;
syms x1 x2 x3;
x=[1;1;1];
e=10^(-6); %Treshold
fx=((x1+5)^2)+((x2+8)^2)+((x3+7)^2)+2*(x1^2)*(x2^2)+4*(x1^2)*(x3^2);
%Objective Function

gx=[diff(fx,x1);diff(fx,x2);diff(fx,x3)]; % Gradient Function

hx1=[diff(gx,x1)];
hx2=[diff(gx,x2)];
hx3=[diff(gx,x3)];
h=[hx1(1) hx2(1) hx3(1);hx1(2) hx2(2) hx3(2);hx1(3) hx2(3) hx3(3)]; %Hessien
Function

% alfa=0.1;
true=1;
iteration=0;
value=[0 0];
while(true)
    iteration=iteration+1;
    gx_val=subs(gx,x1,x(1));
    gx_val=subs(gx_val,x2,x(2));
    gx_val=subs(gx_val,x3,x(3));

    h_val=subs(h,x1,x(1));
    h_val=subs(h_val,x2,x(2));
    h_val=subs(h_val,x3,x(3));

    alfa=vpa((gx_val'*gx_val)/(gx_val'*h_val*gx_val)); % Calculation of Alfa
Iteratively

    second_order_cond=vpa((-gx_val')*h_val*(-gx_val)); %Calculation of Second
Order Nec. Condition

    eucledian=norm(alfa*gx_val);
    if(e<=eucledian)
        xk1=x-alfa*gx_val;
        fk1=subs(fx,x1,xk1(1));
        fk1=subs(fk1,x2,xk1(2));
        fk1=subs(fk1,x3,xk1(3));

        x=double(xk1);
        T=table(iteration, double(alfa), x(1), x(2), x(3), double(fk1),
double(second_order_cond));
```

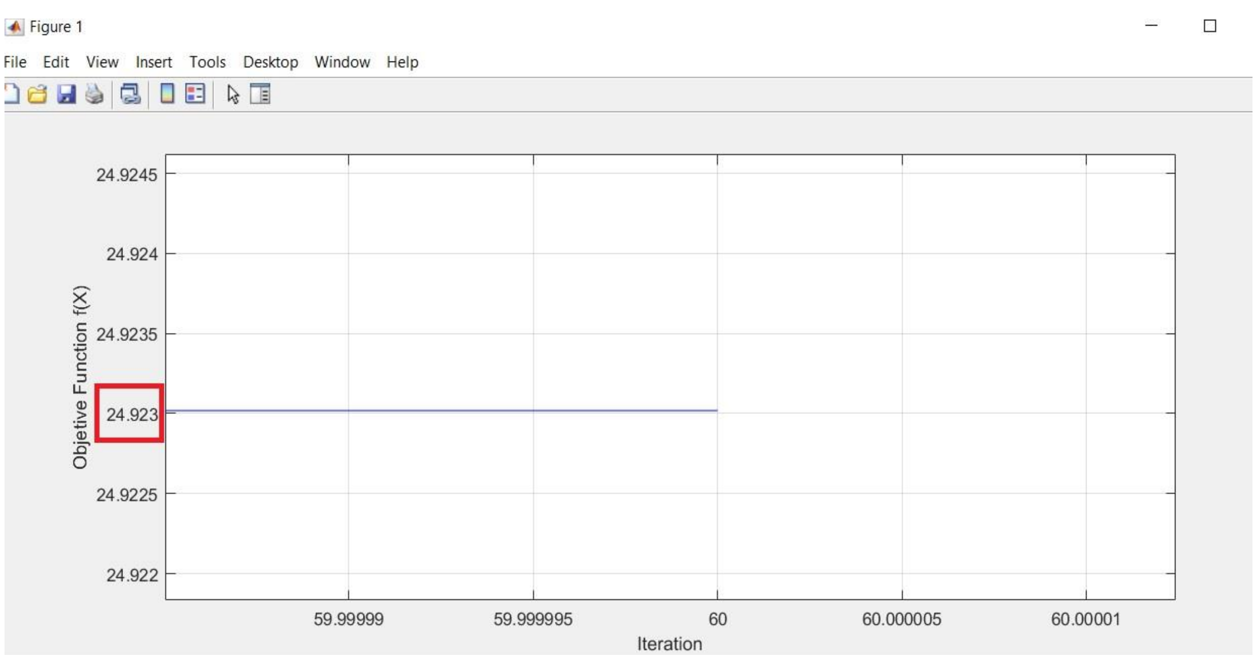
```
        T.Properties.VariableNames =
{'iteration','alfa','x1','x2','x3','obj_func','second_order'}

        plot_matrix= [double(iteration) double(fk1)];
        Generaltable(iteration,:)=plot_matrix;

    else
        disp('Achieved The Optimum Solution')
        true=0;
    end

toc
end
T = array2table(Generaltable,...
    'VariableNames',{'Iteration','Objective Func'})
```

Output Of The Algorithm

Iteration	Objective Func
1	143.24
2	5087.4
3	594.2
4	41.14
5	32.72
6	29.925
7	27.424
8	26.503
9	25.82
10	25.497
11	25.271
12	25.149
13	25.065
14	25.015
15	24.982
16	24.962
17	24.948
18	24.94
19	24.934
20	24.93
21	24.928
22	24.926
23	24.925
24	24.924
25	24.924
:	:
59	24.923
60	24.923

(b) Plot the value of the objective function with respect to the number of iterations and





c)Comment about convergence speed:

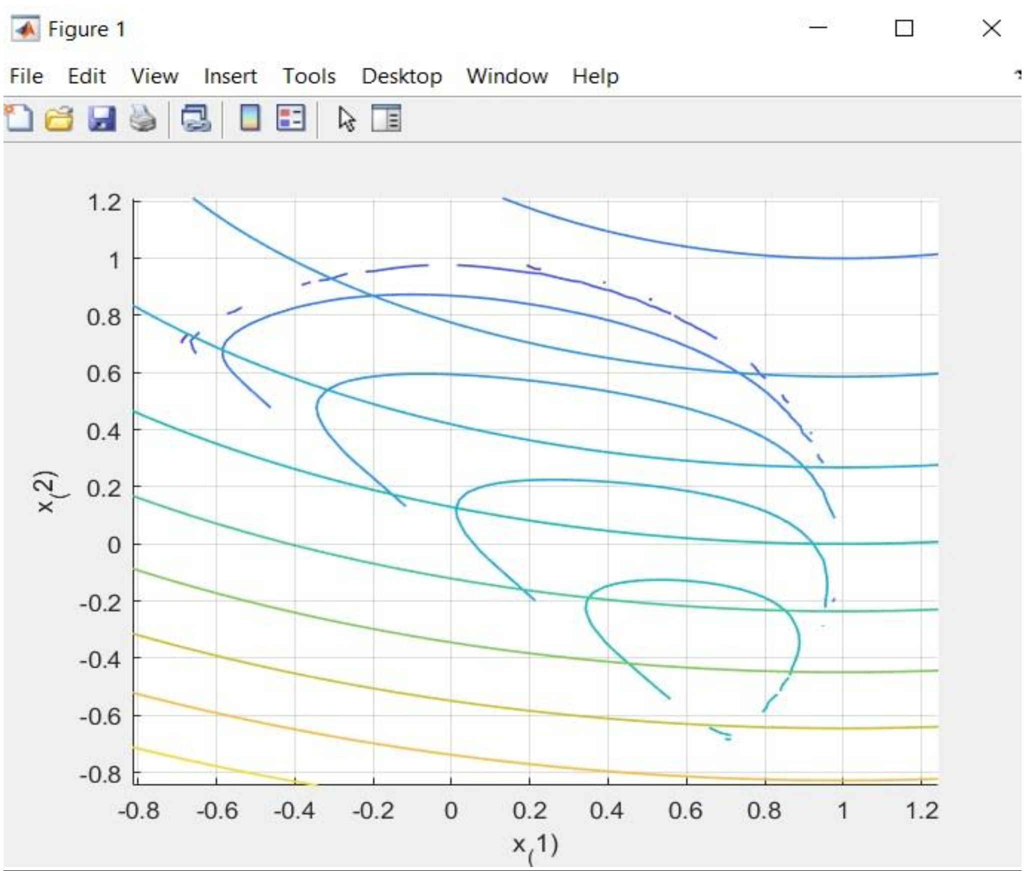
iteration	alfa	x1	x2	x3	obj_func	second_order
60	0.16821	-0.015409	-7.9962	-6.9933	24.923	1.7284e-08

```
Elapsed time is 2.310382 seconds.
Achieved The Optimum Solution
Elapsed time is 2.331585 seconds.
```

As we can see on the output, completing 60 iteration took almost 2.33 seconds. Also, our model almost reached the optimum solution at 25th iteration after the 25th iteration try to reach minimum. We can say that our algorithm converge fast to reach optimum solution.

2.

(a) Plot the contour of f(x) and the feasible set on one single figure, i.e., overlay the feasible set on the contour plot of f(x);



Feasible Set Source Code:

```
function [fx] = objectiveFunc(x1,x2);
format short;
syms x1 x2;
fx=(x1-1)^2+2*(x2-2)^2
```

```
Hx1=1-x1^2-x2^2;
Hx2=x1+x2;                  %constraints

c=1;

Barrier_func=vpa(fx-c*(-log(Hx1)-log(Hx2)));

[x1,x2]=meshgrid(0:0.1:10,0:0.1:10);
figure(1)
hold on
fcontour(Barrier_func,'LineWidth',1)
hold on
fcontour(fx,'LineWidth',1)
hold on
xlabel('x_(1)')
ylabel('x_(2)')
zlabel('fx')
hold on
hold on
xlim([-0.7 1])
ylim([-0.7 1])
grid on
hold on
```

(b) Find a solution to the problem using the natural logarithmic barrier function, i.e., the barrier function is -log(h1(xx)) - log(h2(xx)). Use initialization vector [0.5 0.5]T and the initial penalty parameter equal to 1 and reduce it by ½ in each iteration. Use a stopping threshold of 0.002;

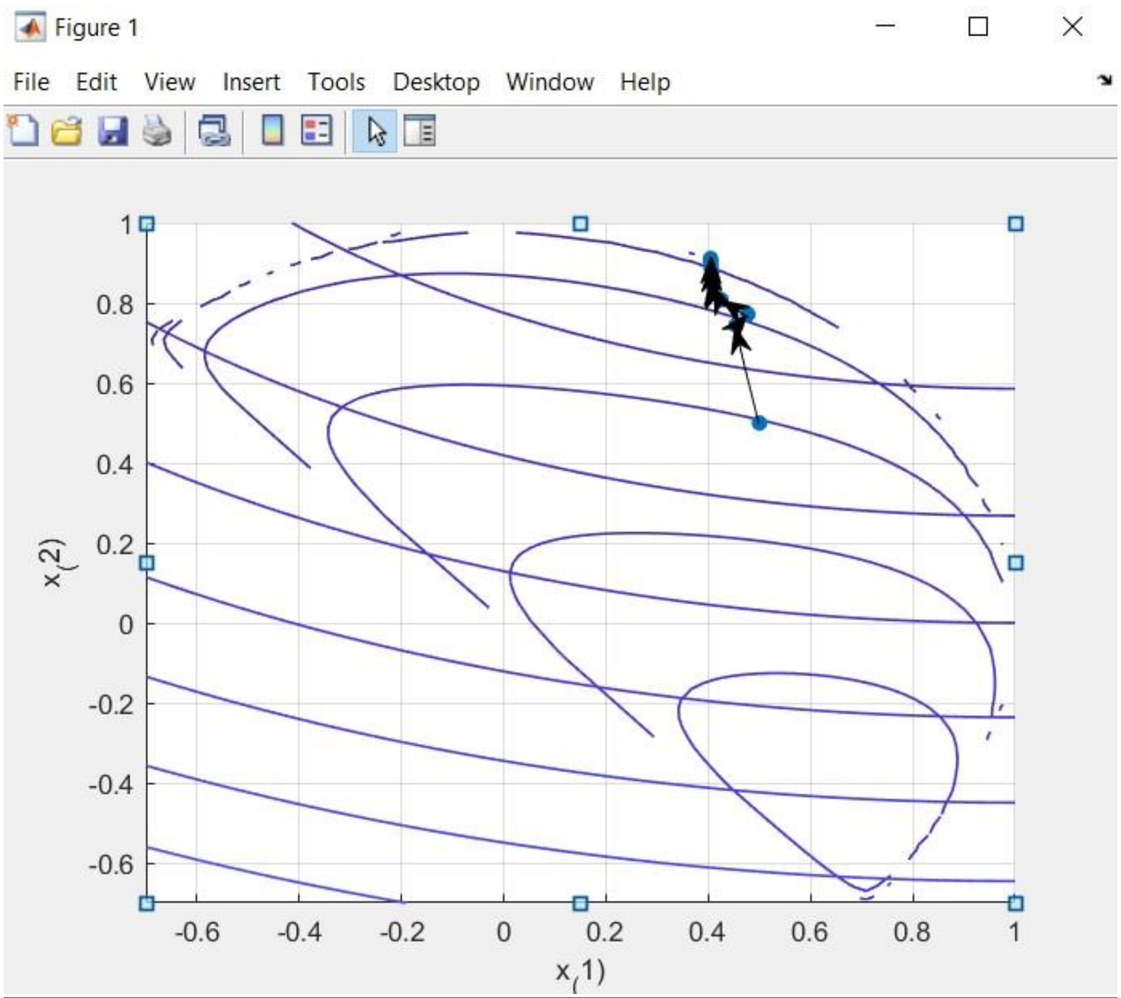
```
function [AugmentedFunc] = BarrierFunc(x1,x2,c);
format short;
syms x1 x2 c;
fx=(x1-1)^2+2*(x2-2)^2;

Hx1=1-x1^2-x2^2;
Hx2=x1+x2;                  %constraints

x=[0.5 0.5];
e=0.002;
c=1;


true=1;
iteration=0;
    while(true)
        iteration=iteration+1;

        Barrier_func=(fx+c*(-log(Hx1)-log(Hx2))^2);
        Delta_Barrier=[diff(Barrier_func,x1);diff(Barrier_func,x2)];

        HessienBarrier=[diff(Delta_Barrier(1),x1) diff(Delta_Barrier(1),x2);
            diff(Delta_Barrier(2),x1) diff(Delta_Barrier(2),x2)];
```

```
        Delta_Barrier_val=subs(Delta_Barrier,x1,x(1));
        Delta_Barrier_val=subs(Delta_Barrier_val,x2,x(2));

        Hessien_val=subs(HessienBarrier,x1,x(1));
        Hessien_val=subs(Hessien_val,x2,x(2));


alpha=vpa((Delta_Barrier_val'*Delta_Barrier_val)/(Delta_Barrier_val'*Hessien_
val*Delta_Barrier_val)); % Calculation of Alpha Iteratively

        eucledian=norm(alpha*Delta_Barrier_val);
        if(e<=eucledian);
            fx_val=subs(fx,x1,x(1));
            fx_val=subs(fx_val,x2,x(2));

            stepsize1=alpha*Delta_Barrier_val(1);
            stepsize2=alpha*Delta_Barrier_val(2);

            x1_new=vpa(x(1)-stepsize1);
            x2_new=vpa(x(2)-stepsize2);
            x=[x1_new x2_new];

            c=c*0.5;

            result_matrix= [double(iteration) double(x(1)) double(x(2))
double(fx_val)];
            ResultTable(iteration,:)=result_matrix;
        else
            true=0;
        end
    end
    GeneralTable = array2table(ResultTable,...
        'VariableNames',{'Iteration','x1','x2','f(x)'})
    writematrix(ResultTable,'IterationData.xlsx');
end
```

Output of the source-code:

Iteration	x1	x2	f(x)
1	0.47725	0.77175	4.75
2	0.45454	0.74455	3.2904
3	0.42506	0.80825	3.4498
4	0.40981	0.84183	3.1711
5	0.40695	0.86712	3.031
6	0.40593	0.88568	2.9185
7	0.40516	0.89694	2.8364
8	0.40486	0.90442	2.7873
9	0.40459	0.90863	2.7548
10	0.4045	0.9113	2.7367

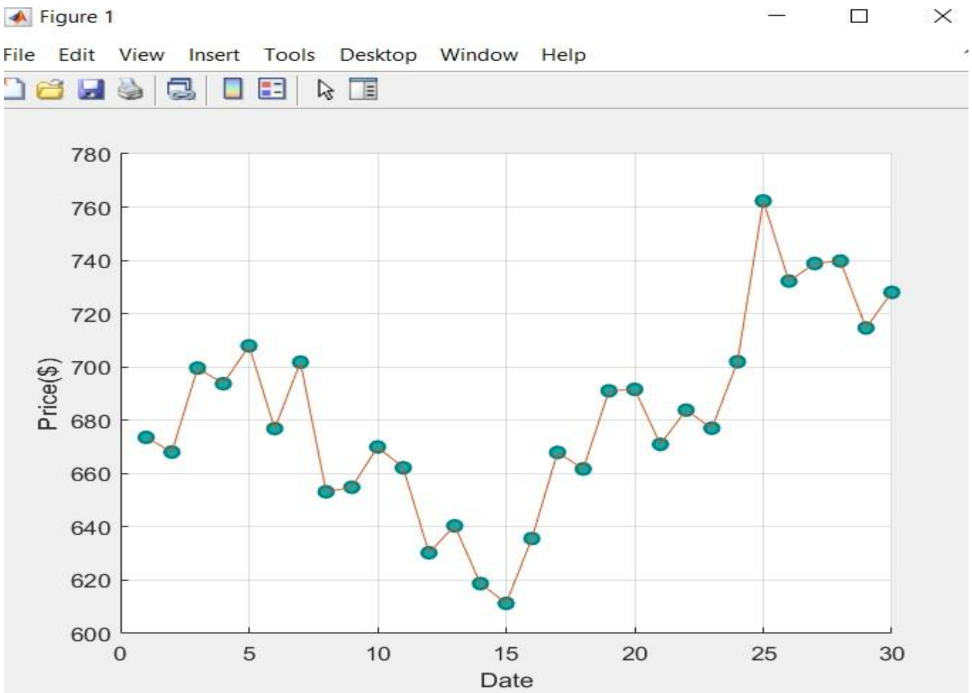
(c) In a 2-D figure, plot the trajectory (i.e., the values connected by lines with arrows) of the computed solution vector as the number of iteration progresses.



2-D Plotting the Trajectory of Computed Solution Vectors

Tesla Stock

a)Plot the data (Date vs.Stock)



Source Code for Plotting Raw Data

```
function Raw_Plot=Raw_Graph(Date,StockData);
StockData=xlsread('Tesla_Stock_Date.xlsx');
Date=[1:1:30];
figure(1)
Data_dist=scatter(Date,StockData,'MarkerEdgeColor',[0 .5 .5],...
              'MarkerFaceColor',[0 .7 .7],...
              'LineWidth',1.5);
hold on
plot(Date,StockData)
hold on
xlabel('Date')
ylabel('Price($)')
grid on
hold off
end
```

b)Implement the stochastic gradient descent algorithm to fit a linear regression model for this data set.

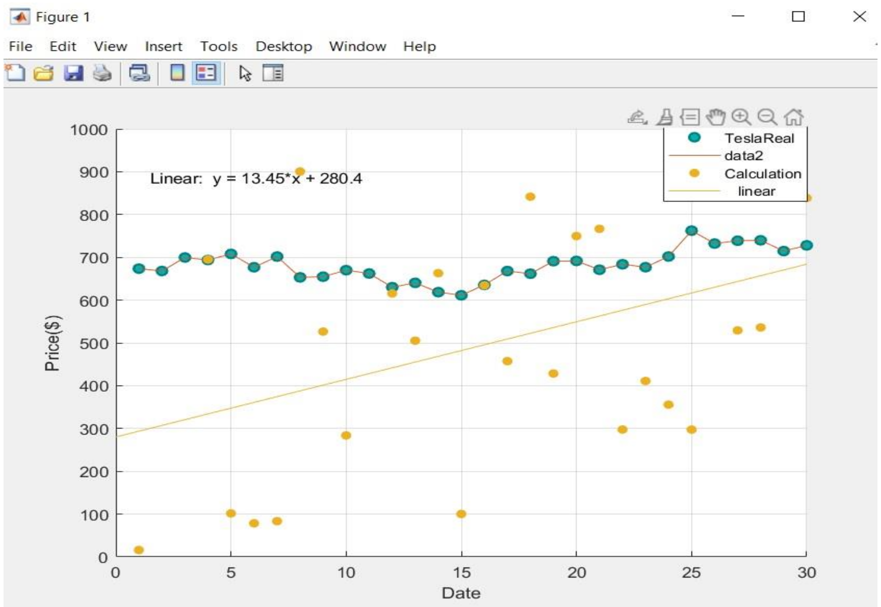
```
function [Q]=ObjectiveFunc(w1,w2,x,iteration);
syms x w1 w2 y_stock;
y_data=xlsread('Tesla_Stock_Date.xlsx');
format longG;

w1=0;
w2=1;
alpha=0.01;
```

```
iteration=30;
i=0;

    while(i<=iteration);
        i=i+1
        k=randi([1 30],1,1);
        x=[1:1:30]';
        y_stock=y_data(k);
        x_date=x(k);
        n=size(y_data);

        y_predict=(w1+w2*x_date);
        Q=vpa((y_stock-y_predict).^2)*(1/(2*i));

        DeltaQ_1=sum((2*w1 - 2*y_stock + 2*w2*x_date)/(2*i));
        DeltaQ_2=sum((x_date*(w1 - y_stock + w2*x_date))/i);

        stepsize_w1=vpa(alpha*DeltaQ_1);
        stepsize_w2=vpa(alpha*DeltaQ_2);

        w1=(w1-stepsize_w1);
        w2=(w2-stepsize_w2);

        Matrix=[i y_predict];
        GeneralTable(i,:)=Matrix;
        k=randi([1 30],1,1);
    end

    ResultTable= array2table(GeneralTable,...
        'VariableNames',{'Iteration','y_stock'})
    writetable(ResultTable,'PredictData.xlsx');
end
```

Output of the Stochastic Gradient Descent Algorithm

Iteration	Stock_Prediction
1	6
2	89.2144
3	1138.593172
4	20.259853
5	229.7089928475
6	171.000702160085
7	583.717965970896
8	180.547308802551
9	197.687571266468
10	452.013752192481
11	55.2262484877639
12	655.615969185525
13	1154.27510869166
14	719.262693452958
15	572.42340262839
16	624.868568958575
17	1045.10579288954
18	214.651475110238
19	443.041617145911
20	493.244047745466
21	276.467076794604
22	459.670411208891
23	898.154868184632
24	420.481809775848
25	817.596852542129
26	989.729448237548
27	747.702560918575
28	798.293430412834
29	693.617649962881
30	484.533039696311
31	225.300950457182

c)Visual Comparison of the Linear Regresion Model and Data Set.



Source Code of The Plot:

```
StockData=xlsread('Tesla_Stock_Date.xlsx');
PredictData=xlsread('PredictData.xlsx');
Date=[1:1:30];
figure(1)
Data_dist=scatter(Date,StockData,'MarkerEdgeColor',[0 .5 .5],...
              'MarkerFaceColor',[0 .7 .7],...
              'LineWidth',1.5);
hold on
plot(Date,StockData)
hold on
y_predict=[16;2609.81540000000;-2450.28906100000;694.998077126667;
                        101.614378566400;78.4954731922871;83.4085109156847;
                        900.584173060655;526.465338209734;283.816473792163;
                        1125.82937100132;615.645694488694;505.210078461766;
                        663.049042961928;100.198199928204;634.609275574038;
                        457.416467470634;841.767559683733;428.476885893180;
                        749.732352970513;766.606814828673;297.650649772695;
                        410.994850439726;355.844189214049;297.472418477945;
                        971.540350253300;529.272832983256;536.137973629843;
                       1073.19774147054;838.634038491472;320.119544281554];
iteration=[1:1:31];
scatter(iteration,y_predict,'filled')
hold on
setLine
hold on
xlabel('Date')
ylabel('Price($)')
hold on
xlim([0 30])
ylim([0 1000])
grid on
hold off
```

d)

Gradient Descent:

Pros: Gradient descent always guarantee to reached to optimum solution.

Cons: If there is a big data set , it is slower than Stochastic Gradient Descent algorithm. To be able to apply for big data set , it requires expensive equipments(GPU,CPU).

Stochastic Gradient Descent:

Pros: Choosing random variables give stochastic gradient descent big advantange comparing the Gradient descent algorithm. It's faster than the traditional Gradient Descent.

Cons: It works on large variance for singular example.