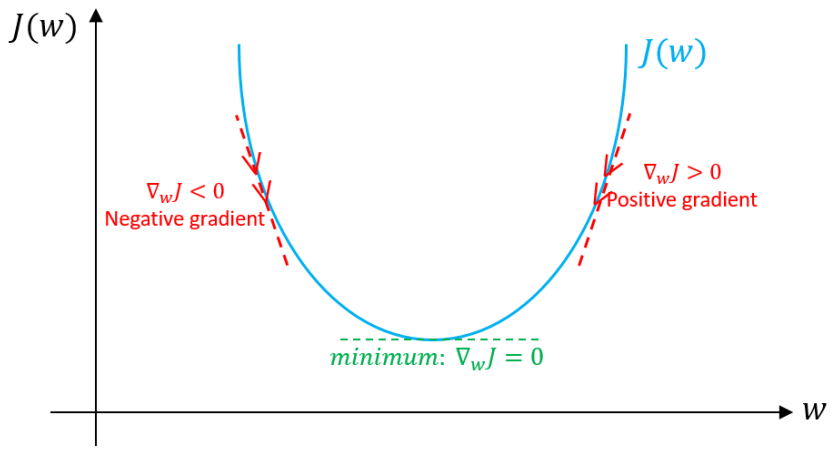
Calculus 2 – Bonus Assignment

"Calculating the minimum using Gradient Descent method"



Computer Science Department

Heraklion Crete

Professor: G. Tsagatakis

Student: Manos Chatzakis

AM: 4238

May, 2019

About the assignment

This is the assignment for HY111, in which we use the gradient descent method to find the minimum of some function models.

Programs / Languages:

- MatLab: Gradient Descent/Plots
- LateX: Part 1
- Microsoft Office: Presentation

The Gradient Descent Method:

Gradient Descent Method is an algorithm used in Machine Learning, in order to update models.

For this assignment, gradient descent will be used to calculate the minimum of 4 given functions.

The main idea contains iterations of $\boldsymbol{x} = \boldsymbol{x} - a\nabla(\boldsymbol{x})$ (x is initialized with a rand value). x is the prediction of the minimum's coordinate, while a is the learning rate.

Based on the number of the iterations and the learning rate, the error of the final prediction is different.

It can be used for 2 variable functions too, after some additions.

Part 2 and Part 5:

The code will be attached in the end of the presentation.

Part 1

For part 1, LateX was used. You may find the relative document in the following dropbox link.

https://www.dropbox.com/s/2n5ajk1y1p7vgui/Calculus2_Part1.pdf?dl=0

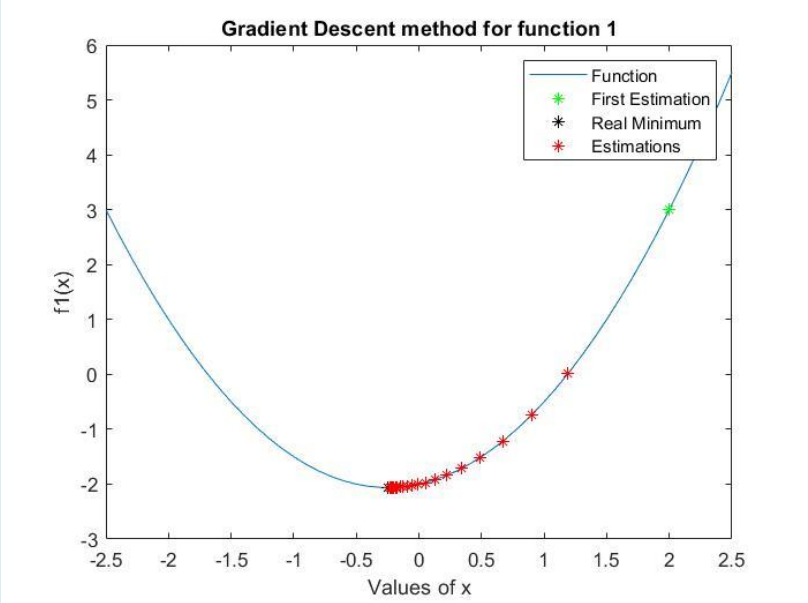
So, using classic calculus methods to calculate minimums, we have the following results:

<table>
<tr><th>Function</th><th colspan="2">x value</th><th>Minimum</th></tr>
<tr><td>1</td><td colspan="2">-0.25</td><td>(-0.25,-33/16)</td></tr>
<tr><td>2</td><td colspan="2">2/3</td><td>(2/3,131/27)</td></tr>
<tr><td>3</td><td colspan="2">-0.35</td><td>(-0.35,1.82719)</td></tr>
<tr><td rowspan="2">4</td><td>x</td><td>y</td><td rowspan="2">(-0.35,0,0.82719)</td></tr>
<tr><td>-0.35</td><td>0</td></tr>
</table>

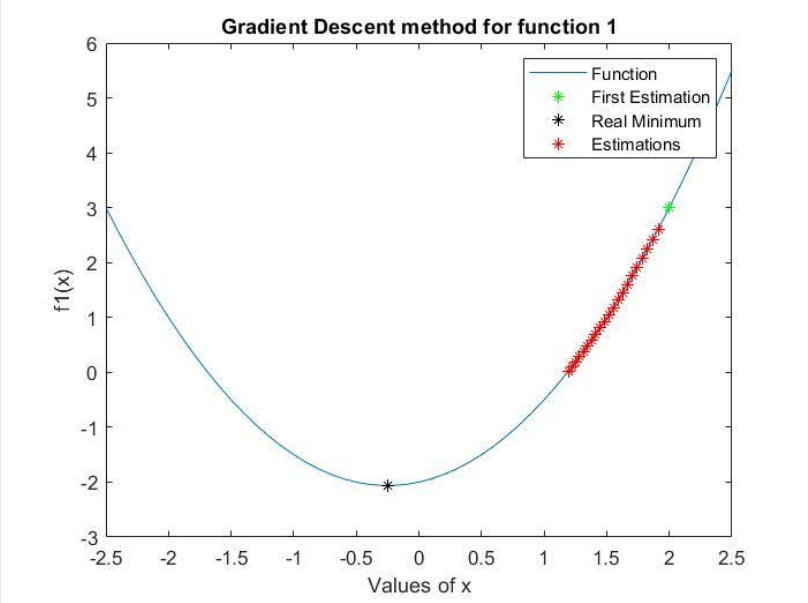
Part 3

For part 3 we will test the accuracy of the gradient descent method, holding a fixed value for iterations (etc. 20) and multiple values of learning rate.

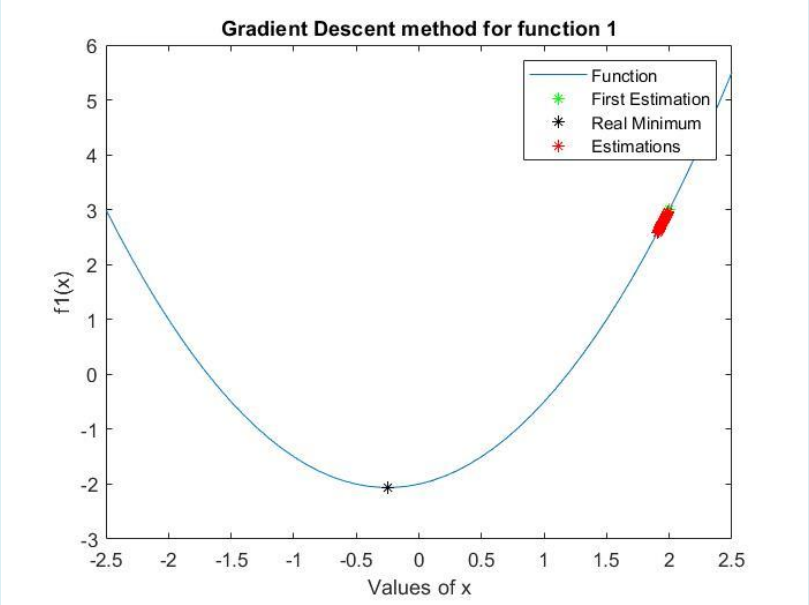
- Function 1:
    1. Learning rate = 0.1



    2. Learning rate = 0.01



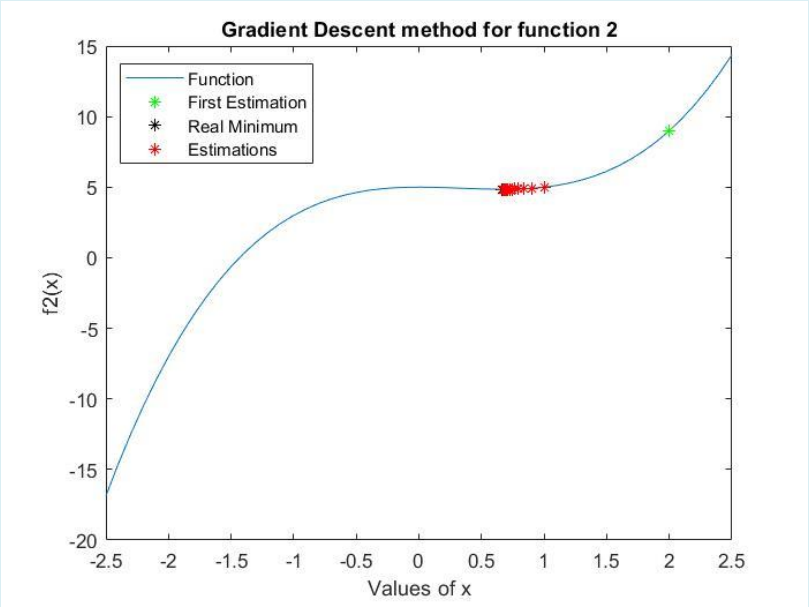
3. Learning rate = 0.001



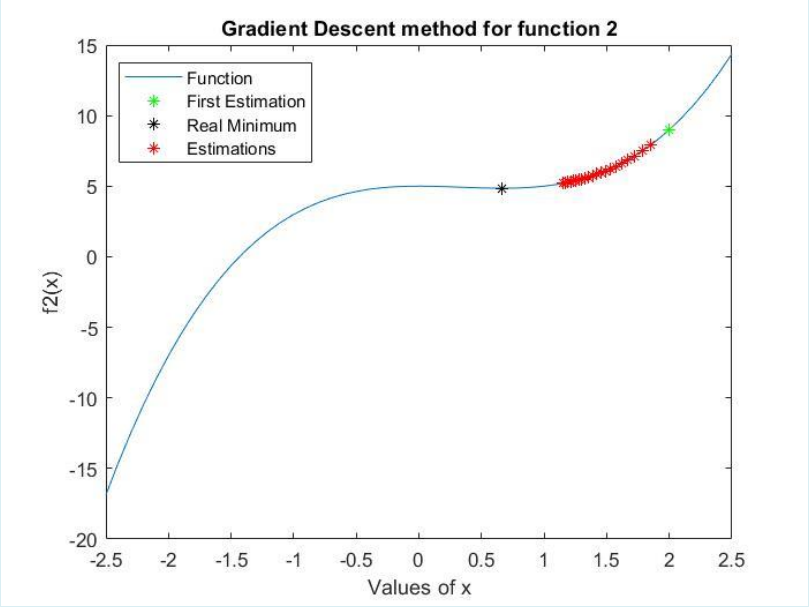
Conclusion(for fixed value of iterations):

Learning rate	Best Estimation	|Error|(x-x1)
0.1	-0.2334	0.0166
0.01	1.1926	1.4426
0.001	1.9031	2.1531

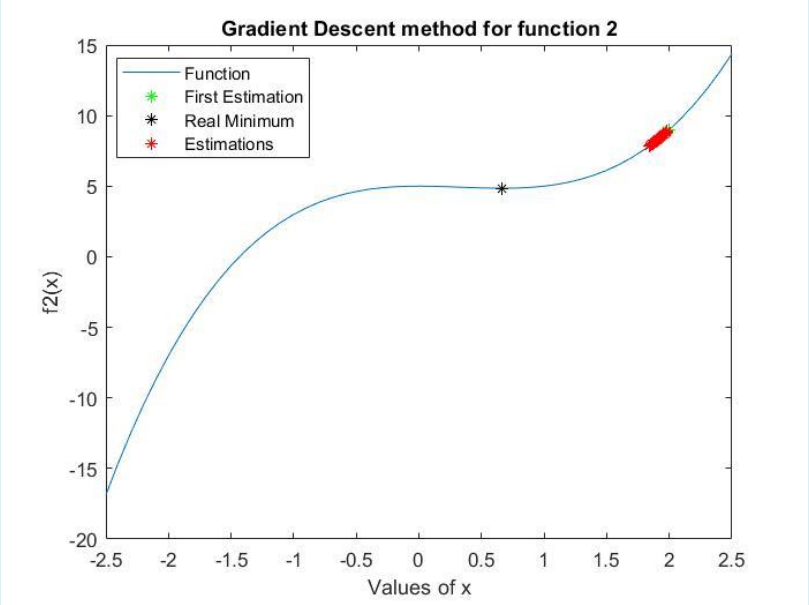
- Function 2:
    1. Learning rate = 0.1



    2. Learning rate = 0.01



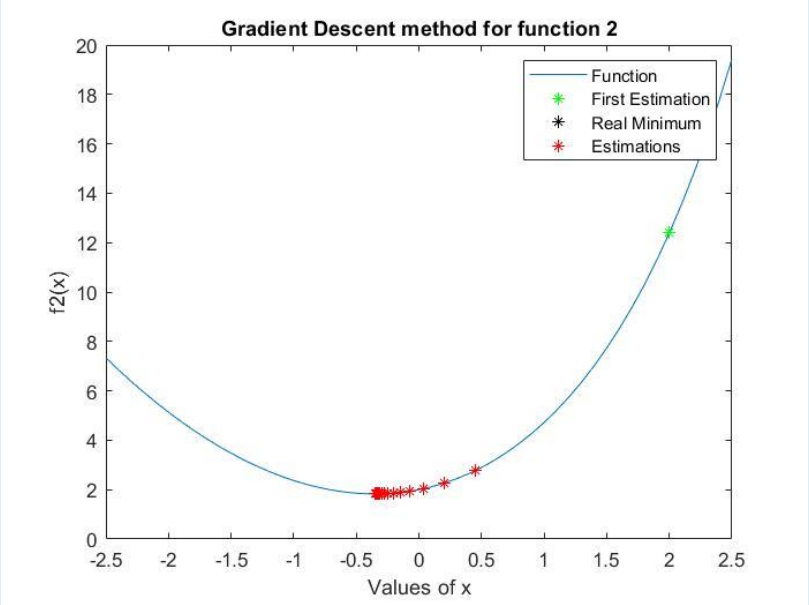
3. Learning rate = 0.001



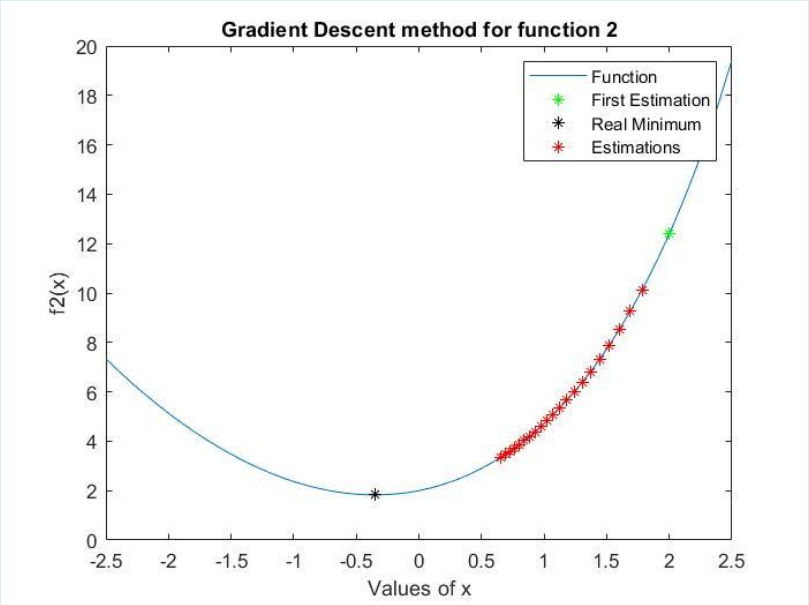
Conclusion(for fixed iterations):

Learning rate	Best estimation	|Error|(x-x1)
0.1	0.6690	0.0024
0.01	1.1546	0.4880
0.001	1.8408	1.1742

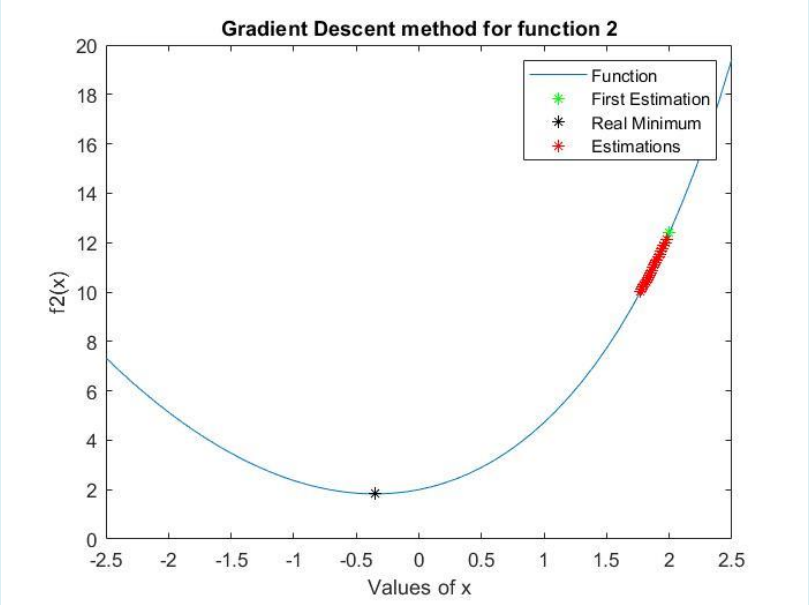
- Function 3:
    1. Learning rate = 0.1



    2. Learning rate = 0.01



3. Learning rate = 0.001



Conclusion(for fixed iterations):

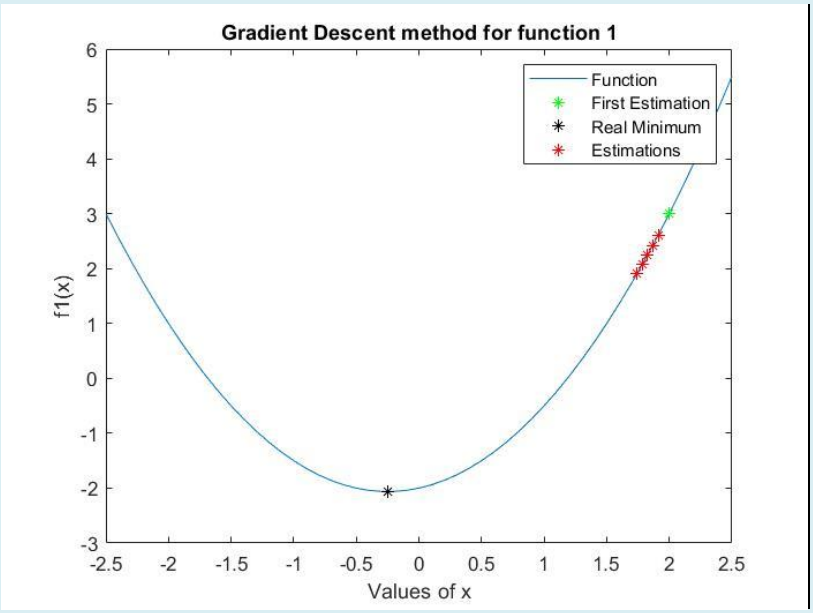
Learning rate	Best estimation	|Error|(x-x1)
0.1	-0.3505	0.0005
0.01	0.6576	1.0076
0.001	1.7714	2.1214

Overall conclusion: For a fixed (low) number (etc. 20 ~ 30) of iterations, gradient descent's estimations have better predictions for bigger learning rate (etc. a = 0.1). This is because with a big value for learning rate, we reach the minimum faster.

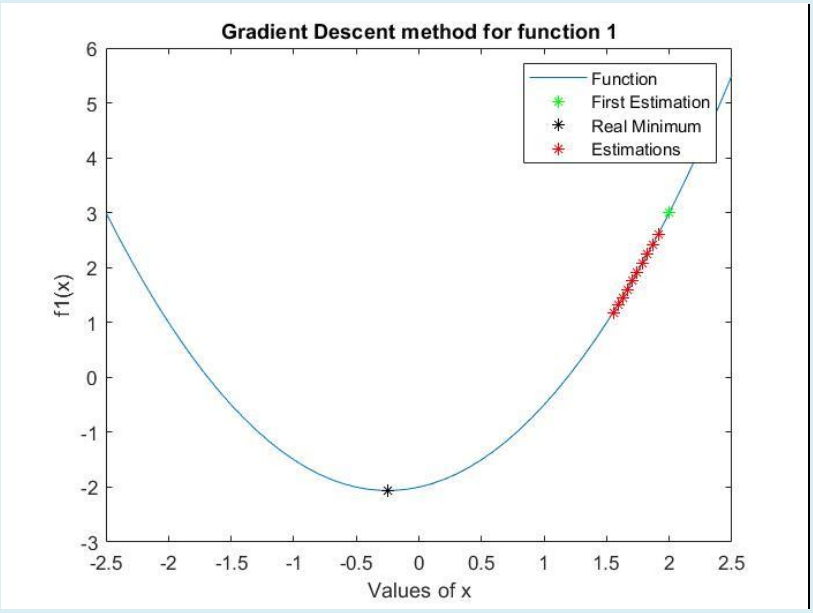
Part 4

In part 4, we keep a fixed value of learning rate (0.01) for different iteration values.

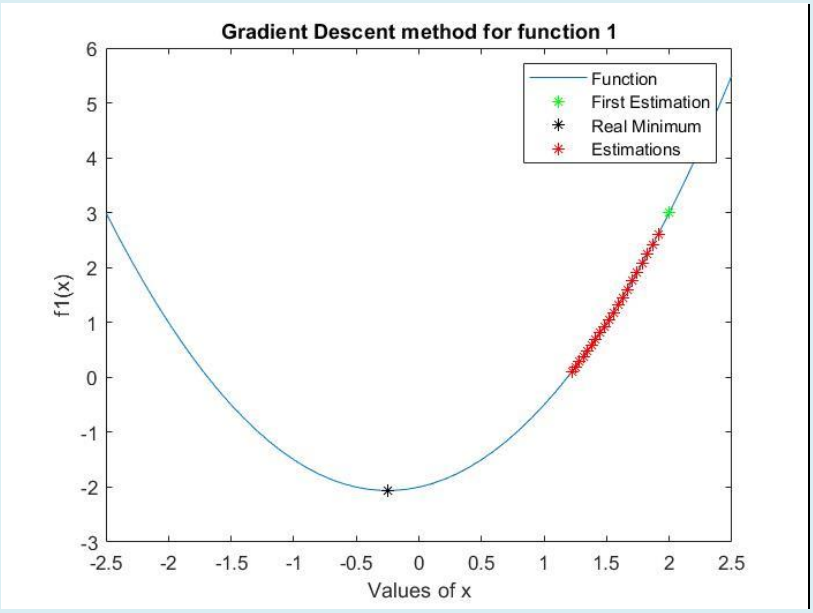
- Function 1:
    1. Iterations = 5



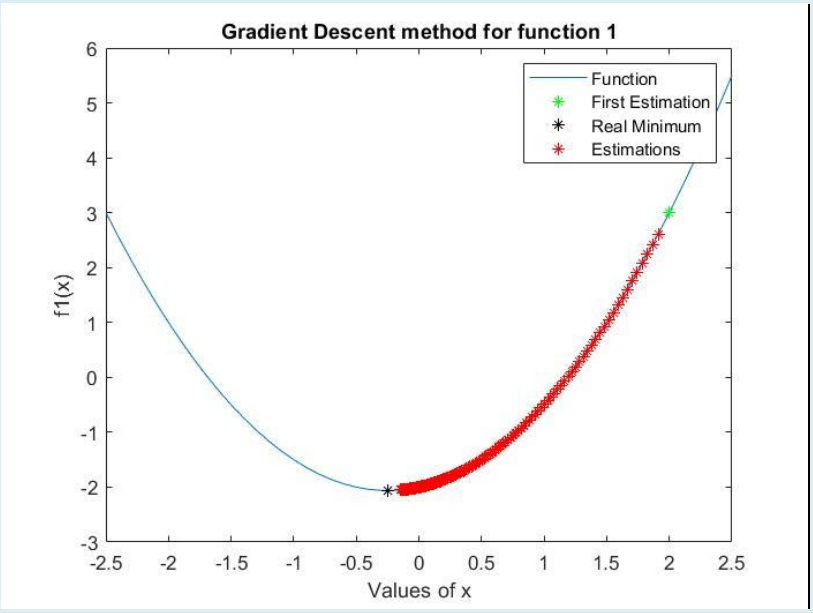
    2. Iterations = 10



3. Iterations = 20



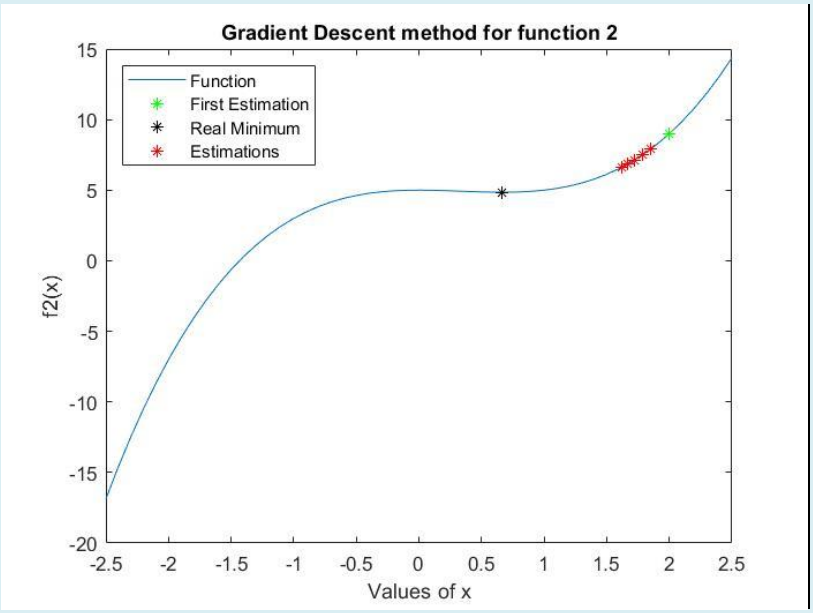
4. Iterations = 150



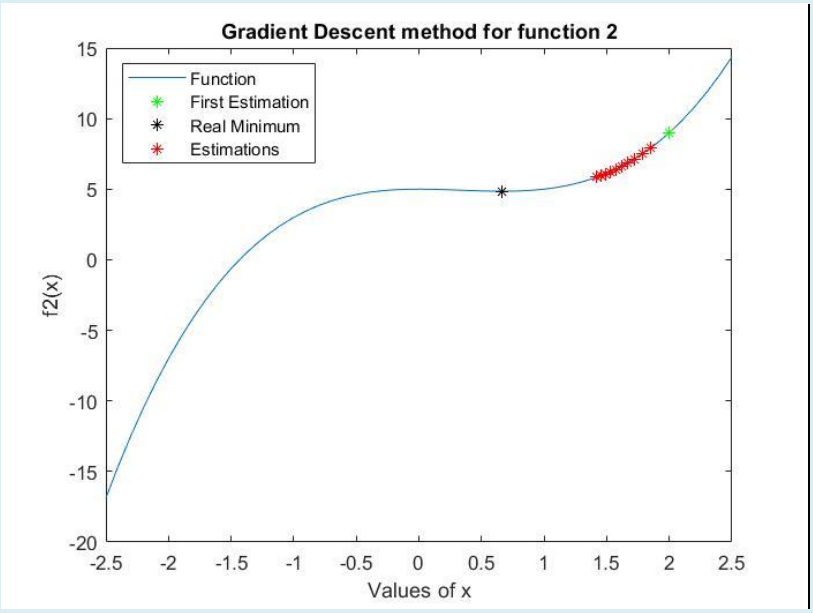
Conclusion for fixed learning rate:

Iterations	Best prediction	|Error|(x-x1)
5	1.7431	1.9931
10	1.5516	1.8016
20	1.2221	1.4721
150	-0.1435	0.1065

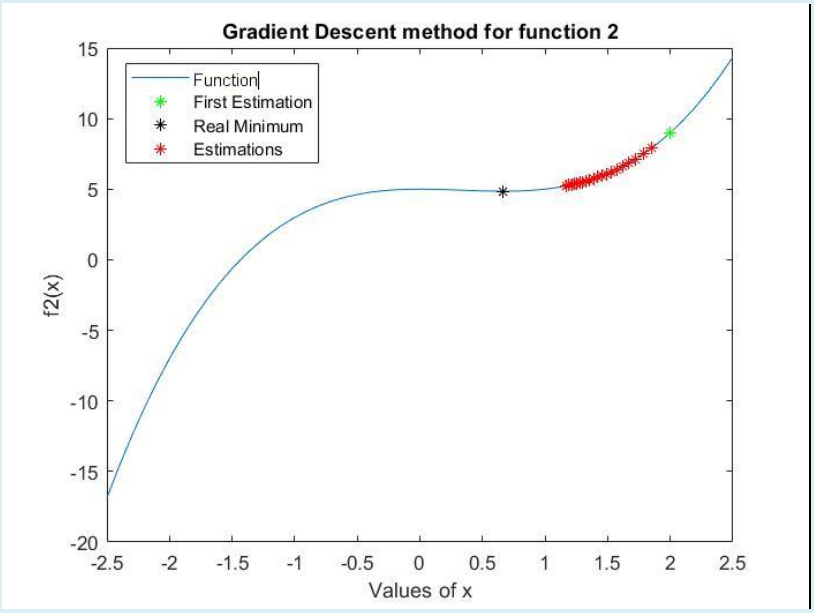
- Function 2:
    1. Iterations = 5



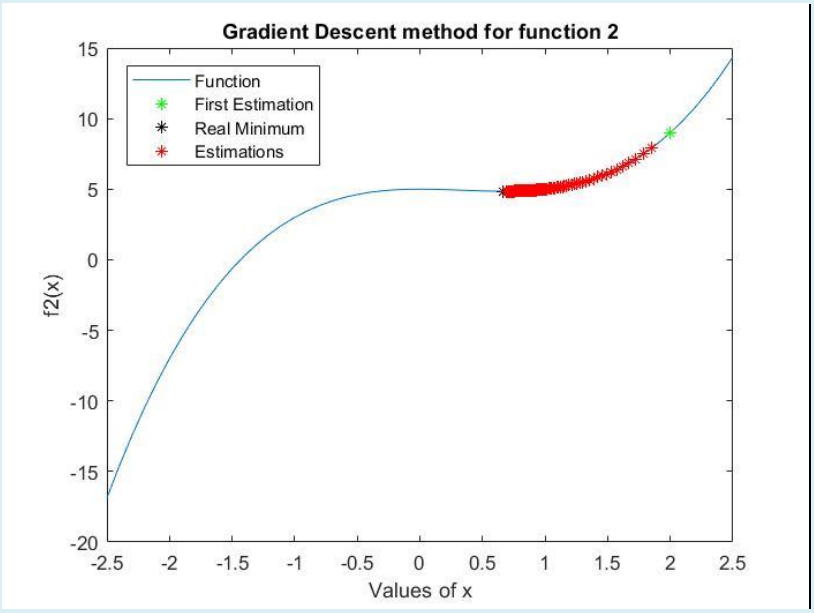
    2. Iterations = 10



3. Iterations = 20



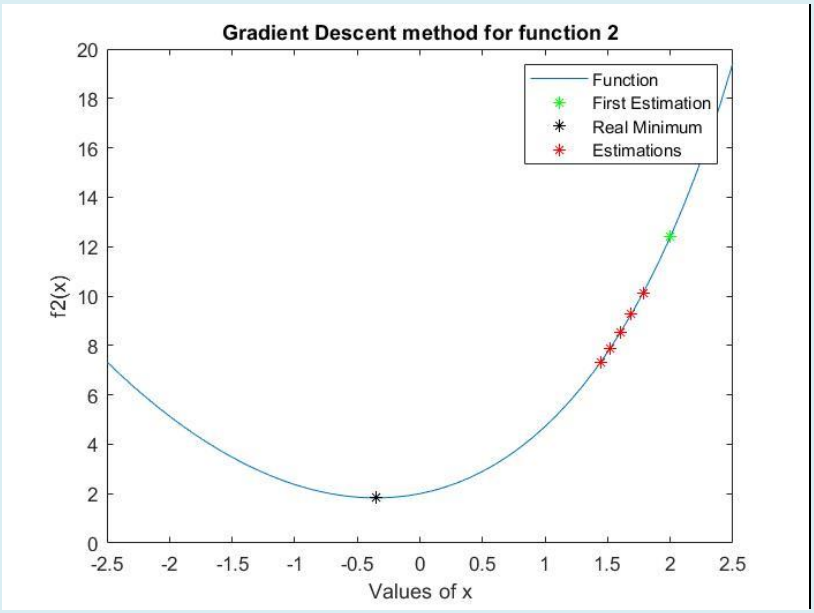
4. Iterations = 150



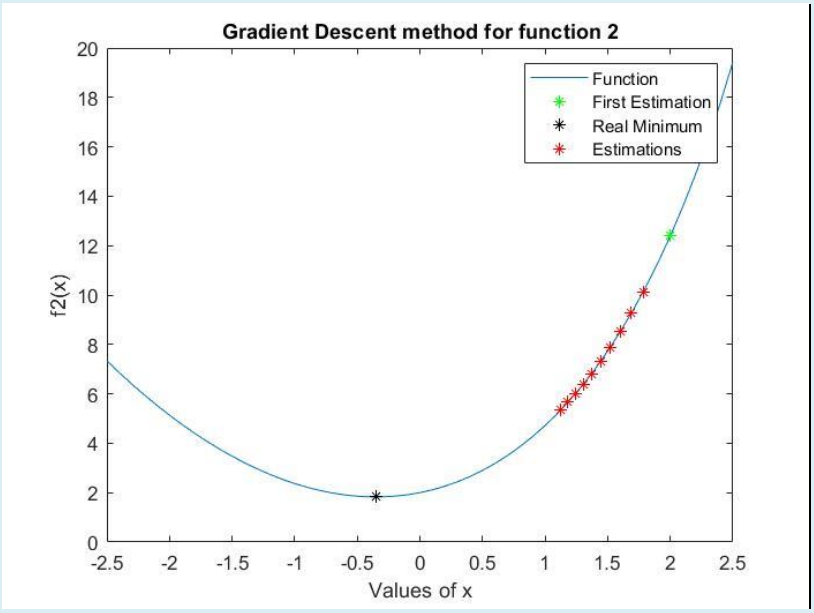
Conclusion for fixed learning rate:

Iterations	Best prediction	|Error|(x-x1)
5	1.6180	0.9514
10	1.4185	0.7519
20	1.1724	0.5058
150	0.6879	0.0213

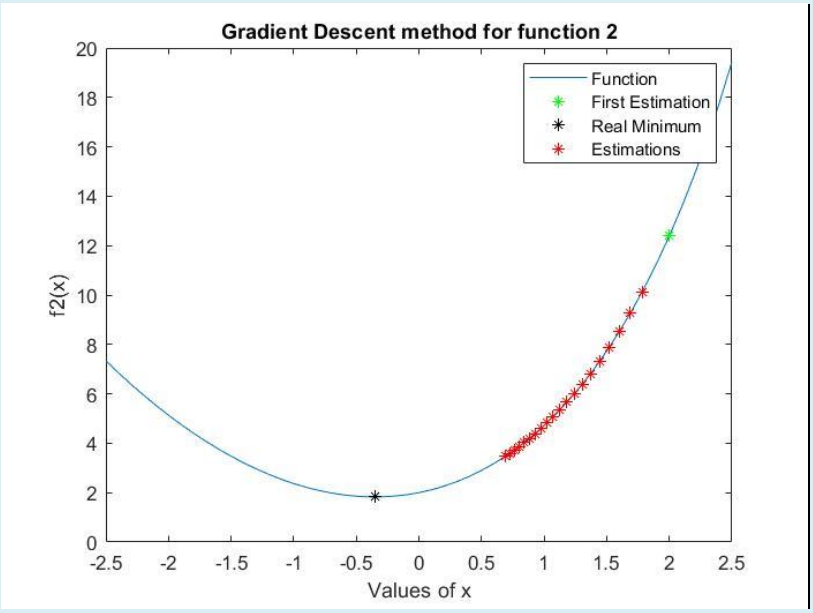
- Function 3:
    1. Iterations = 5



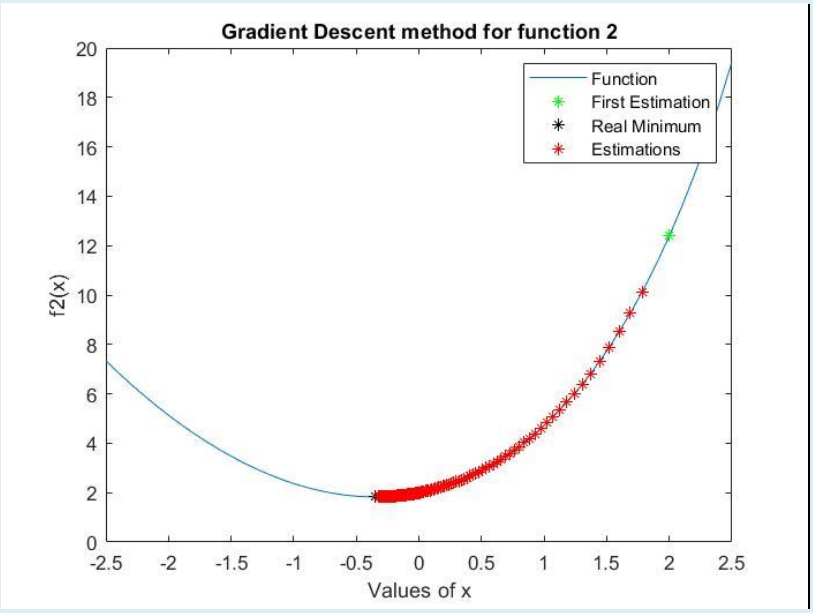
    2. Iterations = 10



3. Iterations = 20



4. Iterations = 150



Conclusion for fixed learning rate:

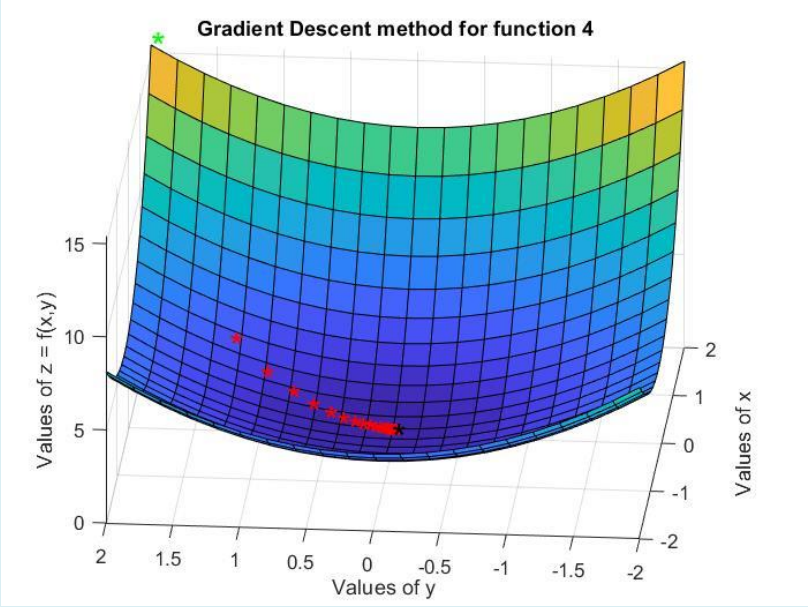
Iterations	Best prediction	|Error|(x-x1)
5	1.4421	1.7921
10	1.1255	1.4755
20	0.6914	1.0414
150	-0.3263	0.0237

Overall conclusion: Holding a fixed value for learning rate, gradient descent's results are more accurate for bigger amount of iterations. So, with a learning value such as 0.01 we are getting closer to the minimum when we use more iterations.

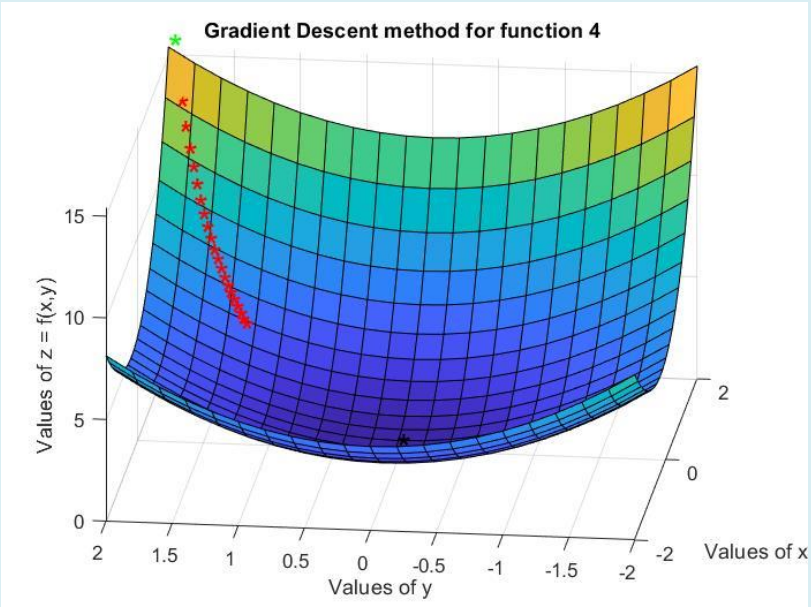
Part 6

Part 6.1: Accuracy of gradient descent for a 2 variable function with fixed number of iterations (20) and multiple values of learning rate:

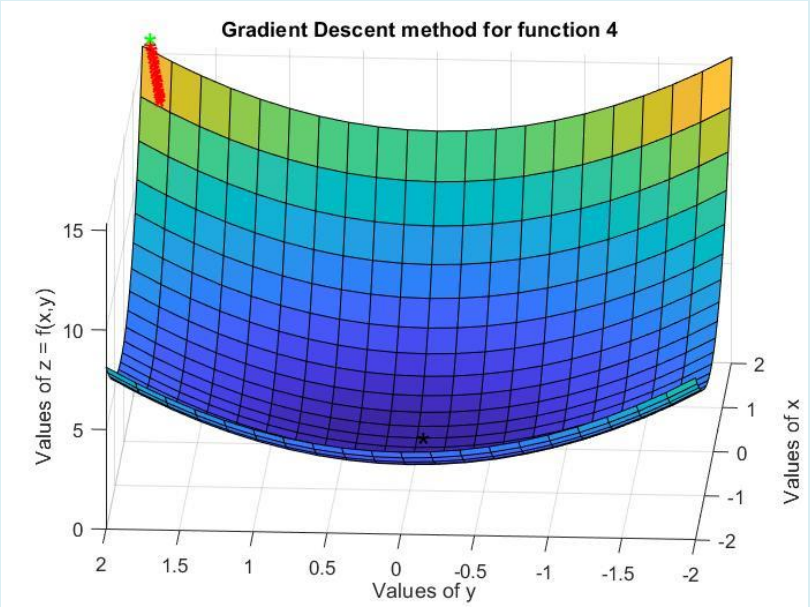
- Learning rate = 0.1



- Learning rate = 0.01



- Learning rate = 0.001



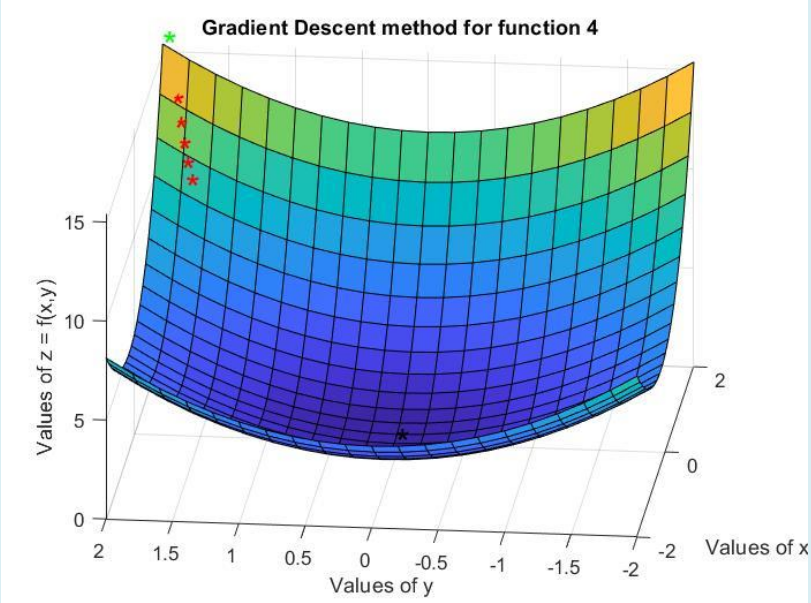
Conclusion:

Learning rate	Best Prediction(x)	Best prediction(y)	|Error|(z-z1)
0.1	-0.3500	0.0184	0.00031
0.01	0.6914	1.3085	3.35951
0.001	1.7809	1.9177	11.95721

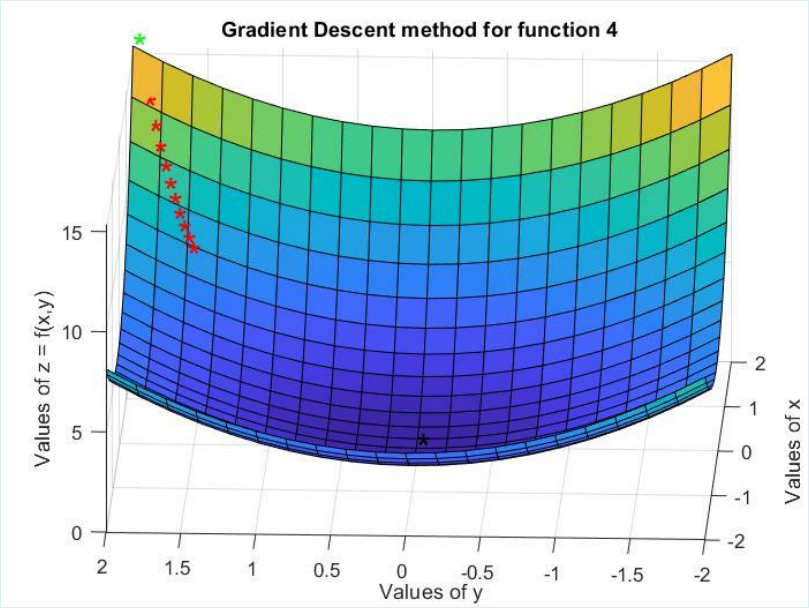
According to these results, the conclusion is the same as the one we got for a single variable function. When we hold a fixed low number of iterations, gradient descent's results are better if we use a bigger value for the learning rate.

NOTE: In this array, this error is the error of function 4 values (z1 – z2), and not the error between the x coordinates.

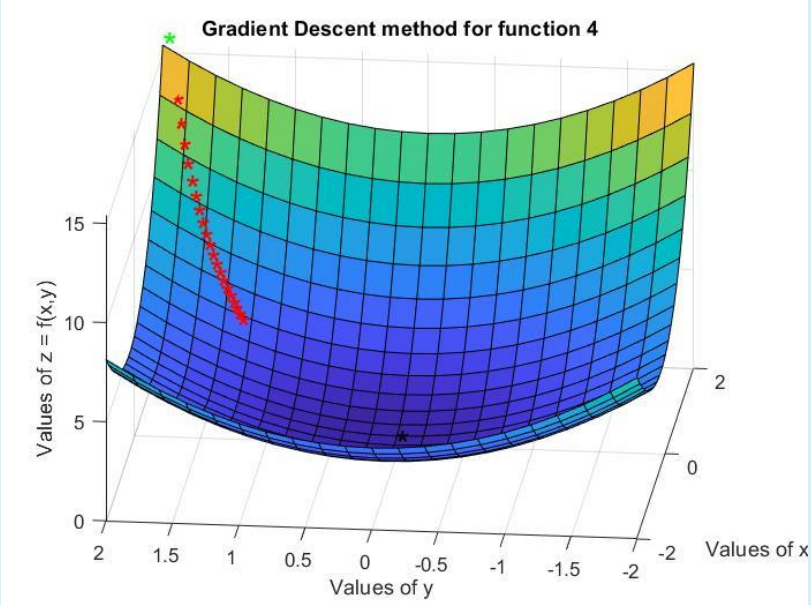
- Part 6.2: Accuracy of gradient descent for a 2 – variable function with fixed learning rate and multiple values of iterations:
    1. Iterations = 5



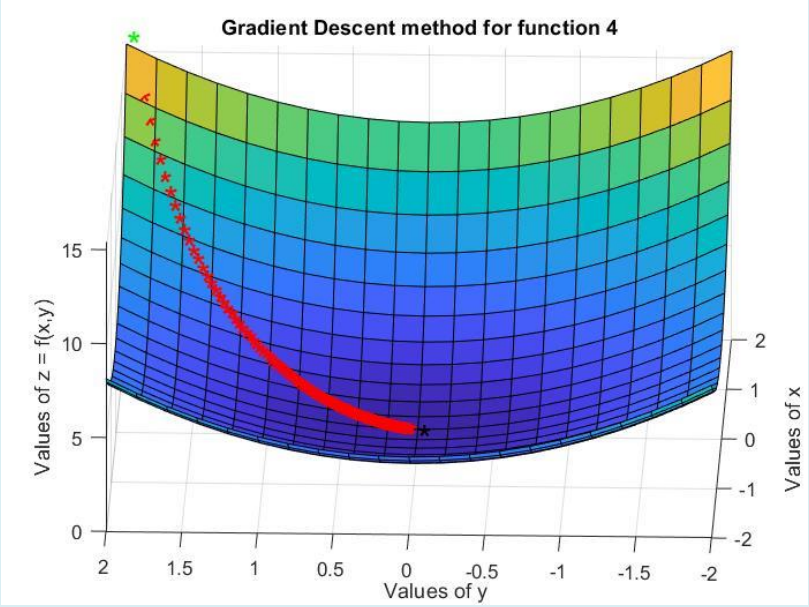
    2. Iterations = 10



3. Iterations = 20



4. Iterations = 150



Conclusion:

Iterations	Best estimation (x)	Best estimation(y)	|Error|(z-z1)
5	1.4421	1.7717	8.6210
10	1.1255	1.6015	6.0861
20	0.6914	1.3085	3.3595
150	-0.3263	0.0947	0.0098

So, according to these results, for a 2 variable function, when we hold a fixes value for learning rate (etc. 0.01) the predictions of the method are better for more iterations.

Part 2: Gradient Descent

```
%Manos Chatzakis
%AM: 4238
%This function executes and visualises the gradient descent method
%It is the solution to Part 2 and 3 of the assignment.
function least = GradientDescent(function_number,max_iterations,a,first_value)
    x = first_value - (a*Derivative(function_number,first_value)); %Get the first
value of the method.
    p = -2.5:0.1:2.5; %x_Axis will be from -2.5 to 2.5
    %Next Step: Draw the spesific function graph and mark the starting point
    if(function_number == 1)
        y = p.^2 + p/2 -2;
        plot(p,y);
        title('Gradient Descent method for function 1');
        xlabel('Values of x');
        ylabel('f1(x)');
        hold on;
        y1 = first_value.^2 + first_value/2 -2;
        plot(first_value,y1,'*','Color','green');
        hold on;
        plot(-1/4,-33/16,'*','Color','black');
        hold on;
    end

    if(function_number == 2)
        y = p.^3 - p.^2 + 5;
        plot(p,y);
        title('Gradient Descent method for function 2');
        xlabel('Values of x');
        ylabel('f2(x)');
        hold on;
        y1 =first_value.^3 - first_value.^2 + 5;
        plot(first_value,y1,'*','Color','green');
        hold on;
        plot(2/3,131/27,'*','Color','black');
        hold on;
    end

    if(function_number == 3)
        y = exp(p) + p.^2 + 1;
        plot(p,y);
        title('Gradient Descent method for function 2');
        xlabel('Values of x');
        ylabel('f2(x)');
        hold on;
        y1 = exp(first_value) + first_value.^2 + 1;
        plot(first_value,y1,'*','Color','green');
        hold on;
        plot(-0.3517,1.82718,'*','Color','black');
        hold on;
    end
    %Gradient Descent method starts:
    for i=0:1:max_iterations-1
        x = x - a*Derivative(function_number,x);
        %For specific function, add the draw each point the method gives.
        if(function_number == 1)
            y1 = x.^2 + x/2 -2;
            plot(x,y1,'*','Color','red');
            legend('Function','First Estimation','Real Minimum','Estimations');
        end

        if(function_number == 2)
            y1 = x.^3 - x.^2 + 5;
            plot(x,y1,'*','Color','red');
            legend('Function','First Estimation','Real Minimum','Estimations');
        end

        if(function_number == 3)
            y1 = exp(x) + x.^2 + 1;
            plot(x,y1,'*','Color','red');
            legend('Function','First Estimation','Real Minimum','Estimations');
        end
    end
    least = x; %Return the minimum.
end
```

```
%Manos Chatzakis
%AM: 4238
%Based on given input, this function returns the exact derivative
function output = Derivative(function_number,x)
    %f1 derivative
    if(function_number == 1)
        output = (2*x) + 0.5;
    end

    %f2 derivative
    if(function_number == 2)
        output = (3*x*x) - (2*x);
    end

    %f3 derivative
    if(function_number == 3)
        output = exp(x) + (2*x);
    end
end
```

Part 5: Gradient Descent a for 2 variable function

```
%Manos Chatzakis
%AM:4238
function [min_x,min_y] =
f4GradientDescent(max_iterations,a,first_value_x,first_value_y)

    first_value_z = first_value_x.^2 + first_value_y.^2 + exp(first_value_x);

    x_value = first_value_x - (a*f4Der_x(first_value_x));
    y_value = first_value_y - (a*f4Der_y(first_value_y));

    [x,y] = meshgrid(-2:.2:2);
    z = x.^2 + y.^2 + exp(x);
    surf(x,y,z);
    title('Gradient Descent method for function 4');
    xlabel('Values of x');
    ylabel('Values of y');
    zlabel('Values of z = f(x,y)');
    hold on;

text(first_value_x,first_value_y,first_value_z,'*','Color','green','Fontsize',20);
text(-0.35,0,0.82719,'*','Color','black','Fontsize',20);

    for i=0:1:(max_iterations-1)
        x_value = x_value - (a*f4Der_x(x_value));
        y_value = y_value - (a*f4Der_y(y_value));
        z = x_value.^2 + y_value.^2 + exp(x_value);
        text(x_value,y_value,z,'*','Color','red','Fontsize',20);
    end

    min_x = x_value;
    min_y = y_value;

end
```

```
%Manos Chatzakis
%AM:4238
function derx = f4Der_x(x)
    derx = exp(x) + (2*x);
end
```

```
%Manos Chatzakis
%AM:4238
function dery = f4Der_y(y)
    dery = (2*y);
end
```