M20 - Introduction to Computer Programming with MATLAB
Instructor: Prof. Enrique López Droguett, Ph.D.
Teacher Assistants: M. Fidansoy, G. San Martín, M. Pishahang, V. Vela.
Fall 2023 – UCLA
Student: *Alex Lie*
UCLA ID: *905901892*

HOMEWORK 5

Task 1: Gradient descent visualization over 2 variables

Introduction

I am using MATLAB to find the temperature in the middle and the end of an aluminum fin. To do this, I am finding the minimum of the function $f(x_1, x_2)$, where x_1 represents the temperature at the middle of the fin, and x_2 represents the temperature at the end of the fin. I will use the gradient descent method to find this minimum. I will also create a contour plot and add the points the gradient descent algorithm visits to show how gradient descent performs from two different initial points with the same learning rate.

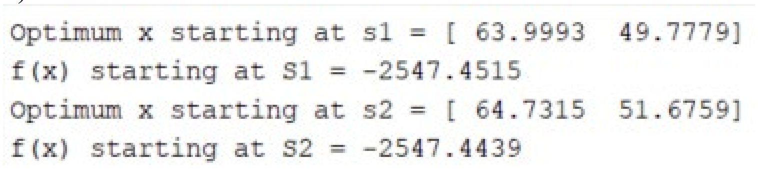
Model and Theory

1) $f(x_1, x_2) = 0.6382x_1^2 + 0.3191x_2^2 - 0.2809x_1x_2 - 67.906x_1 - 14.290x_2$

Methodology

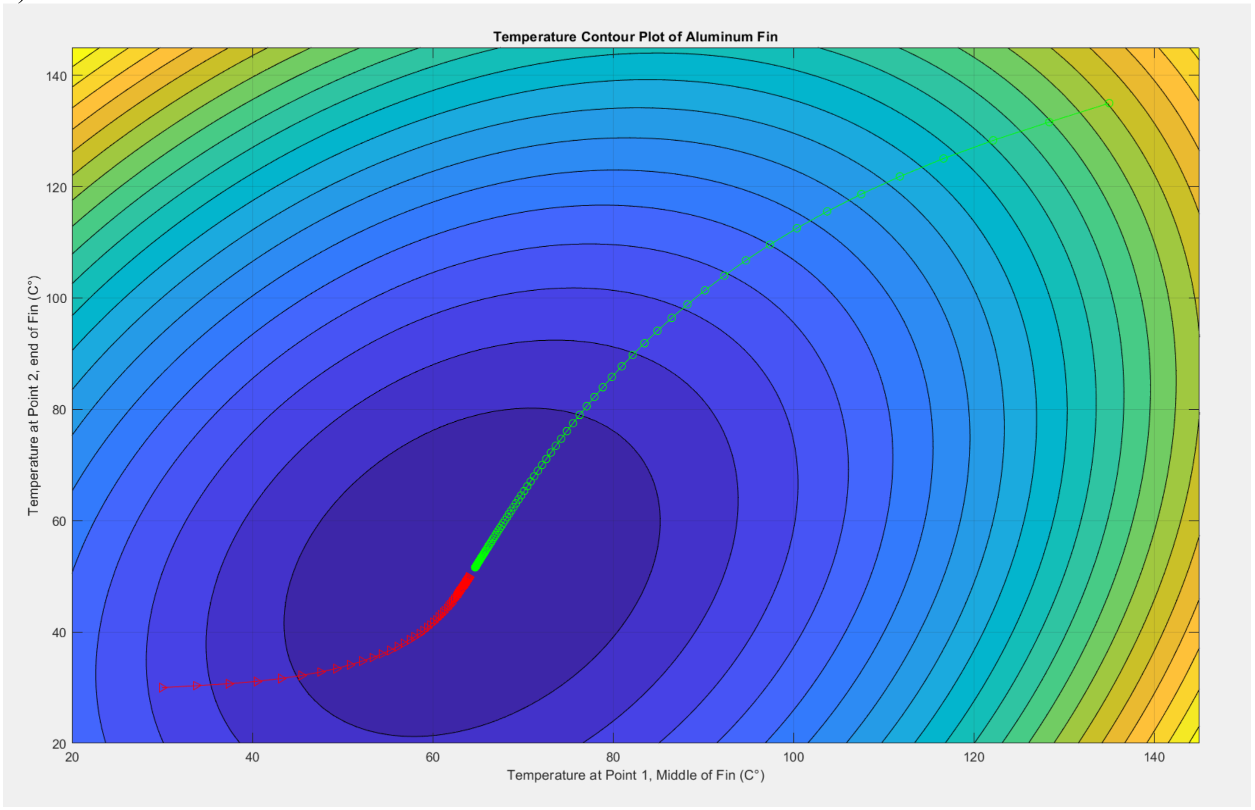
I first cleared the workspace and command window. I then defined the function (1) twice, once with a vector input with two elements, and the other with two scalar inputs. I did this so I can use both the meshgrid() and the gradOpt2D() function. I then created x1, x2, and f(x1,x2) points for my contour graph. I created a set of x1 and x2 coordinates using the linspace() function, creating 100 elements evenly distributed from 20 to 145. I used meshgrid() and the fx_mesh() function to create the corresponding f(x1,x2) values. I then created my contour graph using the contourf() function. I added labels, a title, and a grid. I then defined variables to use as inputs for the gradOpt2D() function. r is the learning rate. The learning rate must be negative if we are using gradOpt2D() to find the minimum of a function. S1 and S2 are the different initial points, tol is the prescribed tolerance, and imax is the maximum number of iterations. The gradient descent algorithm works by moving towards the direction of steepest descent, which is the negative gradient of the function f, and then repeating that until the minimum value is found. After calling the function twice using the different initial points, I outputted the optimum x1 and x2 values, as well as the minimum value of f(x1, x2) in the command window. I then plotted the points the gradient descent algorithm visited on the contour graph using the plot() function and the points outputted from the gradOpt2D() function. This will allow us to see how the gradient descent algorithm traverses through different points to find the minimum value. Lastly, I added comments to make my code easier to understand for other users.

Calculations and results

2)

```
Optimum x starting at s1 = [ 63.9993  49.7779]
f(x) starting at S1 = -2547.4515
Optimum x starting at s2 = [ 64.7315  51.6759]
f(x) starting at S2 = -2547.4439
```

M20 - Introduction to Computer Programming with MATLAB
Instructor: Prof. Enrique López Droguett, Ph.D.
Teacher Assistants: M. Fidansoy, G. San Martín, M. Pishahang, V. Vela.
Fall 2023 – UCLA
Student: *Alex Lie*
UCLA ID: *905901892*

3)



The output shows the optimal x1 and x2 values, as well as the minimum value of f(x1, x2) using the different initial positions of S1 and S2. The red graph shows the points and the path the gradient descent algorithm traverses through from the initial position of S1=[30, 30]. The green graph shows the points and the path the gradient descent algorithm traverses through from the initial position of S1=[135, 135].

Discussions and Conclusions

Looking at the two different paths with the different starting positions in graph 3, we can see they both end up at the same point. This is where the minimum of the function is located, which seems to be in the middle of the contour. The minimum is around -2547.4. There are two different temperatures that achieve this minimum. The temperature at point 1, the middle of the fin, is around 64 degrees Celsius. The temperature at point 2, the end of the fin, is around 50 degrees Celsius.

By looking at the output in figure 2, the difference of the minimums of the function using the different initial positions is 0.0076. The difference of the temperature at the middle of the fin using the different initial positions is 0.7322 degrees Celsius. The difference of the temperature at the end of the fin using the different initial positions is 1.898 degrees Celsius. I am not surprised that there is a difference because using different initial positions will likely result in differing optimal values. After all, the gradient descent method is an approximation.

M20 - Introduction to Computer Programming with MATLAB
Instructor: Prof. Enrique López Droguett, Ph.D.
Teacher Assistants: M. Fidansoy, G. San Martín, M. Pishahang, V. Vela.
Fall 2023 – UCLA
Student: *Alex Lie*
UCLA ID: *905901892*

Task 2: Optimization over 3 variables

Introduction

I am using MATLAB to find the displacement of three carts that are interconnected by springs and have three loads acting on them. I will find the displacements of the carts by minimizing the potential energy of the system. To find the minimum potential energy, I will implement the grid search and gradient descent methods and record how long it took each algorithm to find the minimum potential energy and displacements of the carts. I will then compare which method is more efficient.

Model and Theory

1) $f(x) = \frac{1}{2}\boldsymbol{X}^T[K]\boldsymbol{X} - \boldsymbol{X}^T\boldsymbol{P}$

2) $[K] = \begin{bmatrix} k_1 + k_4 + k_5 & -k_4 & -k_5 \\ -k_4 & k_2 + k_4 + k_6 & -k_6 \\ -k_5 & -k_6 & k_3 + k_5 + k_6 + k_7 + k_8 \end{bmatrix}$

3) $\mathbf{P} = \begin{Bmatrix} P_1 \\ P_2 \\ P_3 \end{Bmatrix}$

4) $\boldsymbol{X} = \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}$

Methodology

I first cleared the workspace and command window. I then declared all the k and P variables with their corresponding values. I then declared the matrices K (2) and P (3). I then defined the function (1) to be used in our optimization algorithms. I wrote the function two different ways: one has a 1x3 vector input, and the other has three numerical inputs. This is because the gradOpt3D() and gridsearch3d() functions access the inputs to my function in different ways.

I then declared variables used for the gradOpt3D() function. X0 is the initial point used in the gradient descent algorithm. r is the learning rate. The learning rate must be negative if we are using gradOpt3D() to find the minimum of a function. tol is the prescribed tolerance, and imax is the maximum number of iterations. The gradient descent algorithm works by moving towards the direction of steepest descent, which is the negative gradient of the function f, and then repeats that until the minimum value is found. I had to adjust the gradOpt2D() function to create the gradOpt3D() function, which will minimize our input function by optimizing three different variables instead of two. I wrote an equation that would approximate for the partial derivative in the third dimension using backward finite difference in the gradOpt3D() function. I then called the function and used tic and toc to record the elapsed time it took for the gradOpt3D() function to find the optimal x values. I then output the optimal X vector.

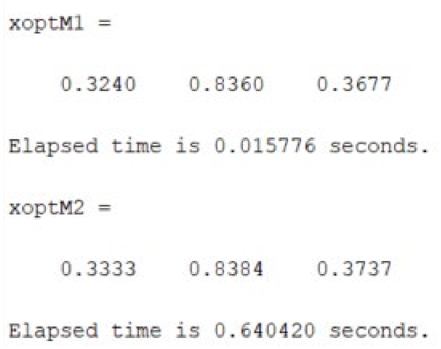
The grid search algorithm works by iteratively calling the function with every possible combination of inputs that we define, and then finds the input values that minimizes the function value. I then declared variables used for the gridsearch3d() function. lowerlim and upperlim are the lower limit and upper limit of the input values we will pass in our function, and N is the number of elements between the lower and upper limit spaced evenly that will also be passed in the function. I then modified the gridsearch2d()

function to create the gridsearch3d(), which will optimize three different variables instead of two. I created a third dimension and a third element in the vector output in the gridsearch3d() function. I then called the function and used tic and toc to record the elapsed time it took for the gridsearch3d() function to find the optimal x values. I then output the optimum X vector.

Lastly, I added comments to make my code easier to understand for other users.

Calculations and results

5)



```
xoptM1 =

    0.3240    0.8360    0.3677

Elapsed time is 0.015776 seconds.

xoptM2 =

    0.3333    0.8384    0.3737

Elapsed time is 0.640420 seconds.
```

The optimal point found by the gradient descent algorithm is $\boldsymbol{X} = \begin{Bmatrix} 0.3240 \\ 0.8360 \\ 0.3677 \end{Bmatrix}$. This algorithm took 0.015776 seconds to run. The optimal point found by the grid search algorithm is $\boldsymbol{X} = \begin{Bmatrix} 0.3333 \\ 0.8384 \\ 0.3737 \end{Bmatrix}$. This algorithm took 0.640420 seconds to run.

Discussions and Conclusions

I used my function fX1 (with a 1x3 vector input), an initial point of [0,0,0], a step size of 1e-4, a tolerance of 1e-3, and a maximum number of iterations of 100 for the parameters of the gradOpt3D() function. I used my function fX2 (with three numerical inputs), -1 and 1 as the lower and upper limits respectively, and a N value of 100 (number of elements between the lower and upper limit) as parameters of the gridsearch3d() function. Based on the results in figure 5, the optimal solutions for both methods are close, however, the gradient descent algorithm is faster than the grid search algorithm by 0.624644 seconds. Although this does not seem like much time, if we increase our N input in the grid search algorithm, then the time difference will be much larger. The gradient decent algorithm only went through 40 iterations, while the grid search algorithm went through 1000000 (100^3) iterations. The gradient descent algorithm is much more efficient than the grid search algorithm.