ECE3790 Lab#5

Mohammad Elsayyed

Maddie Salehpoor

`

“Participation Outline”

We, Maddie Salehpoor, and Mohammad Elsayyed attest that the work we are submitting is our own work and that it has not been copied/plagiarized from online or other sources. Any sourced material used for completing this work has been properly cited.

Signature: Maddie Salehpoor, and Mohammad Elsayyed.

In order for both of us to learn the material thoroughly, we worked on all the parts together and collaborated in developing the algorithms and the testing process. We both know the details of the algorithm and how the program works. We both put equal effort into the lab and we tried to work on different platforms such as discord, and paint to explain stuff, so we can keep each other included in our work.

Best,
Maddie, and Mohammad

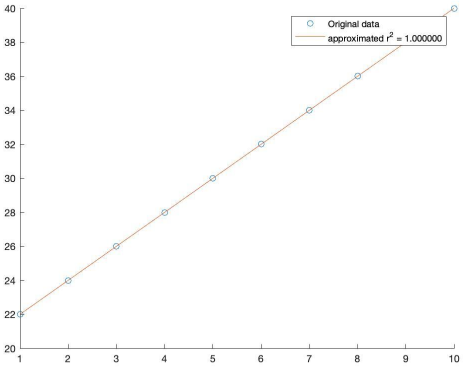
`

Problem 1: Regression

- **regres- sion code works properly. For example, you can create data y = a0+a1x+a2x2 by picking values for the coefficients. If you put this data through your polynomial regression function, you should recover the parameters exactly (to numerical precision) and get a correlation coefficient of 1.**
  In order to verify that our function is working properly, we tested our code with an arbitrary data set. We said that y = 2+20x, (i.e.,$a_0 = 2,\ a_1 = 20,\ x = [1:1:10].'$, ' $denotes\ transpose$) and we used the generated y and x, to create our input dataset to the linear regression function to calculate the a_0, a_1 and as we expected it calculates and returned the original values of a_0, a_1 we previously used. As seen below the achieved coefficients matches our original coefficients and r=1

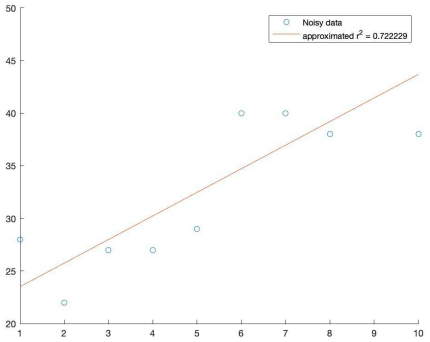
```
a1 =
     2

a0 =
    20

r2 =
     1
```

```
coeffA =

   20.0000
    2.0000

Does the values of coefficient match: true
```



- **Applies your function to a noisy data set (hopefully this is some interesting data, but you can just generate a noisy set yourself if needed).**

  We added random noise to our data and tested the function again. We got $r^2 = 75\%$, which is a pretty good approximation for randomly generated data. The value of the correlation coefficient is 73% which is expected since as seen in the figure below, the data generated is totally randomized and we wouldn't really expect to get a value of r^2 higher than 85%. The result is shown in the following figure:



`

Discussion

1. **Discuss your chosen data. Where did it come from? Was the function you programmed a good model for this data? Why or why not?**

   Since we only implemented the linear regression function, then we decided to choose a linear function as our input data, to make sure and verify that our function is yielding the correct prediction. Our function proved to be working since we got r^2=1 for the linear function. Then for the noisy data we used the same linear function with added noise (rand() function) and tested our output, which again in the above/previous figure proved to be working. A possibly not optimal data set for this function would be a data set that was following an exponential behaviour or a polynomial with a higher order than expected.

2. **Assuming you have n data samples, what is the computational complexity of applying multiple linear regression if we have m independent variables? Show your work/justify your answer.**

```
function [r2 c x] = linearregression(dataSet)

A = dataSet(:,1:end-1); → O(m)
b = dataSet(:,end); %marks → O(m)

x = (A'*A)\(A'*b);

dim = size(x); O(1)
length = dim(1); %get the column O(1)

dim2 = size(b); O(1)
length2 = dim2(1); O(1)

c = zeros(length2,1); → O(m)

average_mark = mean(b); → O(m)
St = sum((b-average_mark).^2); → O(1)

c = c + x(1); → O(m)
%Determine Sr:
for i=2:length
    c = c + x(i)*A(:,i);
end

Sr = sum((b-c).^2); → O(1)

r2 = (St - Sr)./St;

end → O(1)
```

n × m data

(m−1)(n) = O(m*n)

Created by Paint S

```
x = (A'*A)\(A'*b);
```

A'A → O(m^2 n)

A'b → O(m n)

The division (/) assuming LU decomposition: O(m^3)

⇒ O(m^2 n) + O(m^3)

Therefore using the above operation counting we get the overall time complexity of O(m^2 n + m^3), for large enough value of m we can conclude that the overall time complexity is O(m^3)

3. **Why/how can linear regression be applied to data that follows an exponential trend? A high-level explanation is sucient.**
   NOTE: Ask julian to see if the X^2 explanation from statistics works..

`

If we have an exponential model $y = \alpha e^{\beta x}$ to a set of data. The goal is to try to make this look like a linear regression so that we could use the linear regression algorithm:
$ln(y) = ln(\alpha) + \beta x$
The we would define new variables :
$u = x,\ v = ln(y)\ and\ a_0 = ln(\alpha)\ and\ a_1 = \beta$
$v = a_0 + a_1 u$
Then we could solve this linear regression and fit a line for the data set. Then we would convers the coefficients back to the original coefficients and calculate
$y = \alpha e^{\beta x}$

4. **What are the memory requirements of your regression implementation? Is there any room to improve the memory requirements? If so, how?**

   For the input of our linear regression function, we have matrix A which a matrix of size (n*m), and fo**r vector b which's size (m*1)**, and lastly the vector of the $a_0, a_1, a_2 \ldots a_{m-1}$ which stores the coefficient with size (m*1). Therefore the total memory needed would be equal to **mn + n + m.**

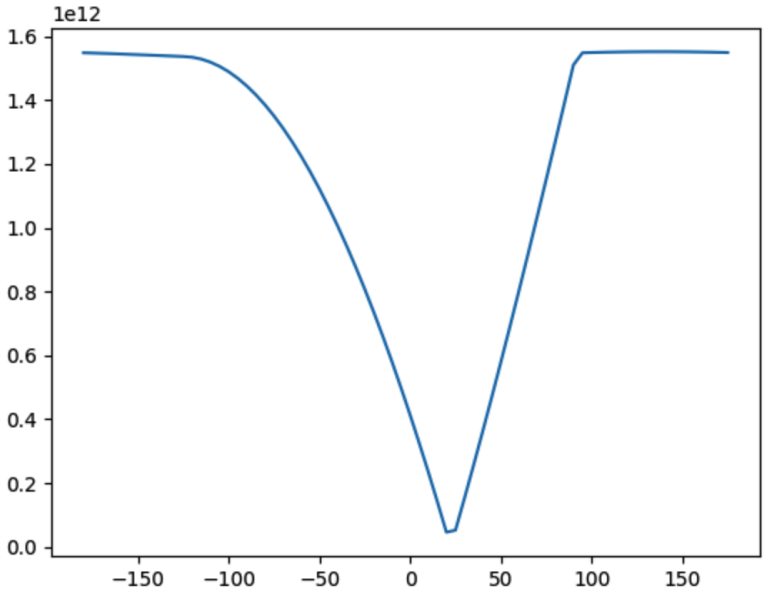
   I believe the way we implement the method needs to allocate the sizes mentioned below, since we're using matlab libraries to do the multiplication, not relying on pointers for indices and what not, however, I believe this would reduce it at best to **mn + m.**

`

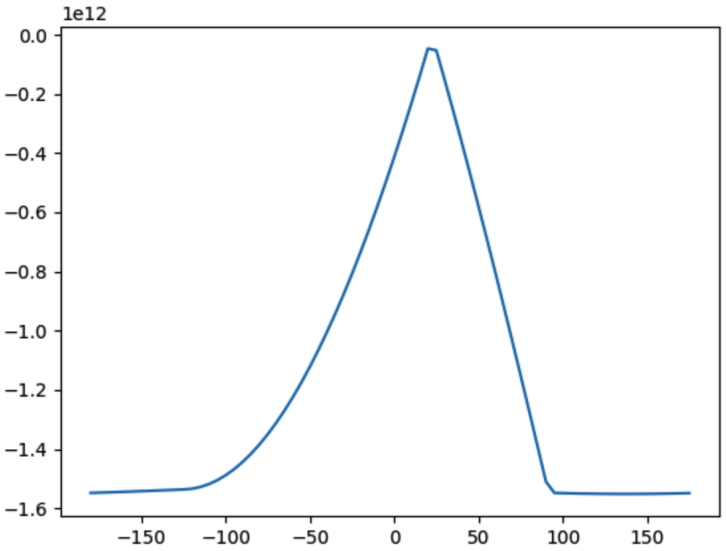
Problem 2 {1-D Optimization}

Validation and Data Collection

Once you have completed writing your functions and main program you need to:
In order to be able to verify that our code is working, we plotted the output of the spaceship function for different values of angles (x) between -180 and 180.
The following figure shows the plot:

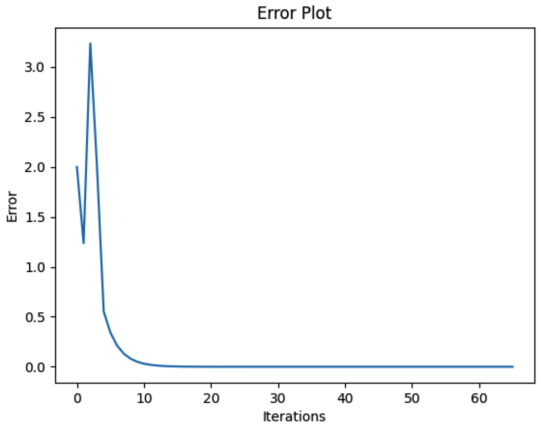


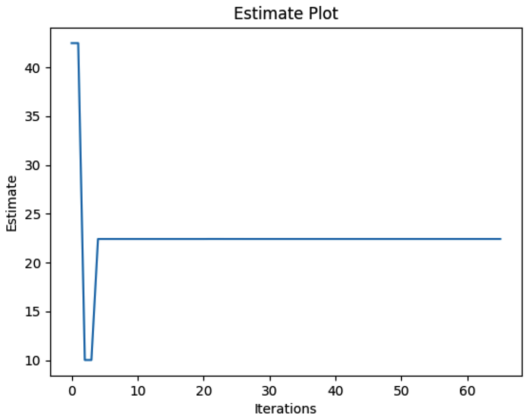
As seen in the plot above the minimum of the function is at an angle around 25 degrees. Our golden search function looks for the maximum of the graph, so we implemented a handle method which would return -f(x) instead of f(x). If we plot the -f(x) we get the following plot. And when given to our algorithm the maximum of -f(x) which would be the minimum of f(x) is determined.:



`

- **Produce plots of the convergence history (error level and minimum value convergence each as a function of iteration, so two plots) for both the spaceship problem.**





Evaluation and Discussion

Answer the following questions:

1Suppose we added another parameter to our spaceship optimization problem so that we could optimize over a fixed angle and fixed speed. Would you still be able to apply something like the golden section search for this 2-D problem? If so what changes would be required?

Yes its possible to apply the golden search algorithm in 2-D and actually the solution to this problem is relatively simple. Since we are moving to 2-D, then we would have to have to dimension for x low, (x_{lx}, x_{ly}), and x upper (x_{ux}, x_{uy}), and s=the same goes for x1 and x2, we would choose to points in between the rectangle xlow and x upper (so the two point lie in between xu and xl in a 2-D space.) and x1 would have (x_{1x}, x_{1y}) and x2 would have (x_{2x}, x_{2y}). and then we would call the handle functions to compare $f(x_{1x}, x_{1y})\ with\ f(x_{2x}, x_{2y})$ and the remainder of the algorithm's process stays the same. So we would determine the estimation at that stage and calculate error and update our coordinates.

What would you need to do in order to try and apply gradient search to that problem?

- **If we use gradient search, we can use the 1-D golden section search algorithm we developed already to minimize *f*(α) at each iteration, assuming we are at a starting point and we found the direction of steepest descent, now we'll need to determine how much distance we should tr**avel (i.e., travel step α) in that direction to arrive at another point with a better/closer solution and once we establish the equation $x_{new} = x_{old} + \alpha d$, and from this equation we can keep use our golden section search calculating the optimal α, in the least amount of iteration, or perhaps reaching more accurate results.

`

2. How many iterations did it take your search to converge and to what error level? From your results do you see value in trying to reduce the number of function evaluations for this optimization problem? What about more complicated/realistic problems?

Error level: 0.0000000000001 = 10^{-12} and as seen in the previous plot, the total number of iterations is 66.
However, if we take a look at the estimation plot we can see that around 10-60 interactions, the estimated value of angle is converging to an almost single value, so it depends on the number of decimal places that we want our function to be accurate to. So in this case maybe it would be enough to be accurate to 1 decimal figure, but in another more complex application we need more accuracy.

In conclusion, there is a trade between time and accuracy, in this case our black box, or the computations for finding f(x) were relatively fast so we could make our values more and more accurate, however in more complex examples we have to be careful about time and accuracy at the same time. So if the cost of time is too high and it's better for the results to be less accurate but take less time, then we could adjust the error tolerance accordingly.

`