University of Manitoba

ECE 3790: Engineering Algorithms **Winter 2021**

Lab 5: Regression and 1-D Optimization

April 5-6, 2021

The purpose of this lab is to give you practical coding experience. It will also reinforce concepts we've covered in class.

Important: A portion of your marks for this lab, and every other lab in this course, will be based on evaluation during your specified lab period. **Please be sure to attend your lab section.**

Labs can be performed in groups of two and feel free to interact with your classmates. **Code should be well documented**.

Each group is responsible for turning in a lab report that includes the answers requested at the end of each problem and a title page. The due date is 11:59pm two weeks after the lab.

Important: Please include the following signed statement with your submission.

I (We), [insert name(s)] attest that the work I am (we are) submitting is my (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(s)]

Also Important: Labs done in pairs must be submitted with a short paragraph outlining each partner's contribution.

Introduction

In this lab we are going to do some regression and optimization. First we are going to try to establish a model for some physical data. Then we will find the right trajectory by which to leave earth (in a *very* simplistic model) in order to make it off the planet and intercept other planets in our solar system.

Problem 1 – Regression

Data

At the heart of any regression problem is data. We give you flexibility in choosing data for applying regression, and encourage you to find something physically interesting. Of course Covid cases offer interesting trends to try and model using regression, but there are many other possibilities. Kaggle (kaggle.com) is an online community for data scientists. It is a great repository for all kinds of data sets (see https://www.kaggle.com/datasets). You may find something interesting there. If you don't want to go digging for data, you can always generate your own. Keep reading to see what kind of data you will need based on the functions you have to write.

Functions

For this problem you can use whatever programming language you prefer, and use linear algebra libraries for solving systems of equations. Personally I believe Matlab/Python is the right choice here, because as rapid prototyping tools you should get them up and running faster than other languages. Note: the point is not just to call some built-in regression/curve-fitting function.

Implement **one** of the following functions (we suggest you implement all of them but it has been a long semester):

A) A function `linearregression`. It should take in a set of sample data and produce the corresponding linear regression coefficients. It should be capable of multiple linear regression, so the input data should be a matrix (or 2-D array) where the first m columns are the independent variables at each sample point, and the last column is the dependent variable (or some format of this sort that works for you). The function should also calculate and return the correlation coefficient.

B) A function `polynomialregression`. This should take in the polynomial order and one-dimensional input data, convert it to multi-linear data, perform regression, and return the

model parameters and correlation coefficient. Note that the correlation coefficient should be computed in the linearized space.

C) A function `exponentialregression`. This should take in the data, linearize it, call `linearregression` and return the model parameters and correlation coefficient. Note that the correlation coefficient should be computed in the linearized space.

Main Program

Write a main program `Lab_5_Problem_1` that:

- Validates your chosen function by generating data and using it to show that your regression code work properly. For example, you can create data $y = a_0 + a_1 x + a_2 x^2$ by picking values for the coefficients. If you put this data through your `polynomialregression` function, you should recover the parameters exactly (to numerical precision) and get a correlation coefficient of 1.
- 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).

Analysis

From the main program produce appropriate plots 1) showing that your implementation works and that error quantification (correlation coefficient) is valid, and 2) showing how the chosen model fits a non-validation noisy data set. Write up a brief discussion of the results commenting on whether or not you believe the model has predictive value based on the quantified fit (correlation coefficient).

Discussion

Answer the following questions:

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?
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.
3. Why/how can linear regression be applied to data that follows an exponential trend? A high-level explanation is sufficient.

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

Hand in:

All codes, plots, your analysis writeup, and answers to the discussion questions.

Problem 2 – 1-D Optimization

In this problem you will write code for the golden section search. You'll validate these on a very simple function. Next you'll do some optimization of a more interesting problem.

You've been given a function `spaceship` in both Python and Matlab. This function takes a single argument, the angle of departure of a spaceship (red x) from the planet earth at time 0, and simulates the time-evolution of the spaceship as it traverses through our solar system (a simple model consisting of the Sun, Earth, Mars, Jupiter, Saturn and Uranus). The model doesn't even compute forces on the spaceship, and depending on the angle of departure the sun can be a serious problem but we will ignore that. I encourage you to look at the function and think about how it works. The input angle is in degrees. The function returns the minimum distance recorded between the spaceship and Saturn. Our goal is to minimize this distance by appropriately changing the angle (and nothing else so that this remains a 1-D optimization problem). Thus the function `spaceship` is our objective function $f(x)$ where x is the angle of departure. Overall you don't need to know how it works, and can simply use it as a "black box".

Functions

You will probably want to write your functions in Python or Matlab as the provided code is in these languages. Otherwise, you will probably have to convert my code to another language. That is ok too, but may not be worth the effort. Also, the right way to do this is to pass a function handle to your method.

A) Implement a function called `goldensectionsearch` that, given a function (handle), lower and upper bracketing bounds, error tolerance, and maximum iterations, will try to find the maximum of the function.

Main Program

Write a main program `Lab_5_Problem_2` that:

- Solves the spaceship optimization problem using your `goldensectionsearch` function and stores the convergence history (error and estimate) of the solution. Remember, your function finds the maximum, but you want to find the minimum distance. Note that once you know things are working it may be worth turning off plotting in the objective function to speed things up.

Validation and Data Collection

Once you have completed writing your functions and main program you need to:

- 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:

1. Suppose 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 to this 2-D problem? If so what changes would be required? What would you need to do in order to try and apply gradient search to that problem?
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?

Hand in:

Hand in all code, plots, validation, and your answers to the discussion questions.