Duncan Di Mauro
UID: 805163177
CEE/MAE M20
November 5, 2019

Homework 5

1 Optimization

1.1 Introduction

The goal in this problem is to explore optimization through the creation of various functions, which are then used for a gradient descent algorithm, along with a second-order optimizer known as Newtown's Method. These algorithms are ultimately used to find the minimizers of a particular function f(x). The first goal of this problem is to create a function that will compute the gradient of a given function f(x). Next, another function to do gradient descent should be created, which utilizes the previously created gradient function. In the gradient descent function, the minimizer is updated according to:

$$x_{k+1} = x_k - \gamma f'(x_k). \tag{1}$$

Next, a function used to compute the second gradient of a function should be created. Afterwards, a function utilizing Newton's method is created, where the minimizer is updated according to:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \tag{2}$$

Finally, a script is written which provides the initial conditions x-initial (x_0) = 4, the amount of iterations to be used = 100, and the given function to use is:

$$f(x) = 3x^2 - 4x + 6. \tag{3}$$

The script calls both the gradient descent and newton's method functions using these initial conditions, and then finds the errors at each iteration for each method (the error is equal to the distance from the current minimizer to the global minimum, which is calculated by hand). Then, the script should plot the error vs. iteration for both optimizers on the same plot.

1.2 Models and Methods

First, a function `computeGradient(f, xk, dx)` is created, with `f` being a function handle, `xk` being a point at which to evaluate the derivative of the function, and `dx` being the step size for computing the derivative. This function utilizes the 3-point central difference equation found in the lecture slides. Then, the function outputs a value for the derivative `dydx`.

Next, a function `gradientDescent(f, x0, iterations)` is created, with `f` being a function handle, `x0` being the starting point of the optimization, and `iterations` being the number of iterations to run. This function outputs `xMins`: a vector containing all the minimizers, and utilizes equation (1) to update the minimizer values. It is important to note that the `computeGradient` function is utilized here to find the first derivative of the function in equation (1). This function starts out by pre-allocating space for out `xMins` array. Next, gamma and dx are assigned to their respective

values. Finally, a `for` loop is created. This loop first assigns `xMins(k)` to the current value of `xk`, and then updates `xk` using equation (1). This way, the loop allows for the initial value of `xk` to be stored, and then updates it with the given amount of iterations.

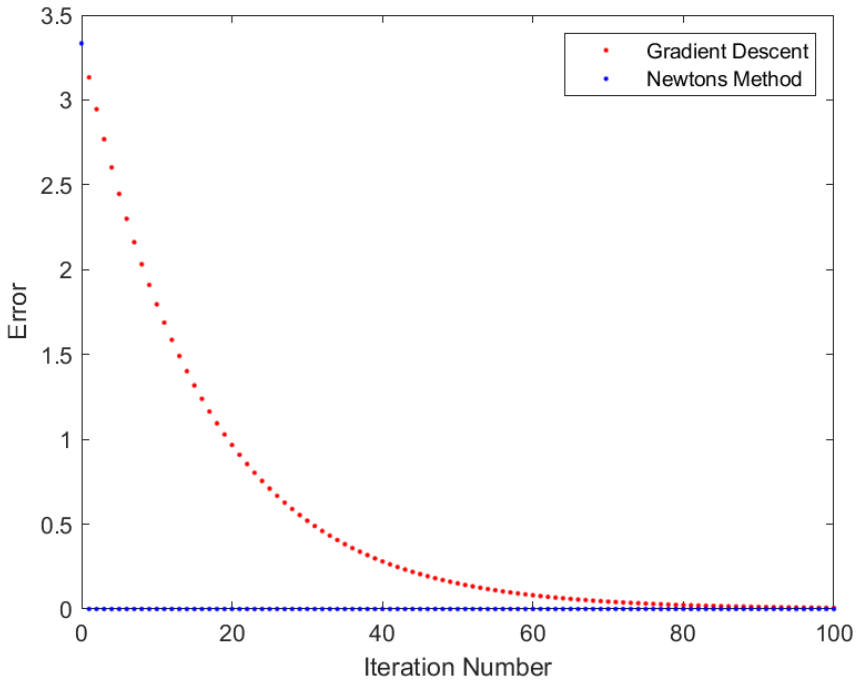
Then, a function `computeSecondGradient(f, xk, dx)` is created, which computes and outputs the second derivative of the function `d2ydx2` utilizing the second derivative equation from lecture slide 36.

Next, a function `newtonsMethod(f, x0, iterations)` is created, which is the exact same as the `gradientDescent` function, except utilizes equation (2) to update the minimizer.

Finally, a main script is written, which first provides the initial conditions `x0 = 4`, `iterations = 100`, and the given function to use is `f` = Eq (3), or `f = @(xk)(3*xk^2 - 4*xk + 6)`. Note that we use `@(xk)` when writing the equation, because at the beginning of the script, `xk` is an undefined variable. Next, a vector representing the number of iterations is creating, going from 0 to the number of iterations given (the "0th" iteration represents the initial given values before any iterations are done.) This will be used for plotting later. The script then calls the `gradientDescent` and `newtonsMethod` functions, and assigns them to respective variables `GD` and `NM`. The global minimum of the given function is then found by hand, and the errors of both optimizers are found such that: `GDerror = abs(GD - 2/3)` and `NMerror = abs(NM - 2/3)`. Finally, a plot with an appropriate legend and axis-labels is created that plots both of the optimization errors vs. their respective iterations.

1.3 Calculations and Results

When the main script is executed, the following figure is created:



1.4 Discussion

This problem was very straightforward, and there were many things that I learned. The main things that I learned were how to use the `function` function and how to understand function handles. Furthermore, I was able to learn about optimization and about two different optimizers: Gradient descent and Newton's method. If we observe the graph above, we can see that Newton's method converges far faster than the gradient descent method. In fact, Newton's method appears to converge after a single iteration. It can therefor be concluded that the speed of convergence for Newton's method is much higher, and that this is most likely also true for other second-order optimizers.

2 Simulating Dynamics

2.1 Introduction

The goal in this problem was to simulate different first order systems with different parameters. More specifically, we will be utilizing the following equation, which describes the dynamics of a first order system with unit step input:

$$\tau\dot{x} + x = k, \tag{4}$$

where *x* is the dependent variable, *τ* is a parameter known as the time constant, and *k* is another parameter called the DC gain. The first goal of this problem was to rewrite equation (4) into the form of $\dot{x} = f(x)$, which should then be used to create a function called `simulateAndPlot`. This function should solve the rearranged form of equation (4) using the `ode45` function and then plot the x-values vs. the time values. Finally, a main script should be created which calls the simulateAndPlot function to simulate and plot equation (4) for four different sets of parameters as seen below:

Trial	*τ*	*k*
1	2.0	0.0
2	1.0	-1.0
3	4.0	2.0
4	5.0	3.0

For all four trials, the final time value should be set to 10, the initial x value should be set to 1, and the range of all the plots should be [-1, 3].

2.2 Models and Methods

The first step of this problem is to rewrite equation (4) into the form of $\dot{x} = f(x)$. Then, a function `simulateAndPlot(f, tf, x0)` is created, where `f` is a function handle, `tf` is the final simulation time, and `x0` is the initial value of x. Also, for this problem, it is assumed that the initial value for time is 0. The function first solves the rearranged form of equation (4) using the `ode45` function as follows:

```
[t,x] = ode45(@(t,x) f(x), [0 tf], x0);
```

Then, the function creates a plot which graphs x-values vs. t-values.

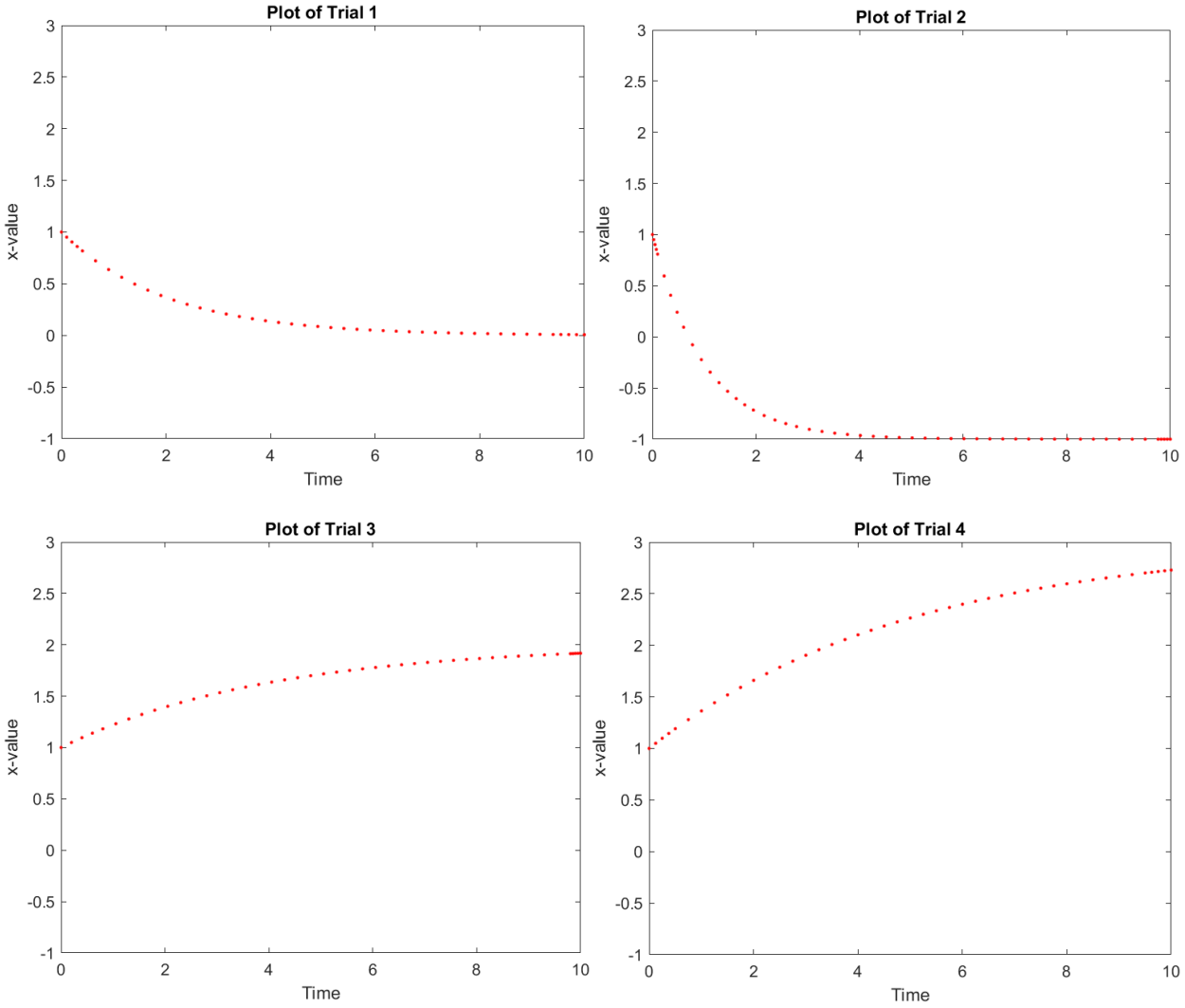
Next, a main script is created, which first assigns the given values to their respective variables: `x0 = 1` and `tf = 10`. The script then uses the `simulateAndPlot` function four separate times, along with equation (4), to generate one plot for each trial. For example, the code for simulating and plotting Trial 1 is seen below:

```
f = @(x)((0 - x)/2); % Equation (4) - appropriate τ and k values for
                     % trial 1 have been substituted in
figure(1);
simulateAndPlot(f, tf, x0)
title('Plot of Trial 1')
```

The script then repeats this exact process for trials 2, 3, and 4, creating a new figure and updating the equation for `f` each time (so for trial 2, the code would say `figure(2);` instead of `figure(1)`).

2.3 Calculations and Results

When the main script is executed, the following figures are created:



2.4 Discussion

This problem was also quite straightforward, and did not cover any new concepts. However, I did learn about the `ode45` function, which can be used to solve a generic first-order differential equation. If we look at the plots above, we notice that the first two plots have a negative slope throughout the graph, while the last two plots have a positive slope throughout. Between the first and second trial, the second trial had smaller values for both tau and k. Additionally, if we look at the table and graphs of all the trials, it can be observed that as tau and k increase, so does the slope and the range of x-values. Furthermore, for all graphs, the absolute value of the slope decreases with time, and the graphs appear to converge to/approach different values. These values that the graphs approach align with the values of k for each of the trials. Additionally, it appears that a smaller value for tau causes the graph to approach k quicker.

3 Linear Regression

3.1 Introduction

The goal in this problem is to apply linear regression to/train a linear model with the form:

$$y = \alpha + \beta x \tag{5}$$

where α is the y-intercept and β is the slope. In order to find the values for α and β that minimize the mean-squared error (this is the objective of linear regression), we can utilize the following explicit solution:

$$\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x} \tag{6}$$

$$\hat{\beta} = \frac{Cov(x,y)}{Var(x)} \tag{7}$$

where $\bar{x}$ is the mean of x, $\bar{y}$ is the mean of y, $Cov(x, y)$ is the covariance between x and y, and $Var(x)$ is the variance of x.

Covariance is defined by the following equation:

$$Cov(x, y) = \sum_{i=1}^{N} \frac{(x_i - \bar{x})(yi - \bar{y})}{N} \tag{8}$$

Additionally,

$$Var(x) = Cov(x, x) \tag{9}$$

In this problem, a function to calculate covariance between two sample sets should be created (without using MATLAB's built-in functions for covariance or variance). Then, another function should be created that calculates the optimal values of $\hat{\alpha}$ and $\hat{\beta}$ from an input dataset. Next, the linear regression function should be applied to the given dataset `hw5_p4_dataset.mat`, followed by scatter plotting the dataset with the linear regression model on top. The equation for the line of best fit and the value of r^2 should be included on the figure, with r^2 being defined as:

$$r^2 = 1 - \frac{SS_{residual}}{SS_{total}} \tag{10}$$

$$SS_{total} = \sum_{i=1}^{N} (y_i - \bar{y})^2 \tag{11}$$

$$SS_{residual} = \sum_{i=1}^{N} \left(y_i - \hat{\alpha} - \hat{\beta} x_i\right)^2 \tag{12}$$

3.2 Models and Methods

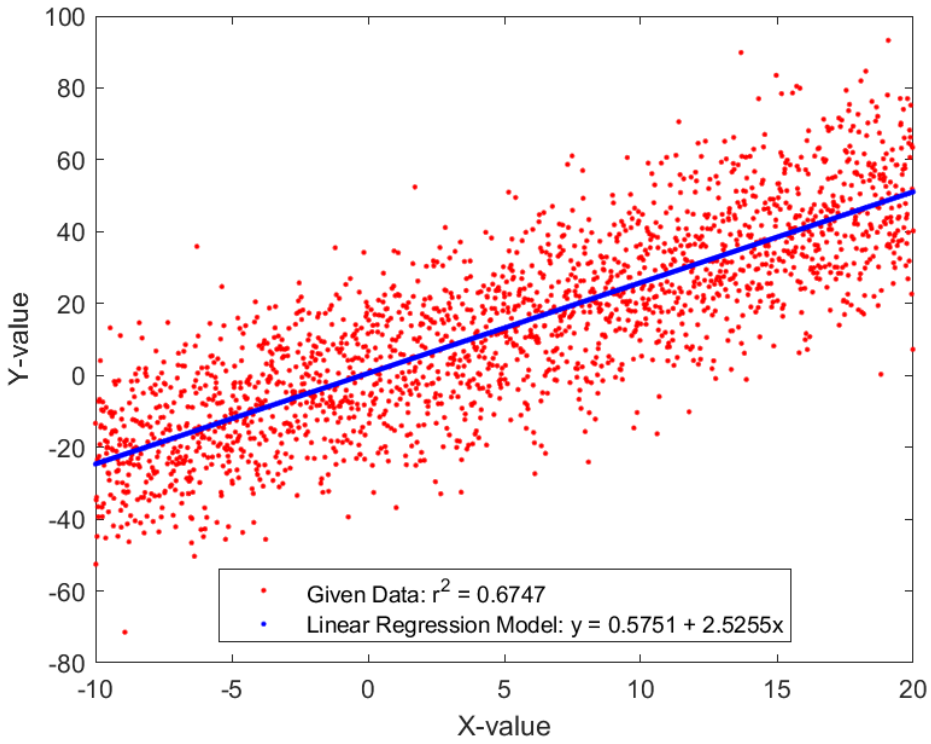
First, a function `calculateCovariance(x, y)` is created, with x and y being values from a dataset. This function first finds the means of the x-values and y-values of the sample set using the `mean` function, and assigns them to variables `xmean` and `ymean`, respectively. Next, equation (8) is utilized to find the covariance, which is the output of the function. Note that for the value of N in equation (8), we can simply use `length(x)` or `length(y)`, since they are both equal to N.

Next, another function `linearRegression(x, y)` is created to calculate the optimal values of $\hat{\alpha}$ and $\hat{\beta}$ from the given dataset. This function then finds outputs a value of $\hat{\beta}$ using equation (7) and the previously created function for covariance. Additionally, equation (9) is used to find the variance. Next, the function finds and outputs a value of $\hat{\alpha}$ using equation (6). It is necessary to find $\hat{\alpha}$ after $\hat{\beta}$, since $\hat{\beta}$ is used in equation (6).

Finally, a main script is created, which first loads the given dataset using the `load` function. Then, the linear regression function is applied to the dataset, and the values for $\hat{\alpha}$ and $\hat{\beta}$ are stored in variables `alpha` and `beta`, respectively. Next, we create our linear regression model by making a vector for the x-values and a vector for the y-values. The x-vector is just a vector starting at the minimum x-value of our dataset and ending at the maximum x-value of our dataset, with a sufficiently small step size (1/100 was chosen in this problem). The x-vector is then plugged into equation (5) to create a vector for the y-values. These will be used to plot the line of best-fit later. Next, the "R-squared" value is found by using and assigning variables to equations (11) and (12), and then plugging those variables into equation (10). Lastly, a figure is created which scatter plots the given data set and plots the line of best fit using the obtained x-vector and y-vector. Additionally, the equation for the line of best fit, along with the value of "R-squared," are indicated in the legend on the figure.

3.3 Calculations and Results

When the main script is executed, the following figure is created:



3.4 Discussion

This problem was also quite straightforward, and no new functions or concepts were used. There is not much to discuss about this problem, however if we look at the “R-squared” value, we notice that it is equal to 0.6747. The R-squared value will be between 0 and 1, and a higher R-squared value indicates that the linear model fits the data better. Therefor, we can conclude that the linear model fits the data relatively well.