Benjamin Asdell
CEE/MAE M20

UID: 005114992 **Homework 5** October 15, 2020

1 Optimization

1. Introduction

   The objective of this problem is to create a script that minimizes a given convex function using two approaches: gradient descent and Newton's method. The program uses first and second numerical derivatives of the function to slowly converge to the global minimum. It then graphs the error between each of these minimizing x-values and the true global minimizer against the iterations completed.

2. Models and Methods

   This program does not take any user inputs, and the function being optimized is hard coded into the script. However, the given function can be interchanged with any convex function and should still operate as intended.
   The basis of the gradient descent method is the following equation:

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

   This method takes an initial guess x_0 as the minimizer and incrementally updates this value using the first derivative of the function in order to move the x-value closer to the true global minimum x-value. This particular implementation of gradient descent makes use of the three point central difference formula for estimating the numerical first derivative at a point:

$$f'(x) \approx \frac{f(x+dx) - f(x-dx)}{2dx} \tag{2}$$

   In this equation, the γ value corresponds to the step size of incremental change, or the learning rate if using this approach to optimize a neural network.
   The second approach, Newton's method, makes uses of a slightly altered formula that takes into the account the effect that the second derivative has on the slope of the function at each step. This changes Eq. 1 to be:

$$x_{k+1} = x_k - \frac{f'(x)}{f''(x)} \tag{3}$$

   Note that the γ value used in Eq. 2 is not present in Eq. 4. In order to estimate the value of the numerical second derivative at the current minimizer, the central difference formula for a second derivative is used:

$$f''(x) \approx \frac{f(x+dx) - 2f(x) + f(x-dx)}{dx^2} \tag{4}$$

   Finally, the program calculates the error between each guess and the actual global minimizer. To accomplish this, the global minimum is calculated by hand since the given function is a relatively simple quadratic function. The global minimum was found to occur at $x = 2/3$ and therefore the following equation is used to calculate the error at each step.

$$error_k = |x(k) - \frac{2}{3}| \tag{5}$$

These errors values are plotted against the iteration number k to generate the plot. The plot is a visual demonstration of how fast each method converges its guesses to the true global minimizer, which is represented by the error reaching a zero value. Each step in this program (Eq.1 - 4) is contained within separate function files in order to facilitate and compartmentalize the code. The functions for gradient descent and Newton's method both call the function for Eq. 2, and Newton's method also utilizes the function for Eq. 3. In the main script, only these larger functions are called, the $f(x)$ is declared, and the plotting is completed.

3. Calculations and Results

With the initial values $x_0 = 4$ and 100 iterations, the following plot is generated:

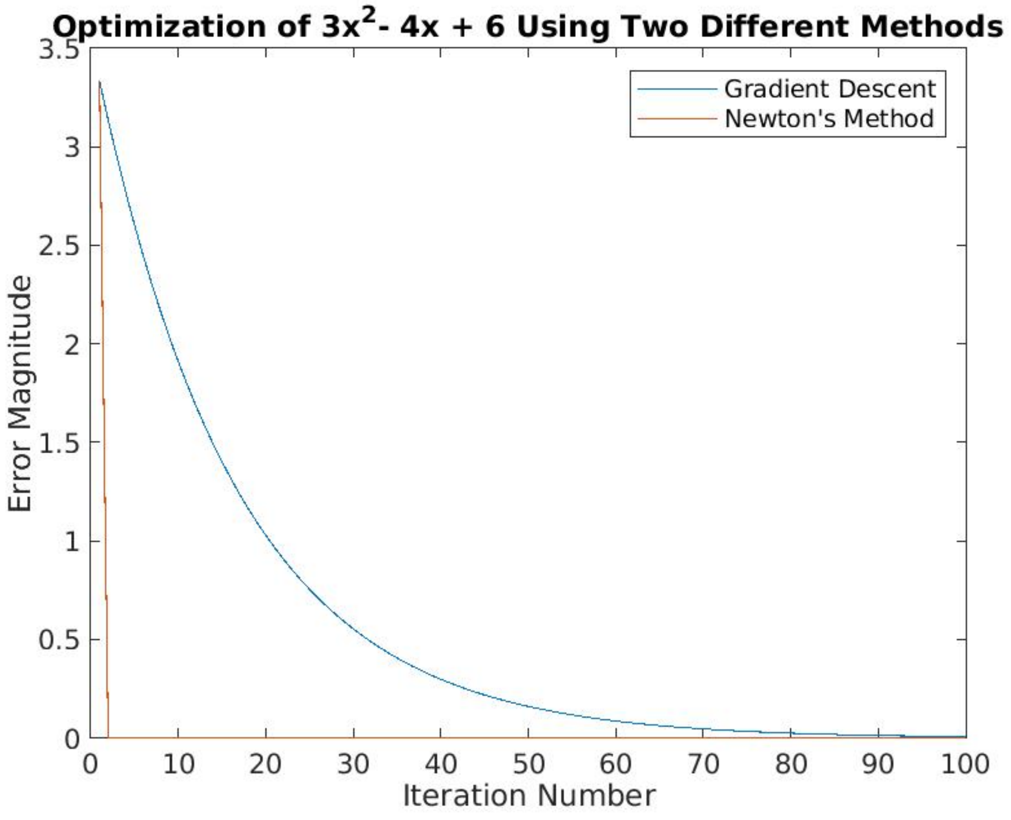


Figure 1: Note that the first data point graphed for both methods occurs at iteration number one, since the first iteration corresponds to guessing x_0 as the optimal minimizer. Visibly, Newton's method is incredibly faster to optimize this function than gradient descent, reaching the true minimizer in two iterations rather than the 100 iterations that gradient descent required.

4. Discussion

   This program was very intriguing to solve, especially because of its implication with regards to machine learning and neural networks. It also showcases MATLAB's ability to efficiently work through relatively complex or tedious mathematical processes like optimization. Calculating and estimating derivatives over dozens of iterations would be extremely time-consuming and error-prone by hand. This program greatly facilitates optimization using the two methods.
   Analyzing the generated plot has some interesting insights into the optimization of this given function. The gradient descent method takes all 100 iterations to converge its minimizer guess to the global minimizer. It is also notable that the 'progress' made between each iteration slows down considerably as the method gets closer to the true minimizer. These observations are reasonable because linear approximations

of the slope of the function will be less and less accurate (relative to the value of the actual derivative) as the function approaches its minimum. Linear estimates simply cannot mimic the curvature of the function near extrema.
On the other hand, Newton's method converges to the true minimizer in two total iterations. This means its second guess after the initial test value x_0 was the true value. This makes complete mathematical sense because this method utilizes a quadratic approximation to estimate the next x-value guess. Since the equation being optimized is a quadratic function, the estimate fits the curvature of the function perfectly and the function can be optimized in a single iteration after the initial guess. This means that Newton's method is most likely the best optimizing method for second degree polynomials. More generally, I believe this experiment demonstrates that the highest order of derivative used in the optimizing algorithm should be as high as the degree of the polynomial being optimized.

2 Simulating Dynamics

1. Introduction

   The goal of this problem is to simulate and plot the dynamics of an abstract system when given the first order differential equation corresponding to those dynamics. This program depends on MATLAB's 'ode45' function and manipulation of the first order differential equation to graph each trial.

2. Models and Methods

   The differential equation is given in the assignment in dot notation; here it is rewritten:

   $$\tau \frac{dx}{dt} + x = k \qquad (1)$$

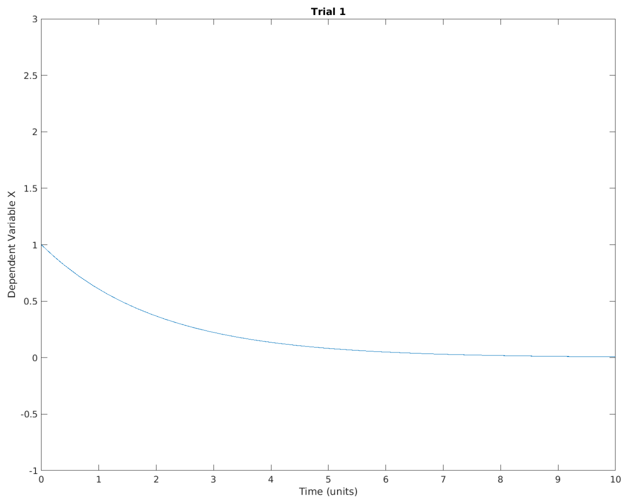
   The equation is then solved for $\frac{dx}{dt}$ in order for MATLAB's ode45 to interpret it.

   $$\frac{dx}{dt} = \frac{k - x}{\tau} \qquad (2)$$

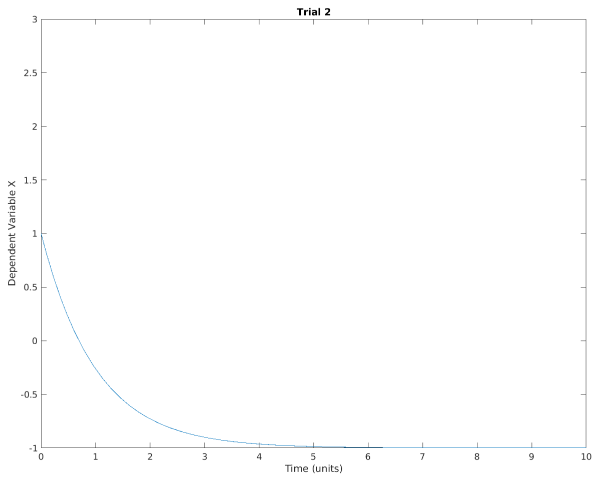
   Since ode45 is designed to work with two variable functions, the t variable is simply ignored when coding the behavior of the simulateAndPlot.m function that makes up the majority of the program's functionality. In this program, the specific differential equations with τ and k values are hard coded for each trial. The program then utilizes MATLAB's plotting syntax to graph each plot on a separate figure.
   As mentioned, the main functionality of this program is contained within the function simulateAndPlot.m. The main script calls this function four times, once for each of the trials.

3. Calculations and Results

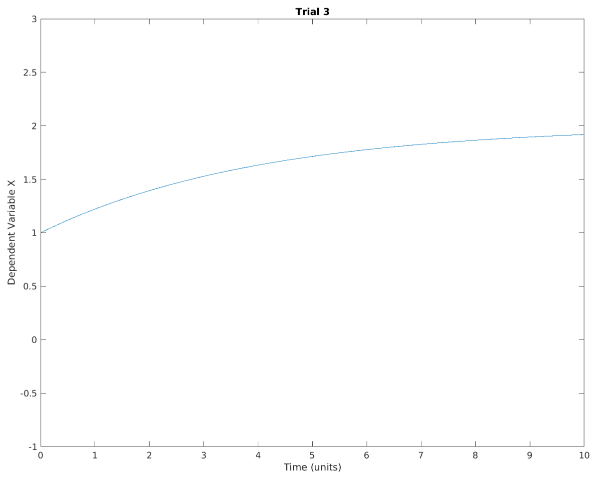
   The program displays no outputs to the command window when executed, but instead generates these four plots with the constants $x_0 = 1.0$ and $t_f = 10$.



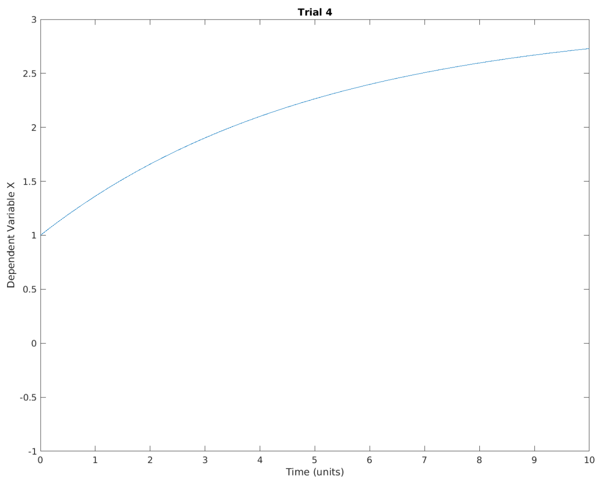
(a)



(b)



(c)



(d)

Figure 2: All four plots generated by this program, going from Trial 1 (top left) to Trial 4 (bottom right). Notice that all four plots have a y-intercept at 1, which is is the initial x_0 value.

4. Discussion

   This program is a display of MATLAB's built-in ode45 function and its ability to handle first order differential equations. Having not taken a differential equation class, the concepts and strategies related to that field of mathematics is alien to me, yet this program provides insights into how they work and allows me to view their actual plots without explicitly solving them myself. The plots all show various curves that arise from solving the differential equation in Eq. 2 using the specific k and τ values for each trial. This program should work for analyzing any first order system dynamics, although it is not as versatile considering the specific equation used in each trial is hard coded. The most common example of such a system would be an RC circuit. Therefore, this program is probably very useful for simple experiments relating to first order systems like an RC circuit, and is most likely pertinent to my engineering major. This program greatly facilitates understanding of such a system because it takes a differential equation that might not have any physical significance to a student and generates a plot that can be visually analyzed.

3 Linear Regression

1. Introduction

   The purpose of this problem is use MATLAB to analyze and interpret a large data set to determine correlation and a line of best fit to the data. The program first calculates variance between different data sets and finds the optimal slope and y-intercept for a line of best fit. It plots the data and the calculated best fit line, then uses a new variance calculation to determine the coefficient of correlation.

2. Models and Methods

   The basis of this problem is to find an equation for a line of best fit (Eq. 1) that minimizes the mean-squared error defined in Eq. 2:

$$y = \alpha + \beta x \tag{1}$$

$$J(x,y) = \sum_{i=1}^{N} (y_i - \alpha - \beta x_i)^2 \tag{2}$$

   The program calculates β using variance formulas and then solves for α algebraically:

$$\beta = \frac{Cov(x,y)}{Var(x)} \tag{3}$$

$$\alpha = \overline{y} - \beta\overline{x} \tag{4}$$

   In order to complete this step, the formula for the covariance between two data sets needs to be used. Note that $Var(x) = Cov(x,x)$

$$Cov(x,y) = \sum_{i=1}^{N} \frac{(y-\overline{y})(x-\overline{x})}{N} \tag{5}$$

   The final equation necessary for this problem is related to calculating the coefficient of determination, also called the correlation coefficient. This r^2 value is bounded between [0,1] and the closer the value is to 1, the more correlated the data can be concluded to be.

$$r^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \alpha - \beta x_i)^2}{\sum_{i=1}^{N}(y-\overline{y})^2} \tag{6}$$

   Like in Problem 2, the functionality of this program is broken down using separate function files, one to implement Eq. 5 and another to implement the entirety of the calculation for α and β. The main script only calls the linearRegression.m function and graphs the scatter plot and best fit line.

3. Calculations and Results

   Using the data set given by the assignment, the following scatter plot is generated by the program in addition to displaying its line of best fit and correlation coefficient.

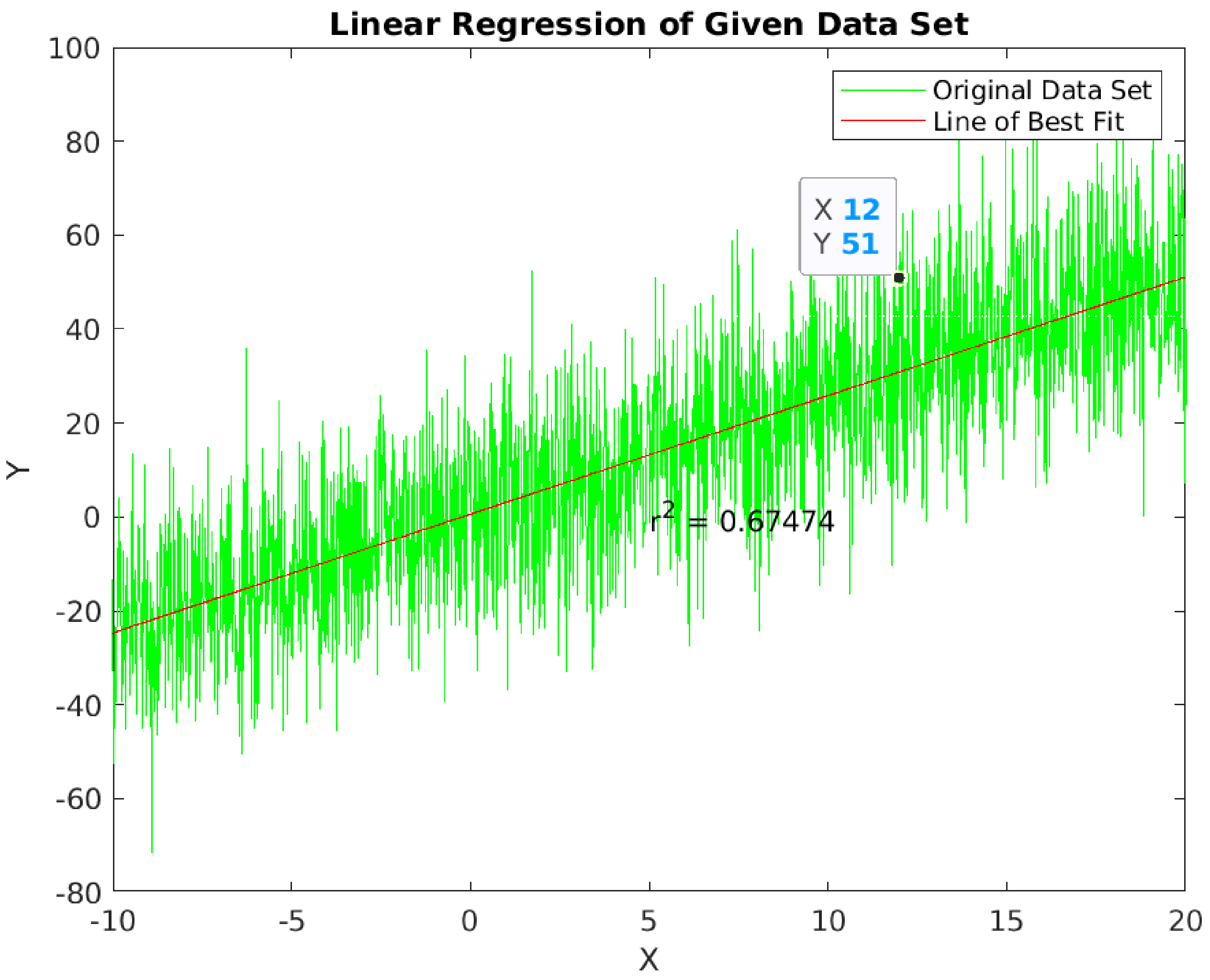


Figure 3: The scatter plot visibly has a positive trend, despite significant noise in the data. From observation, the line of best fit appears to correctly fit the trend of the data. The r^2 value is not extremely close to 1, which is reasonable considering the deviance from the line that the scatter plot shows. However, the correlation coefficient is still high enough to confirm the positive correlation between X and Y.

4. Discussion

   This program showcases MATLAB's efficient functionality in handling and interpreting large data sets, since there are 2000 individual points plotted and used for the various aforementioned calculations. The given data set in this case is definitely positively correlated, as visibly seen by the scatter plot and confirmed by the best fit line. While the $r^2 = 0.67474$ is not relatively close to a perfect correlation of $r^2 = 1$, the correlation is high enough to confirm this positive correlation assessment. An interesting point to notice about this program is that it is naturally versatile; it should be able to function normally for any valid data set of arbitrary size even though it was not inherently coded to be flexible. Simply loading in a data set with arrays x and y should produce useful results. This program is another great example of how MATLAB expedites long, tedious statistical and graphical analysis in relatively simple programs.