**BME 313L: Introduction to Numerical Methods in Biomedical Engineering**

**Lab Report**

**Lab #8: Linear/nonlinear regressions and least-squares**

**Last name, First name: Liu, Vincent**

**EID: VL5649**

**Lab Section: 14035 (Tuesday 9:30-12:30)**

**Problem 1: Least Squares**

An investigator has reported the data tabulated below for an experiment to determine the growth rate of bacteria k (per d) as a function of oxygen concentration c (mg/L). It is known that such data can be modeled by the following equation:



where  and  are parameters. In an experiment, the following values of ‘k’ and ‘c’ are observed:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| c | 0.5 | 0.8 | 1.5 | 2.5 | 4 |
| k | 1.1 | 2.5 | 5.3 | 7.6 | 8.9 |

Answer the following questions: (100 word minimum for each question, 50 word minimum for discussing what you learned, what was reinforced)

1. Find a transformation that linearizes this equation
2. Implement the normal equations of least-squares fit in MATLAB to estimate the parameters  and . The usage of any pre-built regression functions is not permitted.
3. In the same figure, plot the observed growth rates (k) and predicted growth rates () against the oxygen concentration values of c = 0.5, 0.8, 1.5, 2.5 and 4.

(The following is your answer)

**MATLAB code:**

c = [.5;.8;1.5;2.5;4]; %observed c values

k = [1.1;2.5;5.3;7.6;8.9]; %observed k values

x = 1./c.^2; %linearization for x

y = 1 ./ k; %linearization for x

n = length(x); %number of terms

sc = sum(x); sk = sum(y); %sums up x and y's

sc2 = sum(x.\*x); sck = sum(x.\*y); sk2 = sum(y.\*y); %'squares'

a(1) = (n\*sck-sc\*sk)/(n\*sc2-sc^2); %calculations using least squares

a(2) = sk/n-a(1)\*sc/n;

kmax = 1/a(2); %calculates kmax

cs = a(1)/a(2); %calculates cs

kp = kmax.\*c.^2./(cs+c.^2); %predicted values

fprintf('The estimated values of cs and kmax are\n') %formatting

fprintf('%f and %f, respectively\n',cs,kmax)

plot(c,k,'o',c,kp,'.-')

legend('Observed Growth Rates','Predicted Growth Rates')

xlabel('Oxygen Concentration (mg/L)')

ylabel('Growth Rate (per d)')

title('Growth Rate of Bacteria as a Function of Oxygen Concentraion')

grid on

**MATLAB function:**

The purpose of this script was to perform a regression on a set of observed data points that follow a given equation, in order to solve for the coefficients. By solving for the coefficients, we can then generate predicted values that best approximate our data. To do so, we have to first linearize our measured values so that we can perform a linear regression. We can then use the values that we found for the coefficients in order to solve for the predicted values and then plot this alongside our measured values to see our accurate our regression model is.

c = [.5;.8;1.5;2.5;4]; %observed c values

k = [1.1;2.5;5.3;7.6;8.9]; %observed k values

These first 2 lines of code are the observed growth rate and oxygen concentrations (k and c) that were measured as part of the experiment.

x = 1./c.^2; %linearization for x

y = 1 ./ k; %linearization for x

These 2 lines of code are the linearization of the variables for the given equation. This was found by solving for an equation of the form y=a(1)x+a(2).

n = length(x); %number of terms

This line of code finds the number of terms in our data. This could have been accomplished by using either length(c), length(k), or length(y) as well because they are all of the same length. This value is needed later on when calculating our constants.

sc = sum(x); sk = sum(y); %sums up x and y's

sc2 = sum(x.\*x); sck = sum(x.\*y); sk2 = sum(y.\*y); %'squares'

These 2 lines of code sum up the x and y values and calculates the squares of the variables. These values will be used in conjunction with the length that we found earlier in our calculations.

a(1) = (n\*sck-sc\*sk)/(n\*sc2-sc^2); %calculations using least squares

a(2) = sk/n-a(1)\*sc/n;

These 2 lines of code are a0 and a1 where the sum of the squares of the residuals is minimal. These equations were derived by rearranging the “normal equations” in a form that could solve for the coefficients, a(1) and a(2).

kmax = 1/a(2); %calculates kmax

cs = a(1)/a(2); %calculates cs

These 2 lines of code calculate the variables kmax and cs from the original equation that we were asked to solve for. These relationships were found by linearizing the function and setting the coefficients from the original equation equal to the coefficients of a standard linear equation, a(1) and a(2).

kp = kmax.\*c.^2./(cs+c.^2); %predicted values

This lines of code generates potential k values using the original equation that we were given, whose coefficients we have now solved for, giving us our “predicted” growth rates.

fprintf('The estimated values of cs and kmax are\n') %formatting

fprintf('%f and %f, respectively\n',cs,kmax)

These 2 lines of code print out our estimates for cs and kmax into the command window so that the user can easily interpret the results.

plot(c,k,'o',c,kp,'.-')

This line of code plots our measured concentration and growth rates, c and k, as o’s on the plot. Our predicted values at the given concentrations are then plotted alongside our measured values to show how accurate our regression was.

legend('Observed Growth Rates','Predicted Growth Rates')

xlabel('Oxygen Concentration (mg/L)')

ylabel('Growth Rate (per d)')

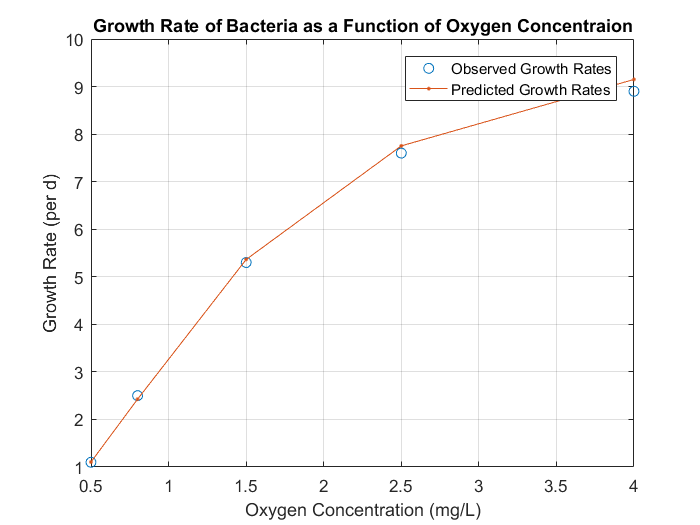
title('Growth Rate of Bacteria as a Function of Oxygen Concentraion')

grid on

These last 5 lines of code make it so that the user can easily interpret and understand all of the aspects of the graph.

**Results:**

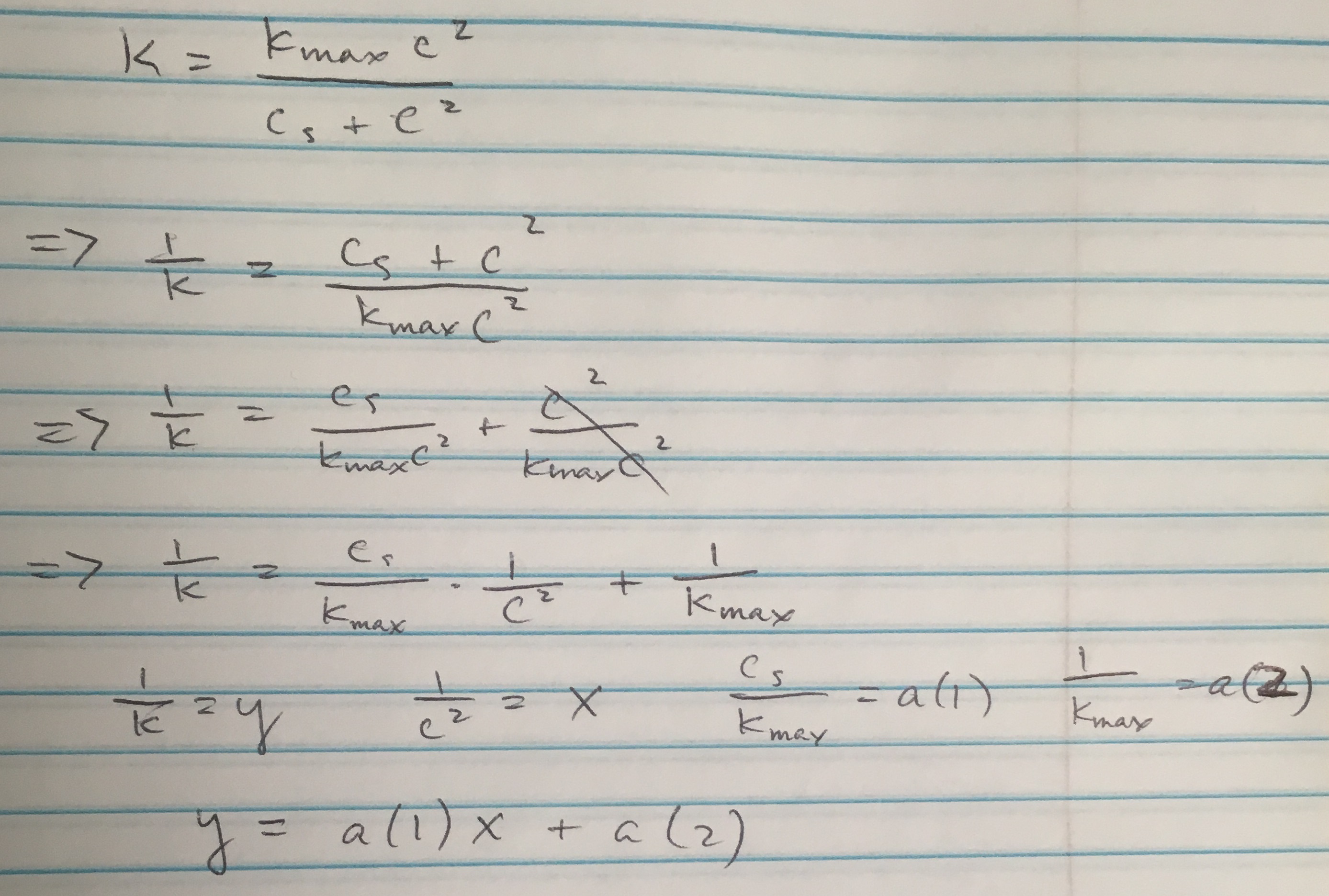
****



**Discussion:**

As shown by the results, we can perform a linear regression on a nonlinear relationship by first linearizing the coefficients so that we have a linear relationship. After solving for the coefficients for the linear relationship, we can then work backwards to solve for the constants and fit a curve that relates our variables, oxygen concentration and growth rate. From this, we determined that the parameters of the relation, Cs and Kmax, are 2.089730 and 10.344931, respectively, for this experiment. As seen in the plot of our regression model against the observed values, our regression model fits very well with each of our predicted values coming very close and no large erroneous differences.

From this problem, we learned how to linearize a set of data, given a function that relates the data, to solve for a nonlinear relationship between values. We reviewed how to perform mathematical operations with arrays and matrices. Furthermore, we reviewed how to print values using MATLAB’s fprintf function, into the command window. Lastly, we reviewed how to plot and then format the plot using arrays and built in MATLAB functions in order to visually represent our data.



Linearization of the given function

**Problem 2: General Linear Least Squares**

Environmental scientists and engineers dealing with the impacts of acid rain must determine the value of the ion product of water as a function of temperature. Scientists have suggested the following equation to model this relationship:



where  = absolute temperature (in K), and a, b, c, and d are parameters. The following data is observed:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1.164x10-15 | 2.950x10-15 | 6.846x10-15 | 1.467x10-14 | 2.929x10-14 |
|  | 1 | 10 | 20 | 30 | 40 |

Answer the following questions: (100 word minimum for each question, 50 word minimum for discussing what you learned, what was reinforced)

1. Is this equation equivalent to a general linear least squares model? If yes, what are the basis functions of this model?
2. Implement an approach in MATLAB that uses the normal equations to estimate the parameters a, b, c and d. The usage of any pre-built regression functions is not permitted.
3. Generate a plot of predicted ion products of water versus temperature (for = 1, 10, 20, 30 and 40)
4. Generate a plot of squared residuals ( versus temperature (for = 1, 10, 20, 30 and 40)

(The following is your answer)

**MATLAB code:**

Kw = [1.164e-15;2.950e-15;6.846e-15;1.467e-14;2.929e-14]; %Observed Ion Product, Kw

Ta = [1;10;20;30;40]; %Observed Temperatures, Ta

y = -log(Kw); %linearizes--sort of

z = [Ta.^-1 log(Ta) Ta ones(size(Ta))]; %basis functions

a = (z'\*z)\(z'\*y); %general linear least squares regression

Kwp = (exp(a(1)./Ta+a(2)\*log(Ta)+a(3).\*Ta+a(4))).^-1; %predicted values of ion product, Kw

plot(Ta,Kw,'o',Ta,Kwp,'.-') %1st plot

xlabel('Absolute Temperature (K)') %1st plot formatting

ylabel('K\_w')

title('Ion Product of Water as a Function of Temperature')

Sr = (Kw-Kwp).^2; %residual squared

figure %new figure

plot(Ta,Sr) %2nd plot

xlabel('Absolute Temperature (K)') %2nd plot formatting

ylabel('Squared Residual')

title('Squared Residual with respect to Temperature')

**MATLAB function:**

The purpose of this function was to perform regression using the general linear least squares method, in order to estimate values for the parameters a, b, c, and d in the equation given to model the relationship between the ion product of water and temperature. To do so, we can solve for each of the coefficients using a matrix of the basis functions and then solve for the coefficients that satisfy for the measured Kws at each point. This model can then be plotted alongside our measured data to see how well our values fit. We can also plot a separate set of data calculating how far off each of the values is at every point.

Kw = [1.164e-15;2.950e-15;6.846e-15;1.467e-14;2.929e-14]; %Observed Ion Product, Kw

Ta = [1;10;20;30;40]; %Observed Temperatures, Ta

These first 2 lines of code are the observed ion product and the observed temperatures that correspond to each of the values, for which we are to perform a regression for as part of the problem.

y = -log(Kw); %linearizes--sort of

This line of code ‘linearizes’ Kw to the value y, in a way. In doing so, we only have to worry about the value, y, instead of -log(Kw).

z = [Ta.^-1 log(Ta) Ta ones(size(Ta))]; %basis functions

This line of code generates a matrix in which each of the columns are values that correspond to the basis functions of the given equation.

a = (z'\*z)\(z'\*y); %general linear least squares regression

This line of code solves for the coefficients, a, that correspond to each of the basis functions where [Z]{a}={y}. Because z is not a square matrix, z transpose is multiplied by both sides, which can then be solved for the coefficients.

Kwp = (exp(a(1)./Ta+a(2)\*log(Ta)+a(3).\*Ta+a(4))).^-1; %predicted values of ion product, Kw

This line of code calculates the predicted values of the ion concentration, Kw, using the coefficients and original values of absolute temperature from earlier. This is done by solving the equation given to us in terms of the variable that we want to solve for, Kw.

plot(Ta,Kw,'o',Ta,Kwp,'.-') %1st plot

This line of code first plots the observed values of Kw to the corresponding temperature values as o’s. A line connecting the predicted values of Kw using our regression model is then plotted alongside it with points at each of the Ta values that we have a Kw for.

xlabel('Absolute Temperature (K)') %1st plot formatting

ylabel('K\_w')

title('Ion Product of Water as a Function of Temperature')

The first 2/3 lines of code make it so that the axes of the plots can be understood. The 3rd line of code then adds a title describing what the plot is of.

Sr = (Kw-Kwp).^2; %residual squared

This line of code calculates the residual be subtracting our predicted regression value from the observed value. This value is then squared so that we can estimate just how well our regression fits at each point.

figure %new figure

plot(Ta,Sr) %2nd plot

These 2 lines of code make sure that there are 2 separate figures in the script and then plots the second figure. The 2nd line plots the square residual values that we calculated 2 lines before against each of the corresponding temperature values.

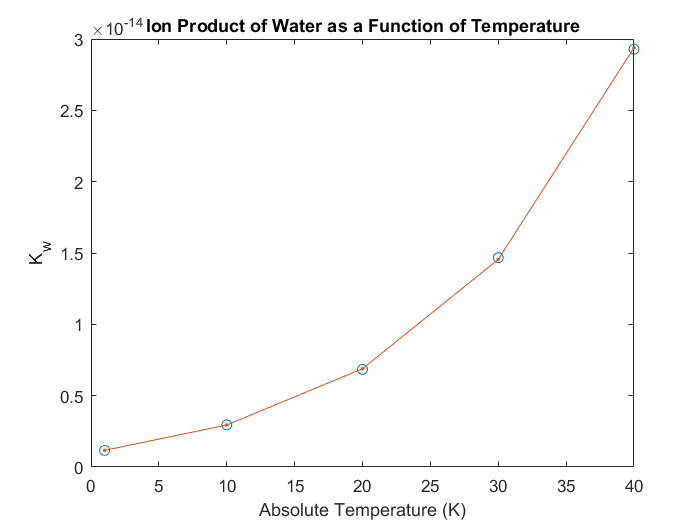
xlabel('Absolute Temperature (K)') %2nd plot formatting

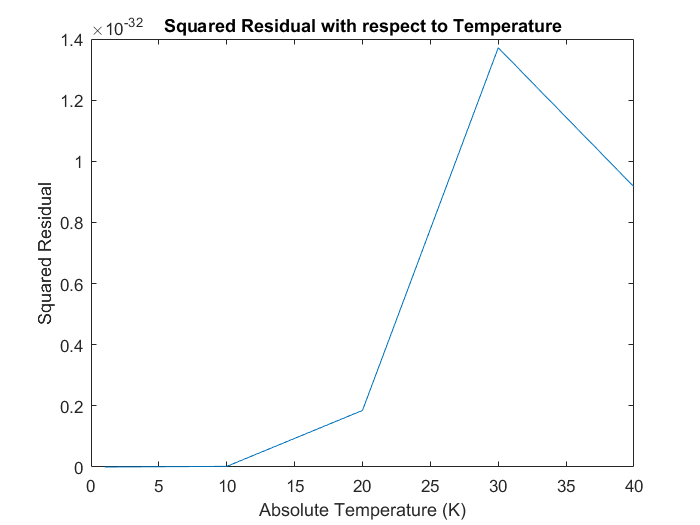
ylabel('Squared Residual')

title('Squared Residual with respect to Temperature')

These last 3 lines of code add labels to the figure so that it can be more easily understood.

**Results:**





**Discussion:**

As shown by the results, we can use the general linear least squares method in order to fit a curve to a set of related data points. Plotting our model against the observed data points, we can see that model fits very well to the data; however, with just the first plot we can’t tell exactly how accurate each of the values are—the second plot does, point by point. As seen in the second plot, the 2 values line up almost exactly, but the square differences begin to increase up until the value for Ta = 30, where the residual squared is the greatest (i.e. greatest difference). However, these squared residual values are all very small, with all falling below 1.4e-32, meaning that our regression model was very accurate. While we could use the general linear least squares method on the equation, it was not quite ‘equivalent’ to the linear least squares model. However, after linearizing the variable on the left side of the equation, Kw as y = -log(Kw), it does follow the model with the basis functions as x^-1, log(x), x, and x^0.

From this problem, we learned how to perform regression, given a function that relates the data, using the general linear least squares method. We reviewed how to perform mathematical operations with arrays and matrices (including left division and element wise operations). Furthermore, we reviewed how to print values using MATLAB’s fprintf function, into the command window. Lastly, we reviewed how to plot and then format the plot using arrays and built in MATLAB functions in order to visually represent our data.

**Problem 3: Nonlinear Regression**

The movement of mammalian cells through an extracellular matrix of tissues is an important topic in cell and tissue engineering. Cell migration is necessary for successful repair of wounds, regeneration of tissues, and repopulation of implanted scaffolds for tissue-engineering therapies. For example, endothelial cells of microcirculatory blood vessel walls need to migrate out into tissue space to form new vascular networks, a process called angiogenesis. White blood cells need to migrate on vascular prosthetic materials to reach and eliminate any bacterial microorganisms, a process reminiscent of acute inflammatory reaction. The population migration of cells can be quantitatively described by the Dunn equation if the cell coordinates and trajectories are obtained through direct visualization by computer-assisted time-lapse digital microscopy:



The mean squared displacement of cells is a nonlinear function of elapsed time, t. The independent parameters of cell movement are root-mean square speed, *S*, and persistence time, *P*. Estimate the parameters *S* (in /min) and *P* (in min) using *fminsearch* given the following values of mean squared displacement of cells  and elapsed time t:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (in ) | .0006 | .0192 | .0433 | .0687 | .0921 | .1169 | .1415 | .1651 | .1899 | .2154 |
| t (in min) | 1 | 11 | 21 | 31 | 41 | 51 | 61 | 71 | 81 | 91 |

Note that *fminsearch* requires as input a function file, which you should submit as part of the solution.

(50 word minimum for discussing what you learned, what was reinforced)

(The following is your answer)

**MATLAB code:**

**Function:**

function f = fSSR\_VL(a,t,d2) %function

dp = 2\*a(1)^2.\*(a(2).\*t-a(2)^2\*(1-exp(-t./a(2)))); %predicted values

f = sum((d2-dp).^2); %summation of squared residuals

**Mainscript:**

d2 = [.0006;.0192;.0433;.0687;.0921;.1169;.1415;.1651;.1899;.2154]; %observed mean squared displacement of cells

t = [1;11;21;31;41;51;61;71;81;91]; %observed elapsed times

a = fminsearch(@fSSR\_VL,[2,2],[],t,d2); %calls fSSR function script into fminsearch

fprintf('The estimated values of S (in um/min) and P (in min) are:\n') %formatting

fprintf('%f and %f, respectively\n',a(1),a(2))

**MATLAB function:**

The purpose of this function is to solve for a least squares fit using nonlinear regression. To do this, we have to write a function that takes the dunn equation and calculates the sum of residual squares of our fit values to the measured values. We can then take advantage of the fminsearch function in order to calculate parameters that minimize the residual squares and then print out these parameters using fprintf.

function f = fSSR\_VL(a,t,d2) %function

This first line of code in the fSSR\_VL function outlines the output, function name, and input parameters to be used in the function.

dp = 2\*a(1)^2.\*(a(2).\*t-a(2)^2\*(1-exp(-t./a(2)))); %predicted values

This line of code calculates potential d squared values, using the equation that was given to us as part of the problem. This line of code is used continually, to generate multiple potential values with different coefficients.

f = sum((d2-dp).^2); %summation of squared residuals

This line of code is the result of the square of the difference between our observed values and our predicted values. These values of f are then passed to the fminsearch function in our mainscript in order to calculate our coefficient values.

d2 = [.0006;.0192;.0433;.0687;.0921;.1169;.1415;.1651;.1899;.2154]; %observed mean squared displacement of cells

t = [1;11;21;31;41;51;61;71;81;91]; %observed elapsed times

These 2 lines of code are the observed mean squared displacements of cells and the corresponding times at which each of the values is observed. These values are given to us as the data we are trying to fit.

a = fminsearch(@fSSR\_VL,[2,2],[],t,d2); %calls fSSR function script into fminsearch

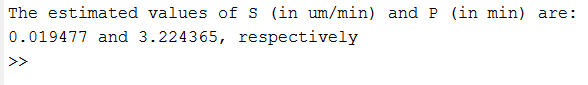
This lines of code calls the fSSR function that I wrote as an anonymous function into the fminsearch function that is built into MATLAB. The initial guesses of S and P, 2 and 2, are written as parameters as well as the 2 sets of observed data, t and d2.

fprintf('The estimated values of S (in um/min) and P (in min) are:\n') %formatting

fprintf('%f and %f, respectively\n',a(1),a(2))

These last 2 lines of code print out the calculated S and P values, using fprintf, into the command window.

**Results:**

****

**Discussion:**

As shown by the results, we can perform a nonlinear regression in order to estimate our coefficients of our least squares fit for the Dunn equation, given a set of data. By using the fminsearch function, we could minimize our squares and obtain the estimated values of S and P as .019477 and 3.224365. Originally, our initial guesses of 1 and 1 yielded a negative value for S; however, it isn’t possible to have a negative speed so our initial guesses were updated to find another local max/min to give us the values we have now.

From this problem, we learned how to use nonlinear regression to find a least squares fit. We learned how to implement MATLAB’s fprintf along with reviewing how to pass function scripts as anonymous functions. Furthermore, we reviewed how to perform mathematical operations with arrays and matrices. Lastly, we reviewed how to print values using MATLAB’s fprintf function, into the command window.