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

**Lab Report**

**Lab #7 Chapter 13: Eigenvalues**

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

**EID: VL5649**

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

**Problem 1. From textbook Problem 13.2 (Power Method)**

Use the power method to determine the highest eigenvalue and corresponding eigenvector for

**Things to discuss**

(1) Describe the details of the algorithm of power method and how it was implemented in this problem.

(2) The smallest eigenvalue and its associated eigenvector can be determined by applying the power method to the matrix inverse of [A] (textbook p.312), why?

(3) The example 13.3 in the textbook has a high approximate relative percentage at the fourth iteration, but it does converge and stabilize on the largest eigenvalue. Why don’t we observe the similar fluctuation in this problem?

**MATLAB code:**

clear all; close all; clc; %resets MATLAB

A = [2 8 10; 8 4 5; 10 5 7]; %coefficient matrix

%[x,y] = powereig\_VL(A,5,50); %calls powereig function

es = .05; %error

n = length(A);

evect = ones(n,1); %forms initial 'guess' for eigenvector

eval = 1; %initializes eigenvalue

ea = 1; %initializes error

while ea >= es

evalold = eval; %stores last eigenvalue

evect = A\*evect; %calculated eigenvector

eval = max(abs(evect)); %updates eigenvalue

evect = evect./eval; %new eigenvector

if eval~=0

ea = abs((eval-evalold)/eval); %calculates error

end

end

fprintf('The highest eigenvalue for the matrix is:\n%f\n',eval) %displays results

fprintf('The corresponding eigenvector is:\n')

fprintf('%f\n',evect)

**MATLAB function:**

The purpose of this function was to solve for an eigenvalue of the matrix (the largest) using the power method. To do so, we had to outline all of the values specified in the problem. We could then iterate the calculations of the power method to get the eigenvalues and eigenvectors of the matrix and print it in the command window.

clear all; close all; clc; %resets MATLAB

This first line of code ensures that MATLAB is completely cleared and that previous scripts do not affect how this script runs.

A = [2 8 10; 8 4 5; 10 5 7]; %coefficient matrix

This line of code is the matrix for which we are solving eigenvalues for. This matrix is given to us as part of the problem.

%[x,y] = powereig\_VL(A,5,50); %calls powereig function

This line of code was originally used to call the powereig function so that I had baseline values to compare to; however, I wanted to write the function as part of the script myself so that I could understand how it worked, so the line was commented out.

es = .05; %error

This line of code designates the error criterion that the loop for calculations stops at once it drops below. Since we weren’t given a value, I went with the value that was given to us in the sample problem.

n = length(A);

evect = ones(n,1); %forms initial 'guess' for eigenvector

These 2 lines of code generate our initial guess for the eigenvector of the problem (which is a vector of the correct length of all 1’s). From this initial guess we can then

eval = 1; %initializes eigenvalue

This line of code initializes the eigenvalue so that there is a value that we can use as the ‘first’ value to calculate the initial error.

ea = 1; %initializes error

This line of code initializes the error value so that the script does not interpret the initial error as 0 and avoid doing the calculations in the while loop later on, altogether.

while ea >= es

This line of code specifies that the condition for the iterations within the loop is the error. While the error is greater, the loop runs, but after the error falls below an error criterion (defined earlier), the loop exits.

evalold = eval; %stores last eigenvalue

This line of code stores the last eigenvalue so that it can be used in calculating the error.

evect = A\*evect; %calculated eigenvector

eval = max(abs(evect)); %updates eigenvalue

These 2 lines of code are used to update the eigenvalue. First, a temporary eigenvector is calculated and then it is solved for the corresponding eigenvalue (following the power method).

evect = evect./eval; %new eigenvector

This line of code calculates a new eigenvector using the new eigenvalue calculated in the previous line.

if eval~=0

ea = abs((eval-evalold)/eval); %calculates error

end

These 3 lines of code only calculate the error if the eigenvalue is not equal to 0. This prevents any division by 0 errors that might be encountered.

end

This line of code closes the conditional while loop.

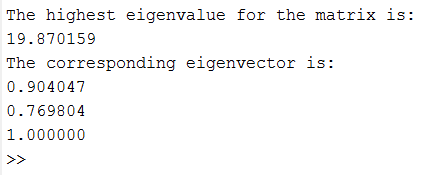
fprintf('The highest eigenvalue for the matrix is:\n%f\n',eval) %displays results

fprintf('The corresponding eigenvector is:\n')

fprintf('%f\n',evect)

These last 3 lines of code print out the results of the script in the command window, using MATLAB’s fprintf function.

**Results:**



**Discussion:**

As shown by the results, we can solve for an eigenvalue corresponding to a matrix by only using basic mathematical operations and not relying on any built in MATLAB functions. The power method works by temporarily calculating an ‘eigenvector’, calculating the corresponding eigenvalue, and then calculating a new eigenvector. This is continually updated until the guesses converge on a value. This method works because [A]{x}=λ{x}. This method works by taking the max of the absolute value of our eigenvector as the eigenvalue and as such results in the largest eigenvalue—if we were to take the max of the inverse, we would effectively be be getting 1/max which would be the minimum value (and as such would be the minimum eigenvalue). While in some cases the function does not immediately converge onto the eigenvalue, it does for this example because A = A^T (transpose).

From this problem, we learned how to apply the power method in order to solve for the largest eigenvalue and eigenvector corresponding to a matrix. We reviewed how to work with basic array and matrix functions in MATLAB. Furthermore, we reviewed how to use conditional while loops in order to continually iterate a set of functions until our error falls below a given criterion. Lastly, we reviewed how to format and output results in the command window using MATLAB’s fprintf function.

**Problem 2. From textbook Problem 13.11**

A system of two homogeneous linear ordinary differential equations with constant coefficients can be written as

.

The solutions for such equations have the form

where and are constants to be determined. Substituting this solution and its derivate into the original equations converts the system into an eigenvalue problem. The resulting eigenvalues and eigenvectors can then be used to derive the general solution to the differential equations. For example, for the two-equation case, the general solution can be written in terms of vectors as

where the eigenvector corresponding to the eigenvalue () and the s are unknown coefficients that be determined with the initial conditions.

(a) Use MATLAB to solve for the eigenvalues and eigenvectors. Print them in the command window.

(b) Employ the results of (a) and the initial conditions to determine the general solution (analytical expression), and develop a MATLAB plot of the solution for to .

**Things to discuss**

(1) Describes the function of eig

(2) Describe how to solve the problem mathematically.

**MATLAB code:**

clear all; close all; clc; %resets everything

A = [-5 3; 100 -301]; %coefficient matrix

[x,y] = eig(A); %solves for eigenvalues and eigenvectors

w = [50;100]; %given initial values

z = linsolve(x,w); %solves for constants

t = 0:.0001:1; %potential t values

v = z(1) \* x(:,1) \* exp(y(1)\*t) + z(2) \* x(:,2) \* exp(y(4)\*t); %general solution for t's

plot(t,v) %Plot

xlabel('t values') %labels

ylabel('Solution')

title('Plot of Solutions to a system of Homogeneous Linear Ordinary differential equations vs. t values')

fprintf('The eigenvalues of the matrix are:\n%f and %f\n',y(1),y(4)) %displays results

fprintf('The corresponding eigenvectors are:\n')

fprintf('%f %f\n',x(1),x(3))

fprintf('%f and %f\nrespectively.\n',x(2),x(4))

**MATLAB function:**

The purpose of this function was to solve for a system of homogeneous linear ordinary differential equations using eigenvalues/eigenvectors. To do so, we can solve for the eigenvalues and eigenvectors of a matrix corresponding to the derivatives of the variables and then solve for corresponding coefficients. These can then be plugged into the general solution for each variable. Values can then be generated and then plotted to show potential solutions to the set of equations.

clear all; close all; clc; %resets everything

This first line of code ensures that MATLAB is completely cleared and that previous scripts do not affect how this script runs.

A = [-5 3; 100 -301]; %coefficient matrix

This line of code corresponds to the ‘coefficient matrix’ given to us as part of the problem (system of equations made from derivatives).

[x,y] = eig(A); %solves for eigenvalues and eigenvectors

This line of code solves for the eigenvalues and eigenvectors of the matrix using MATLAB’s built in ‘eig’ function.

w = [50;100]; %given initial values

z = linsolve(x,w); %solves for constants

These 2 lines of code set the initial values of the problem and then solve for the constants of the general solution, using them.

t = 0:.0001:1; %potential t values

This line of code generates potential t values to be plugged into the general solution

v = z(1) \* x(:,1) \* exp(y(1)\*t) + z(2) \* x(:,2) \* exp(y(4)\*t); %general solution for t's

This line of code solves for potential solutions to the system of equations using the general solution.

plot(t,v) %Plot

This line of code plots the solutions calculated in the previous line against the t values used to generate those values, so that the user can see how the solution varies with t.

xlabel('t values') %labels

ylabel('Solution')

title('Plot of Solutions to a system of Homogeneous Linear Ordinary differential equations vs. t values')

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

fprintf('The eigenvalues of the matrix are:\n%f and %f\n',y(1),y(4)) %displays results

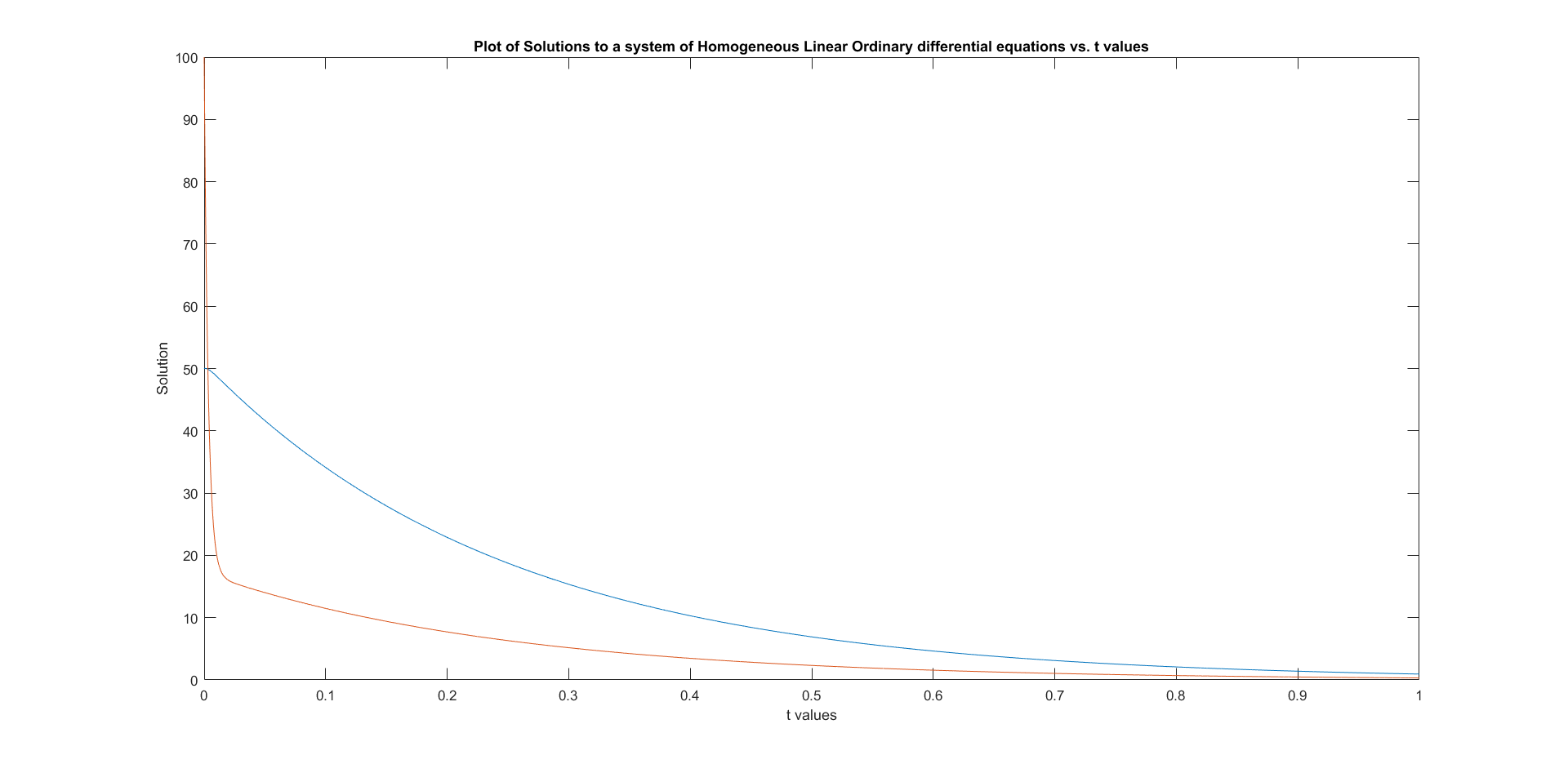
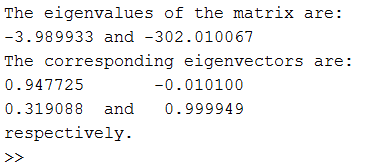
fprintf('The corresponding eigenvectors are:\n')

fprintf('%f %f\n',x(1),x(3))

fprintf('%f and %f\nrespectively.\n',x(2),x(4))

These last 4 lines of code output the eigenvalues and eigenvectors calculated earlier, as part of what the question was asking.

**Results:**

****

**Discussion:**

As shown by the results, we can solve for the general solution of a system of homogeneous linear ordinary differential equations and the result of the solution varies depending on the value t. For both y1 and y2 in the system, as t got larger, the values decreased. The eig function in MATLAB was a very efficient way in obtaining all of the eigenvalues and corresponding eigenvectors for the given matrix with little thinking involves. The results are outputted in a fashion so that the 2 matrices are of the same size as the original matrix and the columns of one correspond to the columns of the other. Mathematically, this method works because we know solutions for homogeneous linear ordinary differential equations have a similar form y = ce^(lambda\*t) (this can be derived but I don’t remember the derivation). We can then turn this into an eigenvalue problem by plugging the solution back into the original equations (where the matrix of the system corresponds to the coefficients of the equations for the derivatives), but because the original equations are derivatives (i.e. changes in values of y) we have to have an initial value in order to properly solve for the solution. By setting the coefficients equal to the initial value and solving for the corresponding constants, we then have the general solution to the system.

From this problem, we learned how to use eigenvalues and eigenvectors to solve for the general solution of a system of homogeneous linear ordinary differential equations. We also learned how to use the eig function in MATLAB in order to solve for the eigenvalues and eigenvectors of a matrix and how to use the linsolve function in order to solve a linear system of equations. Furthermore, we reviewed how to plot 2 arrays against each other in MATLAB. Lastly, we reviewed how to format and output results in the command window using MATLAB’s fprintf function.

**Problem 3. From textbook Problem 13.12**

Water flows between the North American Great Lakes as depicted in Fig. 1. Based on mass balances, the following differential equations can be written for the concentrations in each of the lakes for a pollutant that decays with first-order kinetics:

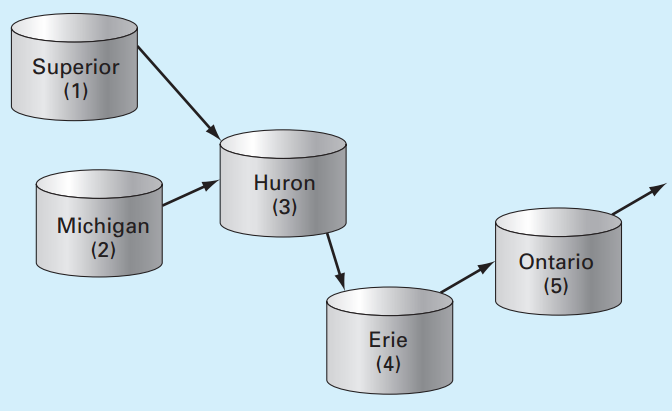


Figure 1. The North American Great Lakes. The arrows indicate how water flows between the lakes.

where the first-order decay rate (/yr), which is equal to 0.69315/(half-life). Note that the constants in each of the equations account for the flow between the lakes. Due to the testing of nuclear weapons in the atmosphere, the concentrations of strontium-90 (90Sr) in the five lakes in 1963 were approximately in unites of Bq/m3. Assuming that no additional 90Sr entered the system thereafter, *use MATLAB and the approach outlined in Problem 2 to compute and plot the concentrations in each of the lakes from 1963 through 2011*. Note that 90Sr has a half-life of 28.8 years.

**Things to discuss**

(1) Describe how to use the eigenvectors and eigenvalues to determine the general solution for the concentrations of 90Sr in each of lakes (analytical expression).

(2) Use the plot you generate to discuss the changes of concentrations in each of the lakes and the relationships between lakes.

**MATLAB code:**

clear all; close all; clc; %resets everything

k = .69315/28.8; %rate constant

A = [-(.0056+k) 0 0 0 0;0 -(.01+k) 0 0 0;.01902 .01387 -(.047+k) 0 0;0 0 .33597 -(.376+k) 0; 0 0 0 .11364 -(.133+k)]; %coefficient matrix

[x,y] = eig(A); %solves for eigenvalues and eigenvectors

v = [17.7;30.5;43.9;136.3;30.1]; %initial cconditions

z = linsolve(x,v); %solves for constants

t = 1963:2011; %years vector

c = z(1) \* x(:,1) \* exp(y(1,1)\*(t-1963)) + z(2) \* x(:,2) \* exp(y(2,2)\*(t-1963)) + z(3) \* x(:,3) \* exp(y(3,3)\*(t-1963)) + z(4) \* x(:,4) \* exp(y(4,4)\*(t-1963)) + z(5) \* x(:,5) \* exp(y(5,5)\*(t-1963)); %general solution

plot(t,c) %plot

axis([1963 2011 0 140]) %aesthetics

xlabel('Year') %labels

ylabel('Concentration of Strontium-90 (Bq/m^3)')

title('Concentration of Strontium pollutant in the North American Great Lakes')

legend('Lake Superior','Lake Michigan','Lake Huron','Lake Erie','Lake Ontario')

**MATLAB function:**

The purpose of this function is to model the concentrations of a pollutant found in the North American Great Lakes. Because we can solve for the concentrations as a system of mass balances (change in mass) between the lakes, we can then solve for the system in a similar fashion to the last problem (solving as a homogeneous linear ordinary differential equation problem). To do so, we first had to create a matrix of the changes of concentration (derivatives) in each of the lakes. We could then calculate eigenvalues and eigenvectors, solve for constants using initial values, and find the general solution by plugging all of these values into an equation. The solution could then be plotted to show the concentration over a set time period.

clear all; close all; clc; %resets everything

This first line of code ensures that MATLAB is completely cleared and that previous scripts do not affect how this script runs.

k = .69315/28.8; %rate constant

This line of code calculates the rate constant, k, as given by the problem.

A = [-(.0056+k) 0 0 0 0;0 -(.01+k) 0 0 0;.01902 .01387 -(.047+k) 0 0;0 0 .33597 -(.376+k) 0; 0 0 0 .11364 -(.133+k)]; %coefficient matrix

This line of code creates a coefficient matrix with rows corresponding to the change in concentration for each lake.

[x,y] = eig(A); %solves for eigenvalues and eigenvectors

This line of code calculates the eigenvalues and eigenvectors of the matrix generated in the line before.

v = [17.7;30.5;43.9;136.3;30.1]; %initial cconditions

z = linsolve(x,v); %solves for constants

These 2 lines of code solve for the constants given the initial conditions (in this case, initial concentrations in the lakes).

t = 1963:2011; %years vector

This line of code generates an array corresponding to each of the years we were asked to calculate concentration for. By default the steps between each value is 1 so no value is needed in the middle.

c = z(1) \* x(:,1) \* exp(y(1,1)\*(t-1963)) + z(2) \* x(:,2) \* exp(y(2,2)\*(t-1963)) + z(3) \* x(:,3) \* exp(y(3,3)\*(t-1963)) + z(4) \* x(:,4) \* exp(y(4,4)\*(t-1963)) + z(5) \* x(:,5) \* exp(y(5,5)\*(t-1963)); %general solution

This line of code is the general solution to the system of differential equations.

plot(t,c) %plot

This line of code plots the solutions to each of the corresponding years

axis([1963 2011 0 140]) %aesthetics

xlabel('Year') %labels

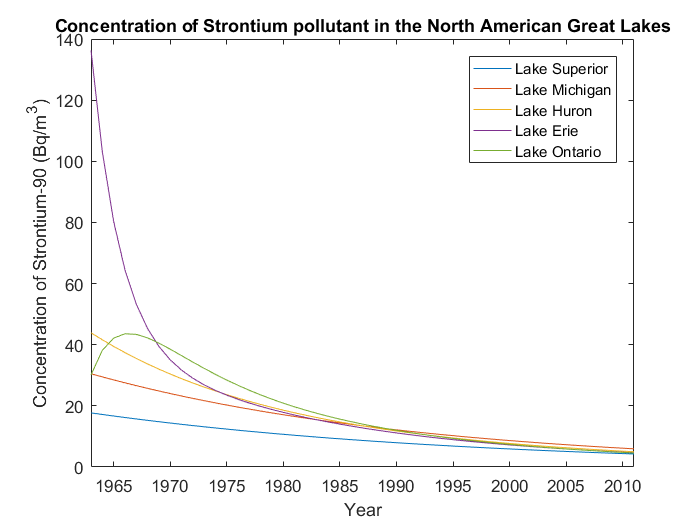
ylabel('Concentration of Strontium-90 (Bq/m^3)')

title('Concentration of Strontium pollutant in the North American Great Lakes')

legend('Lake Superior','Lake Michigan','Lake Huron','Lake Erie','Lake Ontario')

These last 5 lines of code make it so that the user can easily understand everything that is going on in the plot.

**Results:**



**Discussion:**

As shown by the results, the concentration of the pollutant in the lakes, for the most part, exponentially decreased. The one exception being Lake Ontario initially, because of the drastic decrease in concentration in lake Erie, which feeds into Lake Ontario—as the concentration began to level off in Lake Erie, the concentration began to decrease in Lake Ontario. All of this makes sense, logically, as you would expect to amount of pollutant in the lakes to slowly decrease, unless new pollutant is introduced. Like the last problem, we were able to solve for a system of homogeneous linear ordinary differential equations using eigenvalues in order to find the specific concentrations of a pollutant in a given year (and then plot it).

From this problem, we reviewed how to use eigenvalues to solve for a system of homogeneous linear ordinary differential equations (albeit on a larger scale this time). We also reviewed how to use the eig function in MATLAB in order to solve for the eigenvalues and eigenvectors of a matrix and how to use the linsolve function in order to solve a linear system of equations. Lastly, we reviewed how to plot and format plots with various lines in MATLAB.