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

**Lab Report**

**Lab #7 Chapter 13: Eigenvalues**

**Last name, First name: Swindall, Nathan**

**EID: nts279**

**Lab Section: Friday**

**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:**

Main script:

%Prepare workstation

clear all

close all

clc

%initialization

%Values needed for powereig function

A= [2 8 10; 8 4 5; 10 5 7]; %vector of differential equations

es= .00001; %threshold value

maxit =100; %max iteratons value

%Using the powereig function

[eval, evect] = powereig(A,es,maxit);

%outputs

eval

evect

Function script:

function [eval, evect] = powereig(A, es, maxit)

%The purpose of this function is to find the largest or most dominant

%eigen value. It will also find the eigen vector for this value.

%inputs

%A= the vectorized form of the system of different equations

%es= the threshold that the user wants his approximate error

%be below

%maxit= is the maximum number of iterations the program will go

%if it does not go below the threshold

n = length(A); %Finds number of rows

evect = ones(n,1); %Makes a column vector of nx1

eval = 1; %This makes the initial eigen val at 1

iter = 0; %Starts at zero iterations

ea = 100; %assigns ea a value of 100

%This value and the eval do not really matter.

%The variables are jus being set up.

while(1) %Starts a break while loop

evalold = eval; %stores the current value of eval

evect = A\*evect; %multiplies vector A by the evect vector

%The vector of ones set up earlier is out vector

eval = max(abs(evect)); %Finds the max value of the vector evect

evect = evect./eval; %Evect is then changed so that its make value is

%equal to one.

iter = iter+1; %adds one to the iter

% update error

if eval~=0 %if the eigen value is not one, this would mean that there is no solution

ea = abs((eval-evalold)/eval)\*100; %This finds the relative approximate error

end

% check stopping criteria

if (ea<=es || iter>=maxit) %Checks if ea is smaller than or equal to the es threshold

break;

end

end

**MATLAB function:**

In this problem, we are just determining the largest eigen value and its respective eigen vector. To go about this I will make a script that will call the powereig function. This script will contain the values that I will use with the powereig function. After I call the function I will display the highest eigen value and its corresponding eigen vector in the command window.

Main script:

First I will prepare the workstation so that I do not have any variables that will not affect my program.

The bottom lines accomplish this.

%Prepare workstation

clear all

close all

clc

Now it is time to setup the values that I need in order to run the powereig function. I will need of course the matrix that contains all the values, then I will also need an approximate relative threshold so the function knows when to stop. Then I will need a maximum iteration value if the function keeps going without getting to the threshold.

%initialization

The three lines below code for these initial values.

%Values needed for powereig function

A= [2 8 10; 8 4 5; 10 5 7]; %vector of differential equations

es= .00001; %threshold value

maxit =100; %max iteratons value

Now, since I have all the values that I need to call the function, I thus call the function. I put the values in the function and I make sure to set this function equal to the out values that I want. These out values are eval and evect which are the highest eigen value and its corresponding eigen value.

%Using the powereig function

[eval, evect] = powereig(A,es,maxit);

Now I call these outputs so that they will appear in the command window.

%outputs

eval

evect

Function script:

Now, it is time to dive into the function scrip that was called in the main script. Of course, with every function we first define the inputs, and in this case, also the outputs of the function. The function will take the outputs eval and evect and the inputs A, es, and maxit. A is the square matrix we start off with. The es is the approximate threshold the user puts. When the approximate relative error of the function goes below the es user value then the function will cease. The last input is the maxit which is how many iterations you want the program to go before stopping. This is helpful if the approximate error never goes below the user set es value.

function [eval, evect] = powereig(A, es, maxit)

%The purpose of this function is to find the largest or most dominant

%eigen value. It will also find the eigen vector for this value.

%inputs

%A= the vectorized form of the system of different equations

%es= the threshold that the user wants his approximate error

%be below

%maxit= is the maximum number of iterations the program will go

%if it does not go below the threshold

The first thing the program does is find the number of rows. This is done by just using the built in matlab function length(A).

n = length(A); %Finds number of rows

Before we can start looping in the program we have to have set some values already. This includes the eigen vector, the eignen value, the iteration, and the approximate error. This is done because these variables need to exist before you can loop.

The way the powereig works is by employing a initial guess eigen vector and an initial eigen value. Thus the following two lines of code do this. These initial values are used because the method uses them to make closer and closer guesses to the real eigen vale and vector.

evect = ones(n,1); %Makes a column vector of nx1

eval = 1; %This makes the initial eigen val at 1

iter = 0; %Starts at zero iterations

ea = 100; %assigns ea a value of 100

%This value and the eval do not really matter.

%The variables are jus being set up.

Now it is time for the main part of the program. We use a while loop because we want the break to only break for certain conditions. These conditions include the approximate relative error being below the user set threshold and if the program goes over the iteration max.

while(1) %Starts a break while loop

We need to store the old eigen value so we have something to compare the new eigen val to.

evalold = eval; %stores the current value of eval

We then want to get the new eigen vector. To do this we multiple the old evect by the A matrix.

evect = A\*evect; %multiplies vector A by the evect vector

%The vector of ones set up earlier is out vector

Now we need to find the new eigen value. We first find the max value of the absolute value of the eigen vector. We then will divide every value of the eigen vector by this value. The max value of the absolute value of the eigen vector is our new eigen value.

eval = max(abs(evect)); %Finds the max value of the vector evect

evect = evect./eval; %Evect is then changed so that its make value is

%equal to one.

We have to make sure to update the iteration for the next loop if it has to go to that.

iter = iter+1; %adds one to the iter

% update error

Now we have to check some conditions before we move onto to the next loop. We check if the eigen value is zero. If it is zero then there are only trivial solutions for the equation. If the value is not zero, then we find the approximate error of this loop.

if eval~=0 %if the eigen value is not one, this would mean that there is no solution

The below code is for the approximate error in the equation.

ea = abs((eval-evalold)/eval)\*100; %This finds the relative approximate error

end

Now we check if the error is below the threshold or if we have gone over the max number of iterations. If either of these happen then the loop breaks. If not we continue on looping through.

% check stopping criteria

if (ea<=es || iter>=maxit) %Checks if ea is smaller than or equal to the es threshold

break;

end

end

**Results:**

**eval =**

**19.8842**

**evect =**

**0.9035**

**0.7698**

**1.0000**

**Discussion:**

The powereig function does two things; it finds the highest eigen value and then it finds the respective eigen vector. To go about this is uses initial guesses for both the eigen value and vector. In the above problem it used the initial guess 1 as the eigen value and the vector (1;1;1) for the eigen vector. What it does with these values is plug them into the equation Ax=λx. It first finds the left side of the equation; thus it multiplies A by the initial guess eigen value. Then, after doing this we now have a value that equal λx. This λx is actually a nx1 vector. We then find the max absolute value in this vector and then divide this vector’s numbers by that. This will cause the highest value in the vector to be one. The value that we divided the vector by is the new eigen value. This is repeated over and over again until it converges on a single value. If we wanted to find the smallest eigen value and its respective eigen vector, all we have to do is do the same process with the inverse of A. In this case, the power method will converge on the largest value of 1/λ. Thus, because the largest value is 1/λ, we are actually finding the smallest value. There is a problem in the book where, when you do the power method on a matrix, it actually looks like it is diverging first before it quickly converges. This happened because there was a sign shift in the eigen vectors. In this problem, there was no sign shift, thus it did not look like it was going to diverge for a few iterations. The initial guess for the eigen value was positive and the eigen vector at the end of this problem was positive. Because of this lack of sign change, the approximate error got lower and lower each iteration instead of going up and then converging.

**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:**

%Prepare workstation

clear all

close all

clc

%initialization

A=[-5 3; 100 -301];%vector created from differential equations

b=[50;100]; %The solutions to the equations y1,y2 at zero

%eigen stuff

[vectors values] = eig(A); %This finds the eigen values

ev1= values (1,1); %Store first eigen value in a variable

ev2= values (2,2); %Store second eigen value in a variable

evec1= vectors(:,1); %separates the first eigen vector from vectors

evec2= vectors(:,2) ; %esparates the second eigen vector from vectors

%constants

c1c2=inv(vectors)\*b; %This will find all the constants in front of solution to differential equation

C1=c1c2(1) %Stores constant one

C2=c1c2(2) %Stores constant two

%Solutions to differential equations

y1=@(t) C1\*evec1(1)\*exp(ev1\*t)+C2\*evec2(1)\*exp(ev2\*t); %Equation for solution 1

y2=@(t) C1\*evec1(2)\*exp(ev1\*t)+C2\*evec2(2)\*exp(ev2\*t); %Equation for solution 2

%Print eigen values and vectors

fprintf(' Eigen values: %3.3f and %3.5f\n', ev1,ev2)

fprintf(' Eigen vector 1: %3.3f\n %21.3f\n', evec1)

fprintf(' Eigen vector 2: %3.3f\n %22.3f\n', evec2)

%Plot the two equations for the solutions

fplot(y1,[0 1]) %Plots y1

hold on %keeps the plots on the same graph

fplot(y2,[0 1]) %plots y2

hold off %turns off the hold

%Chart formatting

title('y vs t')

legend('y1','y2')

xlabel('t')

ylabel('y')

**MATLAB function:**

In this problem, we will be solving a system of differential equations. We will then find the eigen values and eigen vectors for this system of equations and create a general solution for the set of differential equations.

To start off, as always I prepare the matlab interface before starting the bulk of the code.

%Prepare workstation

clear all

close all

clc

To find the eigen values and vectors for this sytem of differential equations, I first convert them into vector form. The vector that I obtained from converting them is below. Then the other initial vector I will need is the solution to the equation at zero. This is the b vector below.

%initialization

A=[-5 3; 100 -301];%vector created from differential equations

b=[50;100]; %The solutions to the equations y1,y2 at zero

Now that we have all the initial values we can now find the eigen values and vectors. To do this for the vector A we will use the built in Matlab function eig. This function outputs the eigen values and eigen vectors for this matrix. The output vectors will be a matrix that contains both eigen vectors and the values is a diagonal matrix that contains the eigen values.

%eigen stuff

[vectors values] = eig(A); %This finds the eigen values

Later in the code we need the individual eigen values and eigen vectors. Thus the four lines of code below will separate these values form the outputs of eig(A).

ev1= values (1,1); %Store first eigen value in a variable

ev2= values (2,2); %Store second eigen value in a variable

evec1= vectors(:,1); %separates the first eigen vector from vectors

evec2= vectors(:,2) ; %esparates the second eigen vector from vectors

With the output value vectors, we can actually find the constants of the general equation. This can be done by using the inv(vectors)\*b which was used in an earlier chapter to solve system of equations.

%constants

c1c2=inv(vectors)\*b; %This will find all the constants in front of solution to

differential equation

Doing this will make it so that we now have a column vector of the constants. We will separate this column vector into the individual constants with the below code.

C1=c1c2(1) %Stores constant one

C2=c1c2(2) %Stores constant two

I now have all the pieces to put the general equation for the solutions together. This problem has real roots so the general equation will be y=c1eλt(ᶮ) + c2eλt(ᶮ). We can pretty much just insert the values into the equations to get the two solutions to the differential equations. The equations are coded below

%Solutions to differential equations

y1=@(t) C1\*evec1(1)\*exp(ev1\*t)+C2\*evec2(1)\*exp(ev2\*t); %Equation for solution 1

y2=@(t) C1\*evec1(2)\*exp(ev1\*t)+C2\*evec2(2)\*exp(ev2\*t); %Equation for solution 2

The problem called for printing the eigen vectors and eigen values found in the problem. I used the fprintf function for this below.

%Print eigen values and vectors

fprintf(' Eigen values: %3.3f and %3.5f\n', ev1,ev2)

fprintf(' Eigen vector 1: %3.3f\n %21.3f\n', evec1)

fprintf(' Eigen vector 2: %3.3f\n %22.3f\n', evec2)

The problem also called for plotting the two functions with t from 0 to 1. To plot both curves on the same graph the hold function in Matlab was used.

%Plot the two equations for the solutions

fplot(y1,[0 1]) %Plots y1

hold on %keeps the plots on the same graph

fplot(y2,[0 1]) %plots y2

hold off %turns off the hold

Finally, a title, legend and labels for both the x and y axis are needed. The code below adds all of that to the graph.

%Chart formatting

title('y vs t')

legend('y1','y2')

xlabel('t')

ylabel('y')

**Results:**

C1 =

53.6413

C2 =

82.8879

Eigen values: -3.990 and -302.01007

Eigen vector 1: 0.948

0.319

Eigen vector 2: -0.010

1.000



**Discussion:**

In this program, the eig function was used. This function does two things. It finds the eigen values for a matrix and it also finds the eigen vectors for an equation. It only takes one input. This input is the matrix you want to find the eigen values and vectors for. To mathematically do what the eig value does requires quite a bit of work. First one has to find the characteristic equation. To do this you subtract λ from the down diagonal of the matrix. Thus, in a 2x2 matrix, the number is the top left and bottom right corners would subtract the λ. Then you take the determinant of this matrix. The determinant will give you the characteristic equation. Then you use the quadratic equation to find the roots of this characteristic equation. Once you find the roots, which are your eigen values, you plug them back into the equation to find the eigen vectors. You then solve one row with each eigen value in it to get the eigen vectors. This pretty much sums up what the function accomplishes. Then, once we know what kind of roots we have, whether real, double or imaginary, we know the form of the general equation. The general form for this problem is y=c1eλt(ᶮ) + c2eλt(ᶮ). If we have initial values for when variable in the exponent is zero we can solve for the constants. When we plug zero into the equation, the only things left are the constants. We now have a system of equations we can solve. Once the equations are solved, we then have all the constants and now have the general equation for the specific matrix that was used in the problem. Then, the problem called for plotting the functions. After, plugging everything in and then separately the equation into two different solutions would could plot them on the graph. We ended up getting two curves because it is a system of differential equations. It has more than once answer.

**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:

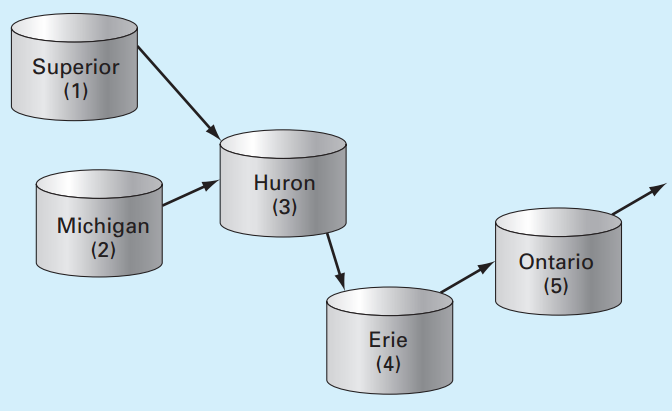


Figure 1. The North American Great Lakes. The arrows indicate how water flows between the lakes.

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:**

%Prepare workstation

clear all

close all

clc

%initialization

k=.69315/28.8; %k value

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)]; %Vector made from differential equations

b=[17.7;30.5;43.9;136.3;30.1]; %The answers when the general equations are at time zero

%Finding the eigen values and vectors

[vec val] = eig(A); %Finds the eigen values

ev1= val(1,1); %Each eigen value is set to a variable

ev2= val(2,2);

ev3= val(3,3);

ev4= val(4,4);

ev5= val(5,5);

eve1= vec(:,1); %Each eigen vector is separated from vec

eve2= vec(:,2);

eve3= vec(:,3);

eve4= vec(:,4);

eve5= vec(:,5);

%finding the constants of the general equation

c= vec\b; %This finds the constants of the general equation

c1=c(1) %this assigns each constant to its own value.

c2=c(2)

c3=c(3)

c4=c(4)

c5=c(5)

%All the forms of the general equations

%Each one of these is a solution to the system of differential equations

y1= @(t) c1\*eve1(1)\*exp(ev1\*t)+c2\*eve2(1)\*exp(ev2\*t) + c3\*eve3(1)\*exp(ev3\*t) + c4\*eve4(1)\*exp(ev4\*t) + c5\*eve5(1)\*exp(ev5\*t);

y2= @(t) c1\*eve1(2)\*exp(ev1\*t)+c2\*eve2(2)\*exp(ev2\*t) + c3\*eve3(2)\*exp(ev3\*t) + c4\*eve4(2)\*exp(ev4\*t) + c5\*eve5(2)\*exp(ev5\*t);

y3= @(t) c1\*eve1(3)\*exp(ev1\*t)+c2\*eve2(3)\*exp(ev2\*t) + c3\*eve3(3)\*exp(ev3\*t) + c4\*eve4(3)\*exp(ev4\*t) + c5\*eve5(3)\*exp(ev5\*t);

y4= @(t) c1\*eve1(4)\*exp(ev1\*t)+c2\*eve2(4)\*exp(ev2\*t) + c3\*eve3(4)\*exp(ev3\*t) + c4\*eve4(4)\*exp(ev4\*t) + c5\*eve5(4)\*exp(ev5\*t);

y5= @(t) c1\*eve1(5)\*exp(ev1\*t)+c2\*eve2(5)\*exp(ev2\*t) + c3\*eve3(5)\*exp(ev3\*t) + c4\*eve4(5)\*exp(ev4\*t) + c5\*eve5(5)\*exp(ev5\*t);

%Print the eigen values and vectors

fprintf(' Eigen values: %3.3f %3.5f %3.5f %3.5f %3.5f\n', ev1,ev2,ev3,ev4,ev5) %this prints the eigen values

fprintf(' Eigen vector 1: %3.3f\n %21.3f\n %21.3f\n %21.3f\n %21.3f\n', eve1) %Each of the following prints the unique eigen vectors found with the eigen values

fprintf(' Eigen vector 2: %3.3f\n %22.3f\n %21.3f\n %21.3f\n %21.3f\n', eve2)

fprintf(' Eigen vector 3: %3.3f\n %22.3f\n %21.3f\n %21.3f\n %21.3f\n', eve3)

fprintf(' Eigen vector 4: %3.3f\n %22.3f\n %21.3f\n %21.3f\n %21.3f\n', eve4)

fprintf(' Eigen vector 5: %3.3f\n %22.3f\n %21.3f\n %21.3f\n %21.3f\n', eve5)

%Plots of the general solutions from 1963-2011

fplot(y1,[0 48]) %plots the first solution

hold on %Makes sure that each graph is on the same plot

fplot(y2,[0 48]) %The rest plot the graphs for y2,y3,y4,y5 on the same plot as y1

fplot(y3,[0 48])

fplot(y4,[0 48])

fplot(y5,[0 48])

hold off %turns off the hold on

%Plot formatting

legend('Superior', 'Michigan','Huron', 'Erie', 'Ontario') %gives which equation goes to which lake

title('Concentration of Strontium-90 in the great lakes from 1963-2011') %title

xlabel('Time elapsed from 1963 (years)') %xlabel

ylabel('Concentration (Bq/m^3)') %ylabel

**MATLAB function:**

For this problem, we are once again solving a system of differential equations like was done in problem 2. This time though there are five differential equations to solve. The approach to this problem will be very similar to problem 2. I will construct a matrix that will represent the system of five differential equations. This system will then be used with the Matlab function eig to find the eigen vectors and eigen values. Once these are found, the constants of the general equation can be found, and then finally the general solution can be found.

The first thing I do is prepare the work the space. To do this I clear all the variables, the command window and close all unnecessary windows. This is done by the code below.

%Prepare workstation

clear all

close all

clc

Now we need to set up the matrix that represents the system of differential equations in this problem. We first set up the constant k. Then we can create the matrix.

%initialization

k=.69315/28.8; %k value

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)]; %Vector made from differential equations

The problem also gave us the initial values for the functions at a t value of zero. I made a vector of these values to be used later in the program to find the constants of the functions

b=[17.7;30.5;43.9;136.3;30.1]; %The answers when the general equations are at time zero

I can now go ahead and find the eigen values and vectors. To do this I will use the Matlab function eig. This will put all the vectors into a single matrix and then it will put all the values into a single matrix. Once, I get this vector matrix and value matrix, I will then split these matrices into their separate parts; each individual value and each individual vector will be assigned to its own variable.

%Finding the eigen values and vectors

[vec val] = eig(A); %Finds the eigen values

Here is the code for separating them.

ev1= val(1,1); %Each eigen value is set to a variable

ev2= val(2,2);

ev3= val(3,3);

ev4= val(4,4);

ev5= val(5,5);

eve1= vec(:,1); %Each eigen vector is separated from vec

eve2= vec(:,2);

eve3= vec(:,3);

eve4= vec(:,4);

eve5= vec(:,5);

Since I have matrix containing the vectors, I can find the constants for the equation at zero. To do this I will use Matrix division to solve a system of equations. I will divide the vec matrix by the initial values that we got, which are in vector b. I will assign the answers to the vector c.

%finding the constants of the general equation

c= vec\b; %This finds the constants of the general equation

I need to split of vector c into the individual constants so I can use them in the general solution. The below code will do that.

c1=c(1) %this assigns each constant to its own value.

c2=c(2)

c3=c(3)

c4=c(4)

c5=c(5)

%All the forms of the general equations

%Each one of these is a solution to the system of differential equations

At this point in the code I have all the values I need to construct five differential equations. The general form for this equation is y=c1eλt(ᶮ) + c2eλt(ᶮ) + c3eλt(ᶮ) + c4eλt(ᶮ) + c1eλt(ᶮ) + c2eλt(ᶮ). I can now do all the substitutions and make the general equations. The code below makes 5 function handles for the differential equations.

y1= @(t) c1\*eve1(1)\*exp(ev1\*t)+c2\*eve2(1)\*exp(ev2\*t) + c3\*eve3(1)\*exp(ev3\*t) + c4\*eve4(1)\*exp(ev4\*t) + c5\*eve5(1)\*exp(ev5\*t);

y2= @(t) c1\*eve1(2)\*exp(ev1\*t)+c2\*eve2(2)\*exp(ev2\*t) + c3\*eve3(2)\*exp(ev3\*t) + c4\*eve4(2)\*exp(ev4\*t) + c5\*eve5(2)\*exp(ev5\*t);

y3= @(t) c1\*eve1(3)\*exp(ev1\*t)+c2\*eve2(3)\*exp(ev2\*t) + c3\*eve3(3)\*exp(ev3\*t) + c4\*eve4(3)\*exp(ev4\*t) + c5\*eve5(3)\*exp(ev5\*t);

y4= @(t) c1\*eve1(4)\*exp(ev1\*t)+c2\*eve2(4)\*exp(ev2\*t) + c3\*eve3(4)\*exp(ev3\*t) + c4\*eve4(4)\*exp(ev4\*t) + c5\*eve5(4)\*exp(ev5\*t);

y5= @(t) c1\*eve1(5)\*exp(ev1\*t)+c2\*eve2(5)\*exp(ev2\*t) + c3\*eve3(5)\*exp(ev3\*t) + c4\*eve4(5)\*exp(ev4\*t) + c5\*eve5(5)\*exp(ev5\*t);

The problem calls for printing the eigen values and eigen vectors on the command window. I used the fprintf function in Matlab to do this.

%Print the eigen values and vectors

fprintf(' Eigen values: %3.3f %3.5f %3.5f %3.5f %3.5f\n', ev1,ev2,ev3,ev4,ev5) %this prints the eigen values

fprintf(' Eigen vector 1: %3.3f\n %21.3f\n %21.3f\n %21.3f\n %21.3f\n', eve1) %Each of the following prints the unique eigen vectors found with the eigen values

fprintf(' Eigen vector 2: %3.3f\n %22.3f\n %21.3f\n %21.3f\n %21.3f\n', eve2)

fprintf(' Eigen vector 3: %3.3f\n %22.3f\n %21.3f\n %21.3f\n %21.3f\n', eve3)

fprintf(' Eigen vector 4: %3.3f\n %22.3f\n %21.3f\n %21.3f\n %21.3f\n', eve4)

fprintf(' Eigen vector 5: %3.3f\n %22.3f\n %21.3f\n %21.3f\n %21.3f\n', eve5)

Finally, the last part of the problem is to make one plot with all the different solutions to the system of differential equations. I used fplot to plot the five functions I made above onto a single plot. To get it on a single plot all I did was use the Matlab function hold on and hold off to do this.

%Plots of the general solutions from 1963-2011

fplot(y1,[0 48]) %plots the first solution

hold on %Makes sure that each graph is on the same plot

fplot(y2,[0 48]) %The rest plot the graphs for y2,y3,y4,y5 on the same plot as y1

fplot(y3,[0 48])

fplot(y4,[0 48])

fplot(y5,[0 48])

hold off %turns off the hold on

Lastly, I needed to make labels for each solution to the differential equations. I made a legend that showed which line represented which lake and then I made a title, and x and y labels for the plot.

%Plot formatting

legend('Superior', 'Michigan','Huron', 'Erie', 'Ontario') %gives which equation goes to which lake

title('Concentration of Strontium-90 in the great lakes from 1963-2011') %title

xlabel('Time elapsed from 1963 (years)') %xlabel

ylabel('Concentration (Bq/m^3)') %ylabel

**Results:**

****

**c1 =**

**24.7493**

**c2 =**

**103.3057**

**c3 =**

**47.8332**

**c4 =**

**21.8427**

**c5 =**

**35.5689**

**Eigen values: -0.157 -0.40007 -0.07107 -0.02967 -0.03407**

**Eigen vector 1: 0.000**

**0.000**

**0.000**

**0.000**

**1.000**

**Eigen vector 2: 0.000**

**0.000**

**0.000**

**0.906**

**-0.424**

**Eigen vector 3: 0.000**

**0.000**

**0.509**

**0.520**

**0.686**

**Eigen vector 4: 0.810**

**0.000**

**0.372**

**0.338**

**0.301**

**Eigen vector 5: 0.000**

**0.857**

**0.321**

**0.295**

**0.273**

**Discussion:**

This problem was very similar to problem two in how you set it up and solve it. I first needed to solve the system of five differential equations. To go about this I made a matrix that contained the coefficients in front of each variable. Then I used the eig function in matlab to find a matrix that contained all the eigen vectors and a matrix that contained all the eigen values. Once you get these values you have to find the constants of the equations. Since I am given the initial values at zero, I can find the constants of the general solution to this system of differential equations which is y=c1eλt(ᶮ) + c2eλt(ᶮ) + c3eλt(ᶮ) + c4eλt(ᶮ) + c1eλt(ᶮ) + c2eλt(ᶮ). When you plug in zero to this equation you will get 5 equations with 5 variables. We can solve this system because there is the same amount of variables as equations. It happens to be that the matrix for this system of equations is the same as the eigen vector matrix obtained from eig. We can then just use the method of matrix division between this vector matrix and the b matrix that contains all the initial values to find the constants. The set up would look like this: x=vec\b. Once we get the answer to this problem, we need to split up the matrix obtained and we can plug in all the values in the general solution. Because we have column vectors with five rows we will have five different solutions. When I got all the solutions, I then constructed the above graph that is in the results. The graph makes sense with the given situation of how the lakes flow into each other. For Lake Ontario, which is the lake that all the other lakes flow into, the concentration actually goes up a bit, and then goes down. This is because it is getting material from the other lakes. Then it makes sense that for Lake Superior, Lake Michigan, and Lake Huron that they would be continuously losing strontium. This is because they drain into other lakes. It is a little strange though that Lake Erie has the highest amount to begin with. It could be the case that it was closest to the nuclear weapons test that took place and this is why it has the highest amount of nuclear material. Also, considering that it is so high in concentration, and Lakes Superior, Huron, and Michigan start of pretty low, we can maybe assume that the increase in strontium seen in Lake Ontario is probably mostly because of the drainage from Lake Erie. Also, going off the theory earlier, as you get closer to Lake Erie with the lakes you see a trench in higher initial concentrations of strontium. Lake Superior which is the farthest has the lowest concentration, then Lake Michigan, and then Lake Hutron