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

**Lab Report**

**Lab #5 Chapter 8-9: Linear Algebra Equations and Gauss Elimination Method**

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

**EID: VL5649**

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

**Problem 1. from textbook Problem 9.4**

Given the system of equations

2x2 + 5x3 = 1

2x1 + x2 + x3 = 1

3x1 + x2 = 2

(a) The system of equations are represented as [A]{x}={b}, please compute the determinant of this system.

(b) Use Cramer’s rule to solve for the x’s.

(c) Use Gauss elimination with partial pivoting to solve for the x’s.

(d) Substitute your results back into the original equations to check your solution.

**Things to discuss:**

There is no discussion for this problem. However, please **manually** solve the system of equations for x’s using Cramer’s rule and Gauss elimination with partial pivoting. Write down your calculation process, and take a photo of it, and then insert the figure into the manual calculation section.

(The following is your answer)

**MATLAB code:**

clear all; close all; clc; %resets everything

A = [0 2 5;2 1 1;3 1 0]; %associated constants

a = det(A); %takes determinant

B = [1;1;2]; %right side of system of equations

fprintf('The determinant of the system (A) is:\n') %formatting

fprintf('%g\n\n',a)

%cramer's rule

for i = 1:3

A(:,i) = B; %replace column

b(i) = det(A); %takes new determinant

x(i) = b(i)/a; %cramer's rule

A = [0 2 5;2 1 1;3 1 0]; %reset matrix

end

fprintf('Using Cramer''s rule: x\_1, x\_2, and x\_3 are equal to \n') %formatting

fprintf('%g, %g, and %g (respectively)\n\n',x(1),x(2),x(3))

%gauss elimination

Aug = [A B]; %forms augmented matrix

for k = 1:2

%partial pivoting portion

[~,i]=max(abs(Aug(k:3,k))); %finds row with largest column

ipr = i+k-1; %calculates largest row (relative to row working with)

if ipr ~= k

Aug([k,ipr],:)= Aug([ipr,k],:); %switches rows (pivots) if current row is less

end

for i = k+1:3

factor = Aug(i,k)/Aug(k,k); %figures out multiple to subtract one row from another

Aug(i,k:4) = Aug(i,k:4) - factor \* Aug(k,k:4); %subtracts one row from another

end

end

x = zeros(3,1); %resets x

x(3) = Aug(3,4)/Aug(3,3); %solves for 3rd value of x

for i = 2:-1:1

x(i) = (Aug(i,4) - Aug(i,i+1:3) \* x(i+1:3))/Aug(i,i); %backwards substitution to solve for second and first values of x

end

fprintf('Using Gauss Elimination: x\_1, x\_2, and x\_3 are equal to: \n') %formatting

fprintf('%g, %g, and %g (respectively)\n\n',x(1),x(2),x(3))

%plugs results back in

fprintf('When the result is plugged back into the equations, ')

Ax = int8(A \* x);

if Ax == B %checks if result is the same.

fprintf('the solution checks out\n')

else

fprintf('the solution does not check out\n')

end

**MATLAB function:**

The purpose of this function was to take the determinant of a system of equations, solve using both Cramer’s Rule and Gauss Elimination, and check to see if the solutions to both check out. To do so, first we had to generate a matrix of the scalars on the left and right side of the system of equations (2 different matrices). We could then manipulate these values to solve for the corresponding variables in 2 different ways using functions built into MATLAB.

clear all; close all; clc; %resets everything

This first line of code clears the variables and command window. I decided to go ahead and write this code in because as I was debugging it was tedious to type it in each and every time and if I didn’t it would often lead to issues calculating with the wrong numbers.

A = [0 2 5;2 1 1;3 1 0]; %associated constants

This line of code creates a matrix corresponding to the scalars in the system of equations. Each row corresponds to a different equation and each column corresponds to a different variable

a = det(A); %takes determinant

This line of code uses MATLAB’s determinant function in order to calculate the determinant of the matrix A. This value is then stored to be outputted later and used in other calculations.

B = [1;1;0]; %right side of system of equations

This line of code creates a matrix (vector) of the constants on the right side of the equal sign in the system of equations. This is needed later to calculate each variable.

fprintf('The determinant of the system (A) is:\n') %formatting

fprintf('%g\n\n',a)

These 2 lines of code display the determinant that we calculated earlier, in the command window, for user readability.

%cramer's rule

for i = 1:3

A(:,i) = B; %replace column

b(i) = det(A); %takes new determinant

x(i) = b(i)/a; %cramer's rule

A = [0 2 5;2 1 1;3 1 0]; %reset matrix

end

These 6 lines of code are the implementation of Cramer’s rule into my code. Since there are 3 variables that we are solving for, corresponding to each column of our matrix, we know we have to iterate it 3 times (once for each variable) and solve the determinant for each. The first line of the loop replaces a single column of the matrix with the vector to the right of the equal sign. The next 2 lines then take the determinant and divide it by the determinant of the original matrix. These are our solutions for the variables of x (by Cramer’s rule). We then have to reset the original matrix as we replaced a column, which is done by the last line of the loop.

fprintf('Using Cramer''s rule: x\_1, x\_2, and x\_3 are equal to \n') %formatting

fprintf('%g, %g, and %g (respectively)\n\n',x(1),x(2),x(3))

These 2 lines of code print our results from Cramer’s rule into the command window, so that the user can easily interpret and understand it.

%gauss elimination

Aug = [A B]; %forms augmented matrix

This line of code forms an Augmented matrix of our scalar matrix and the solutions vector. This is needed in Gaussian elimination because the right sides of the equations are associated with the left side and need to be moved accordingly.

for k = 1:2

This for loop, assigned for both the partial pivoting portion and the forward elimination portions of the code, works from k = 1:2. The k value specifies the row being worked with during the current iteration. The value of 3 is unnecessary because for the pivoting portion, if all of the other rows are swapped into the correct places, the last one will be too (and doesn’t need to move), and for the forward elimination portion is unnecessary because the last row will only have 1 coefficient corresponding to a single variable which can then be solved for.

%partial pivoting portion

[~,i]=max(abs(Aug(k:3,k))); %finds row with largest column

ipr = i+k-1; %calculates largest row (relative to row working with)

if ipr ~= k

Aug([k,ipr],:)= Aug([ipr,k],:); %switches rows (pivots) if current row is less

end

These 5 lines of code form the portion of our code that does partial pivoting as part of gauss elimination. The first line finds the numerical value of the row with the largest value in the column that the script is currently working in. The next line then calculates its location relative to the row that is currently being worked on. For the conditional that follows, if the greatest row is the current row, nothing happens; however, If it is not, the conditional statement then swaps the rows and proceeds on with the function.

for i = k+1:3

factor = Aug(i,k)/Aug(k,k); %figures out multiple to subtract one row from another

Aug(i,k:4) = Aug(i,k:4) - factor \* Aug(k,k:4); %subtracts one row from another

end

end

These 5 lines of code form the forward elimination portion of the code. The for loop only encompasses from whatever row you’re working on-end (3 in this case) because the assumption is that previous rows have already been reduced and the current row has been pivoted (if necessary). The next 2 lines of code then calculate the factor necessary to cancel out the leftmost non 0 coefficient in the row for every row below and subtract the product to ‘eliminate’ the variable.

x = zeros(3,1); %resets x

This line of code creates a vector of zeroes with the length as the number of variables we solve for.

x(3) = Aug(3,4)/Aug(3,3); %solves for 3rd value of x

This line of code solves for the 3rd value of x. Because we are left with x\_3 multiplied by a coefficient (1 in this case) is equal to a constant, this straightforward division will yield x\_3.

for i = 2:-1:1

x(i) = (Aug(i,4) - Aug(i,i+1:3) \* x(i+1:3))/Aug(i,i); %backwards substitution to solve for second and first values of x

end

These 3 lines of code work backwards starting from the second to last row (because we already solved for the last row in the line above). With each iteration of the loop, another variable is found which can then be ‘backwards substituted’ into the previous equations.

fprintf('Using Gauss Elimination: x\_1, x\_2, and x\_3 are equal to: \n') %formatting

fprintf('%g, %g, and %g (respectively)\n\n',x(1),x(2),x(3))

These 2 lines of code simply output the results from using Gaussian Elimination, into the command console in a user friendly way.

%plugs results back in

fprintf('When the result is plugged back into the equations, ')

Ax = int8(A \* x);

if Ax == B %checks if result is the same.

fprintf('the solution checks out\n')

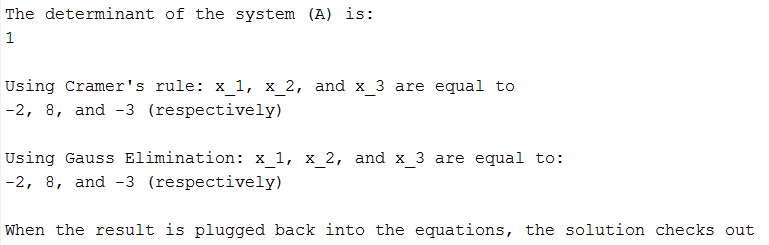
else

fprintf('the solution does not check out\n')

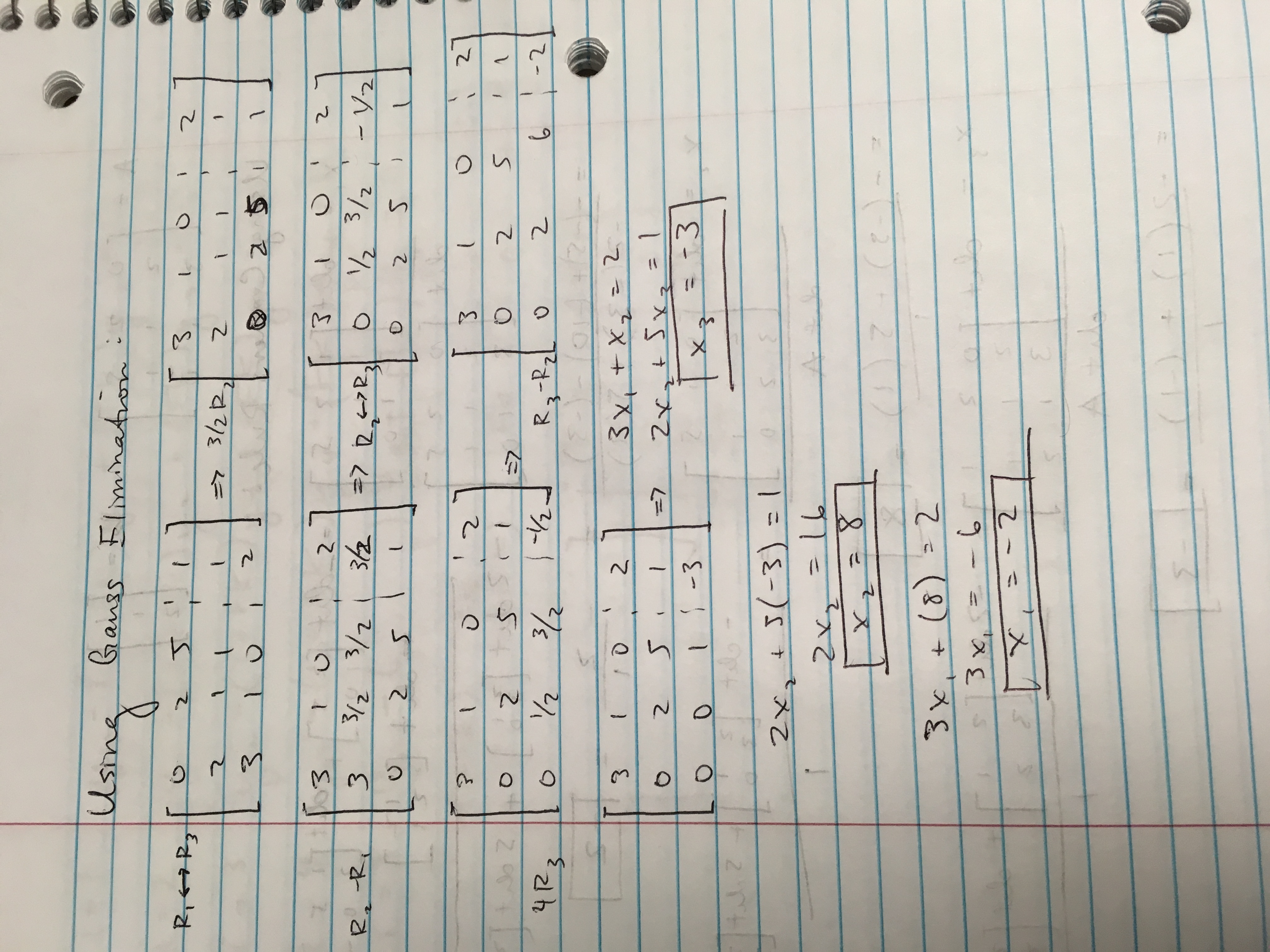
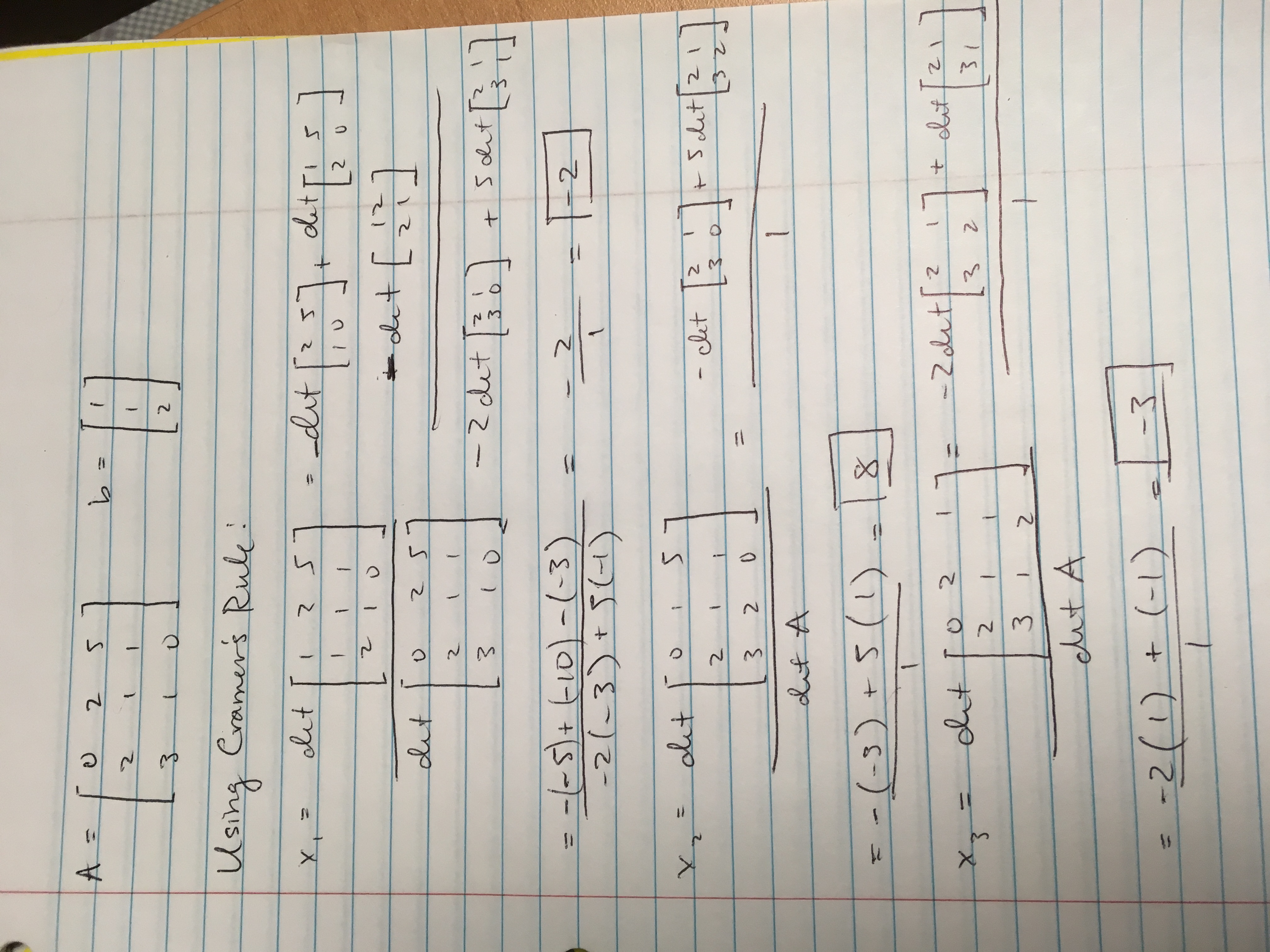
end

Rather than having the user decide if the values were the same, these 7 lines directly tell the user whether or not the results from Cramer’s rule and Gaussian Elimination line up and output a string to the console.

**Results:**

****

**Manual Calculation:**



**Problem 2. (Gauss elimination with partial pivoting)**

One of the most widely used applications of spectroscopy is for the quantitative determination of the concentration of biological molecules in solution. The absorbance of a solution , at wavelength , is given by the sum of the product of , the extinction coefficient/length of all components (from 1 to *N*) obtained at the same wavelength and the concentrations of the components of the solution, (Tinoco et al. 2002)

Consider a hypothetic experiment when a protein solution with four amino acids, *M*, *N*, *O*, and *P*, was measured using four different wavelengths, and the absorbance values were recorded. The extinction coefficients for the four amino acids for the four wavelengths are tabulated below. Use Gauss elimination with partial pivoting to estimate the concentration of the four amino acids in *M*.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Wavelength |  |  |  |  |  |
| 240 | 11300 | 8150 | 4500 | 4000 | 0.6320 |
| 250 | 5000 | 7500 | 3650 | 4200 | 0.5345 |
| 260 | 1900 | 3900 | 3000 | 4800 | 0.3310 |
| 280 | 1500 | 1400 | 2000 | 4850 | 0.196 |

**Things to discuss:**

1. What is the difference between naïve Gauss elimination and Gauss elimination with partial pivoting?
2. Why do need pivoting during the calculation of Gauss elimination?

(The following is your answer)

**MATLAB code:**

**Function:**

function x = GaussPivot\_VL(A,b)

% GaussPivot: Gauss elimination pivoting

% x = GaussPivot(A,b): Gauss elimination with pivoting.

% input:

% A = coefficient matrix

% b = right hand side vector

% output:

% x = solution vector

[m,n]=size(A);

if m~=n, error('Matrix A must be square'); end %rows must be equal to columns (need same number of equations as variables)

nb=n+1;

Aug=[A b]; %forms augmented matrix

% forward elimination

for k = 1:n-1

% partial pivoting

[~,i]=max(abs(Aug(k:n,k))); %finds the row with the highest value for the column the script is working with

ipr=i+k-1; %finds aforementioned value relative to k, current row

if ipr~=k

Aug([k,ipr],:)=Aug([ipr,k],:); %if largest row is not current row, flips them

end

for i = k+1:n

factor=Aug(i,k)/Aug(k,k); %calculates difference between rows

Aug(i,k:nb)=Aug(i,k:nb)-factor\*Aug(k,k:nb); %subtracts difference

end

end

% back substitution

x=zeros(n,1); %creates zeroes vector

x(n)=Aug(n,nb)/Aug(n,n); %solves for last value, first

for i = n-1:-1:1

x(i)=(Aug(i,nb)-Aug(i,i+1:n)\*x(i+1:n))/Aug(i,i); %plugs previous values in until all variables are solved.

end

**Main script:**

clear all; close all; clc; %resets everything

A = [11300 8150 4500 4000;5000 7500 3650 4200;1900 3900 3000 4800;1500 1400 2000 4850]; %coefficients matrix

b = [.6320;.5345;.3310;.196]; %right side of equal sign (A\_i)

x = GaussPivot\_VL(A,b); %calls Gauss elimination function with pivoting

fprintf('The concentrations for the Amino Acids M, N, O, and P are\n') %formatting

fprintf('%f, %f, %f, and %f, respectively.\n',x(1),x(2),x(3),x(4))

**MATLAB function:**

The purpose of this question was to model a real-life scenario in which you can measure the absorbencies of a solution of different amino acids of different concentrations at different wavelengths. Given the extinction coefficients at the various wavelengths, we can set up a matrix with rows corresponding to different wavelengths and columns corresponding to different amino acids. Setting this equal to the absorbencies and using Gaussian Elimination to solve for the variables yields the concentrations of the various amino acids.

function x = GaussPivot\_VL(A,b)

This first line of code in the function designates the name of the function, the end result of the function, and the variables accepted in the function.

[m,n]=size(A);

This line of code sets m equal to the number of rows and n equal to the number of columns.

if m~=n, error('Matrix A must be square'); end %rows must be equal to columns (need same number of equations as variables)

This line of code produces an error and terminates the function if the number of rows and columns don’t match. This is necessary because in order to solve for a system of equations, you must have the same number of unique equations as you do variables.

nb=n+1;

This line of code designates nb as n+1. This variable corresponds to the right side of the equation or the rightmost term in an augmented matrix.

Aug=[A b]; %forms augmented matrix

This line of code forms an Augmented matrix of our scalar matrix and the solutions vector. This is needed in Gaussian elimination because the right sides of the equations are associated with the left side and need to be moved accordingly.

for k = 1:n-1

This for loop, assigned for both the partial pivoting portion and the forward elimination portions of the code, works from k = 1:n-1 (second to last row). The k value specifies the row being worked with during the current iteration. The value of n his unnecessary because for the pivoting portion, if all of the other rows are swapped into the correct places, the last one will be too (and doesn’t need to move), and for the forward elimination portion is unnecessary because the last row will only have 1 coefficient corresponding to a single variable which can then be solved for.

[~,i]=max(abs(Aug(k:n,k))); %finds the row with the highest value for the column the script is working with

ipr=i+k-1; %finds aforementioned value relative to k, current row

if ipr~=k

Aug([k,ipr],:)=Aug([ipr,k],:); %if largest row is not current row, flips them

end

These 5 lines of code form the portion of our code that does partial pivoting as part of gauss elimination. The first line finds the numerical value of the row with the largest value in the column that the script is currently working in. The next line then calculates its location relative to the row that is currently being worked on. For the conditional statement that follows, if the greatest row is the current row, nothing happens; however, if it is not, the conditional statement will swap the current row for the one that is the greatest and proceed on from there.

for i = k+1:n

factor=Aug(i,k)/Aug(k,k); %calculates difference between rows

Aug(i,k:nb)=Aug(i,k:nb)-factor\*Aug(k,k:nb); %subtracts difference

end

end

These 5 lines of code form the forward elimination portion of the code. The for loop only encompasses from whatever row you’re working on-end (n) because the assumption is that previous rows have already been reduced and the current row has been pivoted (if necessary). The next 2 lines of code then calculate the factor necessary to cancel out the leftmost non 0 coefficient in the row for every row below and subtract the product to ‘eliminate’ the variable.

x=zeros(n,1); %creates zeroes vector

This line of code creates a zeroes vector the length of the number of variables

x(n)=Aug(n,nb)/Aug(n,n); %solves for last value, first

This line of code solves for the last row, corresponding to the last variable, of the set of equations. This is done first because it will be used in later calculations to solve for the other variables. Because the last row should just be a coefficient multiplied by the last variable is equal to a constant, this straightforward division should yield the value of the last variable.

for i = n-1:-1:1

x(i)=(Aug(i,nb)-Aug(i,i+1:n)\*x(i+1:n))/Aug(i,i); %plugs previous values in until all variables are solved.

end

These last 3 lines of code in the function work starting from the second to last row (because we already solved for the last row in the line above) in order to solve for the rest of the variables. The for loop is written backwards because the function works backwards to plug in values to solve for variables. With each iteration of the loop, another variable is found which can then be ‘backwards substituted’.

clear all; close all; clc; %resets everything

This line of code resets everything within MATLAB to ensure that leftover variables, etc. from previous functions will not affect the results of the current function.

A = [11300 8150 4500 4000;5000 7500 3650 4200;1900 3900 3000 4800;1500 1400 2000 4850]; %coefficients matrix

b = [.6320;.5345;.3310;.196]; %right side of equal sign (A\_i)

These 2 lines of code correspond to the coefficient matrix and the vector on the right side of the equal sign in the system of equations, given to us as part of the problem.

x = GaussPivot\_VL(A,b); %calls Gauss elimination function with pivoting

This line of code is very straightforward—it calls the GaussPivot function.

fprintf('The concentrations for the Amino Acids M, N, O, and P are\n') %formatting

fprintf('%f, %f, %f, and %f, respectively.\n',x(1),x(2),x(3),x(4))

These last 2 lines of code print out the results of the GaussPivot function.

**Results:**

****

**Discussion:**

As shown by the results, we were able to solve for each corresponding concentration of amino acid given the system of equations. The important aspect to be taken away from this problem is that given a system of equations, we can solve for multiple unknowns as long as we have enough equations (1 equation per unknown is needed). Using this knowledge, we can apply it to other scenarios in which we can measure the amount of the total, but not individual components, and solve for the components.

In Naïve Gaussian Elimination method, there is no pivoting involved (hence the naïve), while in the Partial Pivoting Gaussian Elimination method there is pivoting. In pivoting or being able to swap rows within the matrix, we can eliminate potential divide by 0 errors introduced in being unable to pivot in the naïve gauss elimination method.

From this problem, we learned work with matrices. We also reviewed calling a function and being able to output the results. Furthermore, we reviewed our ability to format and print values using the fprintf function in MATLAB. Lastly, we learned how to implement Gaussian elimination with pivoting as a function in MATLAB.

**Problem 3. from textbook Problem 9.13 (Tridiagonal system)**

A stage extraction process is depicted in Fig. 1. In such systems, a stream containing a weight fraction of a chemical enters from the left at a mass flow rate of . Simultaneously, a solvent carrying a weight fraction of the same chemical enters from the right at a flow rate of . Thus, for stage , a mass balance can be represented as

(1)

At each stage, an equilibrium is assumed to be established between and as in

(2)

where *K* is called a distribution coefficient*.* Equation (2) can be solved for and substituted into Eq. (1) to yield

(3)

If 400 kg/h, 0.1, 800 kg/h, 0, and 5, determine the values of and with *Tridiag*.m for a five-stage reactor. Note that Eq. (3) must be modified to account for the inflow weight fractions when applied to the first and last stage. Compare the result with that obtained by Gauss elimination method.



Figure. 1

**Things to discuss:**

1. Please write down or type in the system of equations using equation (3) and insert in the discussion section.
2. Please find an example of tridiagonal system, describe it physical or chemical meaning, and derived the equation which is similar to equation (3). For example, the 9.5 case study talks about a model of a heated rod which is exactly a tridiagonal system.

(The following is your answer)

**MATLAB code:**

**Function:**

function x = Tridiag\_VL(e,f,g,r)

% Tridiag: Tridiagonal equation solver banded system

% x = Tridiag(e,f,g,r): Tridiagonal system solver.

% input:

% e = subdiagonal vector

% f = diagonal vector

% g = superdiagonal vector

% r = right hand side vector

% output:

% x = solution vector

n=length(f); %calculates dimensions of vector

% forward elimination

for k = 2:n

factor = e(k)/f(k-1); %calculates scalar difference

f(k) = f(k) - factor\*g(k-1); %updates diagonal vector

r(k) = r(k) - factor\*r(k-1); %updates right hand side vector

end

% back substitution

x(n) = r(n)/f(n); %solves for last value

for k = n-1:-1:1

x(k) = (r(k)-g(k)\*x(k+1))/f(k); %Plugs previous values to solve for current

end

**Main script:**

clear all; close all; clc; %resets everything

F1 = 400; %flow in y direction

F2 = 800; %flow in x direction

F = F2/F1; %ratio of flows (simplifying)

K = 5; %given constant

a = -(1+F\*K); %for the sake of simplification

B = F\*K; %for the sake of simplification

A = [a B 0 0 0;1 a B 0 0;0 1 a B 0;0 0 1 a B;0 0 0 1 a]; %matrix of coefficients for gauss elimination

b = [-.1;0;0;0;0]; %right side of equations

e = [0;1;1;1;1]; %diagonals for Tridiag

f = [-11;-11;-11;-11;-11];

g = [10;10;10;10;0];

r = b;

y = Tridiag(e,f,g,r); %calls Tridiag function

yg = GaussPivot(A,b); %calls GaussPivot function

xout = y(1) \* K; %calculates xout using equation 2 and solution from tridiag

xgout = yg(1) \* K; %calculates xout using equation 2 and solution from gauss elimination

fprintf('The values for yout and xout, when solved with tridiag, are\n') %formatting

fprintf('%f and %f, respectively.\n\n',y(5),xout)

fprintf('When the same set of equations is solved with GaussPivot, yout and xout are\n')

fprintf('%f and %f, respectively.\n\n',yg(5),xgout)

if y(5) == yg(5) & xout == xgout

fprintf('These values check out.\n')

else

fprintf('These values do not check out.\n')

end

**MATLAB function:**

The purpose of this function was to model the flow of solvent in a five stage reactor given certain equations. Using the equations given, we could solve for the solvent flow at any given stage in terms of the solvent flow at other stages. With the given values of F1, F2, K, y(in), and y(6), we could then create a matrix and solve using both the tridiagonal method and the Gaussian elimination method.

function x = Tridiag\_VL(e,f,g,r)

This first line of code in the function designates the name of the function, the end result of the function, and the variables accepted in the function.

n=length(f); %calculates dimensions of vector

This line of code calculates the dimensions of the vector using the length of the diagonal vector. This is possible because we know that the matrix has to be a square matrix.

for k = 2:n

factor = e(k)/f(k-1); %calculates scalar difference

f(k) = f(k) - factor\*g(k-1); %updates diagonal vector

r(k) = r(k) - factor\*r(k-1); %updates right hand side vector

end

These 5 lines of code form the forward elimination portion of the function. We only need to work with values starting with the second value and going to the end because the first value is the reference row. The 2nd line of code calculates the factor between the f value and the e value of the same row. This factor can then be used to update the f and corresponding r vector in order to reduce the coefficients (and update constants accordingly).

x(n) = r(n)/f(n); %solves for last value

for k = n-1:-1:1

x(k) = (r(k)-g(k)\*x(k+1))/f(k); %Plugs previous values to solve for current

end

Similar to the GaussPivot function, the tridiagonal function uses backwards substitution in order to solve for the unknowns. The first line of code solves for the last unknown corresponding to the last row. Because it is a coefficient multiplied by a single unknown equal to a constant, the solution is easy to reach.

clear all; close all; clc; %resets everything

This line of code ensure the success of the program is unaffected by other programs.

F1 = 400; %flow in y direction

F2 = 800; %flow in x direction

These 2 lines of code are given values of flow in either direction.

F = F2/F1; %ratio of flows (simplifying)

This line of code was technically unnecessary, but I included it so that I could simplify a later line of code.

K = 5; %given constant

This line of code was also a given constant.

a = -(1+F\*K); %for the sake of simplification

B = F\*K; %for the sake of simplification

These 2 lines of code are also for the sake of simplification. After creating the matrix I noticed that these values come up a lot so rather than type them every time I set them as a variable.

A = [a B 0 0 0;1 a B 0 0;0 1 a B 0;0 0 1 a B;0 0 0 1 a]; %matrix of coefficients for gauss elimination

This line of code generates the coefficient matrix corresponding to the various y flows at every stage.

b = [-.1;0;0;0;0]; %right side of equations

This line of code corresponds to the right hand side matrix of constants to the right of the equal sign

e = [0;1;1;1;1]; %diagonals for Tridiag

f = [-11;-11;-11;-11;-11];

g = [10;10;10;10;0];

r = b;

These 4 lines correspond to the 4 vectors needed to use the Tridiagonal function. These values were very easy to find once the coefficient matrix, A, was found.

y = Tridiag(e,f,g,r); %calls Tridiag function

yg = GaussPivot(A,b); %calls GaussPivot function

These 2 lines of code call the Tridiag function and GaussPivot function, respectively. By calling both of these functions, we can compare the result the verify that the result is correct.

xout = y(1) \* K; %calculates xout using equation 2 and solution from tridiag

xgout = yg(1) \* K; %calculates xout using equation 2 and solution from gauss elimination

These 2 lines of code solve for xout using the values found from both Tridiag and GaussPivot. This was also done twice to check to see if the values were the same.

fprintf('The values for yout and xout, when solved with tridiag, are\n') %formatting

fprintf('%f and %f, respectively.\n\n',y(5),xout)

fprintf('When the same set of equations is solved with GaussPivot, yout and xout are\n')

fprintf('%f and %f, respectively.\n\n',yg(5),xgout)

if y(5) == yg(5) & xout == xgout

fprintf('These values check out.\n')

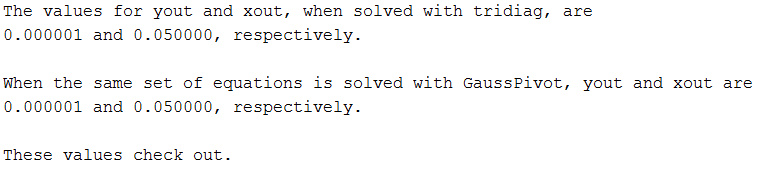
else

fprintf('These values do not check out.\n')

end

These last 9 lines of code print out all of the results of our calculations. The first 2 lines correspond to our results using the Tridiagonal function. The next 2 lines then correspond to our results using the GaussPivot Function. The last 5 lines of code form a conditional statement that prints out a string letting the user know whether or not the results between the 2 functions are the same

**Results:**



**Discussion:**

As shown by the results, yout of the system is .000001 and xout of the system is .05. These results can be verified by the fact that both the Tridiagonal and GaussPivot functions yield the same results. The important takeaways from this problem is that for a given problem, there are multiple ways to solve to be able to reach the same results and that for variables dependent on other variables, a system of equations can be written and then solved for.

Another example of a tridiagonal system is temperature change throughout a given medium. The temperature at a given position in a continuous medium can be defined as dependent on the temperature at a point on either side of that position. If the position is surrounded by hotter material, the change will be positive, but if the position is surround by colder, the change will be negative. The equation to model this would be something along the lines of

t\_(i-1) + t\_i + t\_(i+1) = 0

with other constants in between.

From this problem, we reviewed how to call functions in order to solve for a system of equations. Furthermore, we learned other iterative methods in order to solve for unknowns (tridiagonal). Lastly, we reviewed how to output results in a user-friendly method so that it can be quickly and easily understood.

