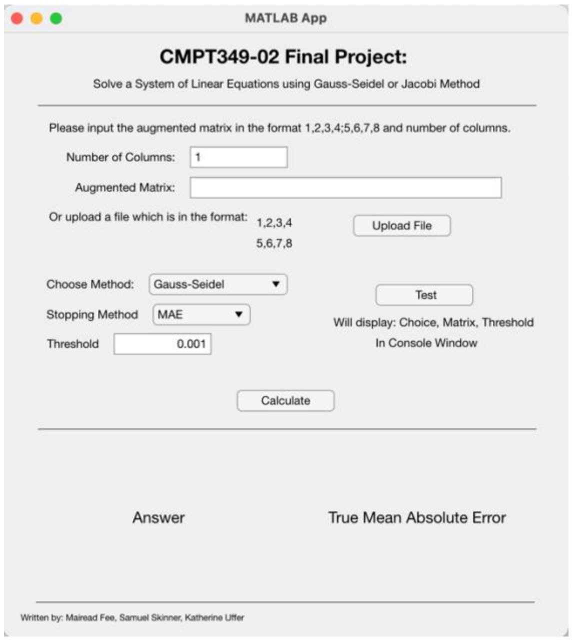
Katherine Uffer, Samuel Skinner, Mairead Fee

December 15, 2021

Numerical Computation Final Project

Contributions

For the software system, my biggest contributions were working with the design-end of the GUI, implementing the direct-input of a matrix into the GUI, and the calculation and display of the True Mean Absolute Error. My partners and I all sat down together and built up a basic knowledge of MATLAB's AppDesigner. Front-end, visual development has always intrigued me, as I am just as invested in how something operates as well as how something looks and works for the end-users. After we had spent substantial time learning the visual and technical aspects of AppDesigner, I took the initiative to make the application more user friendly, by adding instructions and listing format requirements, as well as changing font sizes and organizing the inputs and outputs within the application.


MATLAB App

CMPT349-02 Final Project:

Solve a System of Linear Equations using Gauss-Seidel or Jacobi Method

Please input the augmented matrix in the format 1,2,3,4;5,6,7,8 and number of columns.

Number of Columns: 1

Augmented Matrix:

Or upload a file which is in the format: 1,2,3,4
5,6,7,8

Choose Method: Gauss-Seidel

Stopping Method: MAE

Threshold: 0.001

Will display: Choice, Matrix, Threshold
In Console Window

Answer

True Mean Absolute Error

Written by: Mairead Fee, Samuel Skinner, Katherine Uffer

Figure 1 – Final GUI Design

As we began tackling the more specific requirements of the GUI for this project, Sam and I split the responsibility of inputting the augmented matrix, and I worked on direct input from the GUI. The simplest way for the user to input an augmented matrix was using an edit field that the user could type into. It was determined that the most efficient way to read the input data into a correctly-formatted matrix was to utilize some of MATLAB's internal functions, most notably of which is the reshape() and str2num() functions. First, the user input is taken as a string of characters, then the semicolons and commas are removed and replaces with a space character. Finally, the string is transformed into a single-dimensional array of numbers using str2num(), and the output of this call is directly put into the parameters of reshape(), which also takes the number of columns, in order to output the correct augmented matrix.

It can be noted that the use of semicolons and commas is arbitrary, as they are just removed from the string before the input reaches str2num(), but this is necessary for str2num() to operate properly, but their inclusion is for the convenience of the user, as the use of these punctuation marks help the user visually identify rows, columns, and any mistakes that may have been made inputting the function. Additionally, when deciding if either "Number of Columns" or Augmented Matrix" should create the augmented matrix from when one field was changed, it was determined that when either field is changed, the matrix should be updated, since it cannot be assumed that the user will input the number of columns first and then the augmented matrix.

```
% Value changed function: AugmentedMatrixEditField
    function AugmentedMatrixEditFieldValueChanged(app, event)
        value = app.AugmentedMatrixEditField.Value;
        B = strrep(value,';',',');
        C = char(strsplit(B));
        app.enteredMatrix = reshape(str2num(C), app.columns, [])'; %create number array with user
inputted number of columns

    end
```

Figure 2 – Function That Updates Augmented Matrix When the Matrix Itself is Updated

```
% Value changed function: NumberofColumnsEditField
    function NumberofColumnsEditFieldValueChanged(app, event)
        app.columns = app.NumberofColumnsEditField.Value; %change number of columns

        %adjust matrix if there is one
        value = app.AugmentedMatrixEditField.Value;
        B = strrep(value,';',',');
        C = char(strsplit(B));
        app.enteredMatrix = reshape(str2num(C), app.columns, [])'; %create number array with user
inputted number of columns
```

```
    end
```

Figure 3 – Function That Updates Augmented Matrix When the Number of Columns is Updated

My last major contribution was the trueError() function, which calculated the True Mean Absolute Error of the solution. Since the function needed to be called at the end of the individual GaussSeidel and Jacobi functions, is was best to call trueErrror() not inside of these functions, but outside of them after the solutions had been found. This way, no matter whose functions are being used to calculate, no change needs to be made to their source code.

```
function y = trueError(a, c)%augmented matrix, solutions
    [n,m] = size(a);
    y = 0;

    %calculate True Mean Absolute Error for solution
    for i = 1:n
        temp = 0;
        for j = 1:n
            temp = temp + a(i,j)*c(j);%vpa increases precision
        end
        y = y + abs(temp-a(i,m));
    end
    y = y/n;
end
```

Figure 4 – True Mean Absolute Error Function

Besides these specific contributions, I also assisted in troubleshooting the code that my partners were working on. Other than just miscellaneous code for the project itself, when testing the trueError() function, we were able to find a discrepancy in Sam's Gauss-Sheidel and Jacobi functions, where they were giving the correct approximations, but with incorrect error calculations.

<u>Test Runs</u>

$\begin{bmatrix} 3 & 1 & -4 & 7 \\ -2 & 3 & 1 & -5 \\ 2 & 0 & 5 & 10 \end{bmatrix}$	$\begin{bmatrix} 1 & -2 & 4 & 6 \\ 8 & -3 & 2 & 2 \\ -1 & 10 & 2 & 4 \end{bmatrix}$
System A	**System B**

<u>Gauss-Seidel: Solution to System A</u>

	Approximation	True Mean Absolute Error
Mean Absolute Approximate Error	$[3.2094 \quad 0.23432 \quad 0.71626]$	1.1512e-3
Approximate Root Mean Square Error	$[3.2094 \quad 0.23432 \quad 0.71626]$	1.1512e-3

Gauss-Seidel: Solution to System B

	Approximation	True Mean Absolute Error
Mean Absolute Approximate Error	$[-0.11323 \quad 0.075598 \quad 1.5661]$	5.3376e-4
Approximate Root Mean Square Error	$[-0.11317 \quad 0.075467 \quad 1.566]$	1.318e-4

Jacobi: Solution to System A

	Approximation	True Mean Absolute Error
Mean Absolute Approximate Error	$[3.2088 \quad 0.2353 \quad 0.71573]$	2.8881e3
Approximate Root Mean Square Error	$[3.2092 \quad 0.23399 \quad 0.71646]$	1.6623e-3

Jacobi: Solution to System B

	Approximation	True Mean Absolute Error
Mean Absolute Approximate Error	$[-0.1132 \quad 0.075469 \quad 1.566]$	1.5625e-4

Approximate Root Mean Square Error	$[-0.1132 \quad 0.075469 \quad 1.566]$	1.5625e-4

Source Code

ufferPartialPivot.m

```
%Katherine Uffer
%CMPT 439 - Fall 2021
%Due Date - October 28, 2021
%Project 6

function y = ufferPartialPivot(A)

  trigger = 0;
  y = A;
  sz = size(y); %must save temp array to access indecies, rows = 1, columns = 2
  for col = 1:sz(1) %starting at the first column, check all elements so see which is largest, then
switch rows

    maxRow = col; %default largest element is in current row, so (1,1), (2,2), ect.

    for row = col:sz(1) %since we are starting at the diagonal and moving down, start at column
value

      if gt(abs(y(row, col)), abs(y(maxRow, col))) %if the current element is greater than the
saved max element
        maxRow = row; %save new max element
        trigger = 1;
      end
    end

    for elem = 1:sz(2) %preform row swap
      temp = y(maxRow, elem);%save max element
      y(maxRow, elem) = y(col, elem); %move original element to max element
      y(col, elem) = temp; %move max element to original element
    end
  end

  if trigger == 1
    disp("The function was not diagonally dominant")
  end
end
```

ufferGaussSeidel.m

```
%Katherine Uffer
%CMPT 439 - Fall 2021
%Due Date - October 28, 2021
%Project 6

function y = ufferGaussSeidel(matrix, t, s) %augmented matrix, approximations, tolerance,
stopping criteria
   A = ufferPartialPivot(matrix);
   sz = size(A);
   v = zeros(1, sz(1));
   y = zeros(1, sz(1)); %x1, x2, x3, ... xn
   newApprox = zeros(1, sz(1));
   oldApprox = zeros(1, sz(1));
   for row = 1:sz(1)
      A(row, sz(2)) = A(row, sz(2))/A(row, row);
      newApprox(1, row) = v(1, row);
      oldApprox(1, row) = newApprox(1, row);
      for col = 1:sz(1)
         if row ~= col
            A(row, col) = A(row, col)/A(row, row);
         end
      end
   end

   err = 10;

   while err > t
      %y(1, sz(2)) = y(1, sz(2)) + 1;
      err = 0;
      for row = 1:sz(1)
         newApprox(1, row) = A(row, sz(2));
         for col = 1:sz(1)
            if row ~= col
               newApprox(1, row) = newApprox(1, row) - A(row, col)*newApprox(1, col);
            end
         end
         if s == 1
            err = err + abs(newApprox(1, row) - oldApprox(1, row));
         elseif s == 2
            err = err + (newApprox(1, row) - oldApprox(1, row))^2;
         end
      oldApprox(1, row) = newApprox(1, row);
    end

    if s == 1 %mean absolute approximate error
      err = err/sz(1);
    elseif s == 2 %approximate root mean square error
      err = (err/sz(1))^(1/2);
    end

  end

  for i = 1:sz(1)
    y(1, i) = newApprox(1, i);
  end
end
```

ufferJacobi

```
%Katherine Uffer
%CMPT 439 - Fall 2021
%Due Date - October 28, 2021
%Project 6

function y = ufferJacobi(matrix, t, s) %augmented matrix, approximations, tolerance, stopping
criteria

  A = ufferPartialPivot(matrix);
  sz = size(A);
  v = zeros(1, sz(1));
  y = zeros(1, sz(1)); %x1, x2, x3, ... xn
  newApprox = zeros(1, sz(1));
  oldApprox = zeros(1, sz(1));
  for row = 1:sz(1)
    A(row, sz(2)) = A(row, sz(2))/A(row, row);
    newApprox(1, row) = v(1, row);
    for col = 1:sz(1)
      if row ~= col
        A(row, col) = A(row, col)/A(row, row);
      end
    end
  end

  err = 10;
```

```
   while err > t
      %y(1, sz(2)) = y(1, sz(2)) + 1;
      err = 0;
      for i = 1:sz(1)
         oldApprox(1, i) = newApprox(1, i);
         newApprox(1, i) = A(i, sz(2));
      end

      for row = 1:sz(1)
         for col = 1:sz(1)
            if row ~= col
               newApprox(1, row) = newApprox(1, row) - A(row, col)*oldApprox(1, col);
            end
         end
         if s == 1
            err = err + abs(newApprox(1, row) - oldApprox(1, row));
         elseif s == 2
            err = err + (newApprox(1, row) - oldApprox(1, row))^2;
         end
      end

      if s == 1 %mean absolute approximate error
         err = err/sz(1);
      elseif s == 2 %approximate root mean square error
         err = (err/sz(1))^(1/2);
      end

   end

   err = err
   for i = 1:sz(1)
      y(1, i) = newApprox(1, i);
   end

end
```