ECON90055 Computational Economics: Assignment 2

Zelin Chen - 797036

April 11, 2017

1 Algorithm for Gaussian Elimination

I created a function called "gausselim" in Matlab. With two inputs, coefficients and results, the function can return the solution by using Guassian Elimination. I have made notes in my codes to justify purposes for each group of code.

Here I further explain my code in word:

1. Take the input and create augmented coefficient matrix. Record its size.

2. Start the loop for each column.

2-1. First thing is to make sure diagonal element in each column is non-zero. We make it by switching that row (the row where zero-value diagonal element exists in the given column) with another row that has largest value in that column.

2-2. Make the diagonal element value to one through dividing the whole row by the value of that diagonal element.

2-3. Remove the value of other rows in the given column to zero. Since diagonal element in each column has become 1. Multiply the row where diagonal element locates by the value of the row in a given column that we want to remove will produce the same value as the value of the row in a given column that we want to remove. By subtraction, we complete the elimination for one row with a given column. We repeat this for each row (other than rows where diagonal element locates) and each column.

3. After the loop, we will have eliminated augmented matrix as a combination of identity matrix and a column vector for solutions. The value of each x is simply corresponding to each value in the right-hand side of augmented matrix.

Listing 1: Code for Gaussian Elimination

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 2 - "Gaussian Elimination"
% Computational Economics
%---------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
% Cx = b, C = constant, b = results
% assume C is a square, non-singular matrix and has full rank

function x = gausselim(C,b)

% create the argument A
A = [C b];

% row of A: column is assumed to be (n+1)
n = size(A,1);

% predefine storage space for row switch
vec01 = zeros(1,n);
index01 = 0;
vec02 = zeros(1,n);

% start elimination
for j = 1:n
    % check for each diagnol element to be non-zero. if A(j,j)=0, replace row j by the row below with largest number in the same column j
    if A(j,j) == 0
        % gather each element in column j below row j, then locate the row of that largest number. Switch two rows.
        for k = j+1:n
            vec01(k) = A(k,j);
        end
        index01 = find(vec01==max(vec01));
        vec02 = A(j,:);
        A(j,:) = A(index01,:);
        A(index01,:) = vec02;
    end

        % make diagnol element into 1
        A(j,:) = A(j,:) / A(j,j);

        % make each value other than diagnol element in column j become zero
        for k = 1:n
            if (k ~= j) && (A(k,j)~=0)
                A(k,:) = A(k,:) - A(j,:)*A(k,j);
            end
        end
end
```

```
48
49 % collect values for x:
50 x = zeros(1,n);
51 for j = 1:n
52     x(j) = A(j,n+1);
53 end
54
55 end
```

2 Algorithm for Gauss-Seidel Method

I create a function called "gaussei" in Matlab. Again with two inputs, the coefficients and results, the function can return the solution by using Guassian-Seidel Method. The algorithm is explained as following:

1. Take the input and create augmented matrix. Collect number of columns and rows of augmented matrix.

2. Pre-define space to store errors. The error between each x in each iteration.

3. While the maximum of error from the updated vector of solutions is larger than the stopping rule, we run the loop.

3-1. For each x, we find its updated value by values of all other x, and then store it back to solution matrix.

3-2. compare the new updated solution vector to previous solutions, save their difference as errors. Continue the loop until the maximum element of error vector becomes smaller than the stopping rule.

4. If the loop break, that means the values in solutions have converged. So, they are the solutions for the input matrix.

Listing 2: Code for Gauss-Seidel Method

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % ECON90055 Assignment 2 - "Gaussian-Seidel"
3  % Computational Economics
4  %--------------------------
5  % Coder: Zelin Chen (797036)
6  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7
8  % Cx = b, C = constant, b = results
9  % assume C is a square, non-singular matrix and has
       full rank
10 % assume iteration will converge
11 function x = gaussei(C,b)
12 % create the argument A
13 A = [C b];
14
```

```
% row of A: column is assumed to be (n+1)
n = size(A,1);

% define a starting value of x
x_val = zeros(1,n);

% define errors
x_err = zeros(1,n);
x_err(1,1) = 1; % a value to make the while loop work,
    will be overwritten later
x_tem = zeros(1,n);
term = 0;

% iteration continue until the maxmium element in
   errors is lower than 0.0001
while max(x_err)>0.00001
    term = 1 + term;

    % iterate value for each x, given the value of all
        other x.
    for j =1:n  %row
        x_val(j) = A(j,n+1);
        for k =1:n
            if k~=j
            x_val(j) = x_val(j) - A(j,k)*x_val(k);
            end
        end
        x_val(j) = x_val(j)/A(j,j);

    end

    % update errors.
    x_err = x_val - x_tem;
    x_tem = x_val;
end

% collect value of x
x = x_val;

end
```

3 Algorithm for LU Decomposition

A function called "LU" in Matlab takes one input, coefficient matrix, and produce two matrices: lower and upper triangular matrix. The algorithm is explained as following:

1. Obtain the size of the input matrix (rows and columns).

2. Predefine a matrix space to store simplified information of each Gauss matrix. Since if the size of Gauss matrix goes large, we will have to store Gauss matrices while most of its elements are zeros. We know that Gauss matrices have their diagonal elements equal to one. In the loop of my elimination, it will eliminate one single element each step, which means that the Gauss matrices generated from each loop will have only one non-diagonal element has a value different from zero. Therefore, we only need to store the value and its location for further usage.

3. Start the loop

3-1. First thing again is to ensure non-zero in its diagonal element in each given column.

3-2. For a given column, we run a loop to reduce the value of rows below the row where diagonal element locates. Store the key information of each Gauss matrix into the pre-define space.

4. After the loop, the input matrix should have been eliminated into a upper triangular matrices.

5. Create a identity matrix to inherit information of each Gauss matrices.

6. Create a loop calculate the value for lower triangular matrices via dividing it by each Gauss matrices in order.

Listing 3: Code for LU Decomposition

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECON90055 Assignment 2 - "LU-decomposition"
% Computational Economics
%--------------------------
% Coder: Zelin Chen (797036)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% assume C is a square, non-singular matrix and has
    full rank

function [L,U] = LU(C)

% row/column of C
n = size(C,1);

% predefine storage space for row switch
vec01 = zeros(1,n);
index01 = 0; % to store the position of max
vec02 = zeros(1,n);
```

```
19
20 % in each step, we will generate one Gauss matrix 'E'.
21 % each E has only one value different from zero in its
       no-diagnol elements.
22 % to save space, we predefine a matrix to save only
      useful information for each E: its non-zero non-
      diagnol value and its location(row and column).
23 e = sum(1:n-1);
24 E_matrix = zeros(e,3);
25 e_count = 1;
26
27 % start the loop to eliminate C for U, and store E in
      each step. (for each column)
28 for i = 1:n
29    % check for each diagnol element to be non-zero. if
          A(j,j)=0, replace row j by the row below with
         largest number in the same column j
30    if C(i,i) == 0
31         % gather each element in column j below row j,
              then locate the row with largest value.
             Switch it to the row i.
32         for k = i+1:n
33             vec01(k) = C(k,i);
34         end
35         index01 = find(vec01==max(vec01));
36         vec02 = C(i,:);
37         C(i,:) = C(index01,:);
38         C(index01,:) = vec02;
39    end
40
41    % under each column, we want to remove element in
         column below (i,i) to
42    % zero.
43    for j = i+1:n
44        % only conduct when C(j,i) is not zero. if C(j,
             i) is already zero, we move to next value of
              j.
45        if C(j,i) ~= 0
46            mul = - C(j,i)/C(i,i); % the value that
                 make R(i,i)*mul = R(j,i)
47            C(j,:) = mul*C(i,:) + C(j,:);
48
49            % store key info of each E.
50            E_matrix(e_count,:) = [mul,j,i];
51            e_count = e_count+1;
52        end
```

```
53      end
54
55  end
56
57  % after our iteration, C has been eliminated into U.
58  U = C;
59
60  % solve for L
61      % we have a matix of information of all E.
62      % create a diagnol space to carry on the info of
            each E
63
64  % diagnol space: D
65  D = zeros(n,n);
66  for l = (1:n)
67      for m = (1:n)
68          if l==m
69              D(l,m) = 1;
70          else
71              D(l,m) = 0;
72          end
73      end
74  end
75
76  % let L be identity matrix first
77  L = D;
78
79  % then multiply it by inverse of each E one by one.
80  % == divide it by each E
81  for k = (1:e)
82      E = D;
83      E(E_matrix(k,2),E_matrix(k,3)) = E_matrix(k,1);
84      L = L / E;
85  end
86
87  end
```