APPM3021 Numerical Methods • Lab 1
Tyson Cross 1239448

Exercises

exercise1.m :

```
% APPM3021 Lab 1, Exercise 1

clc
clear all

A = [ 1 1 1 2;...
      1 3 2 1;...
     -1 1 3 1;...
      1 0 2 4]

b = [2; 0; 3; 2]

solution = gaussElimination(A,b)                          % Here is the function

% Output and check
correct_solution = A\b;
if ~isequal(solution,correct_solution)
    warning(['Solution is inaccurate, by a max difference of ',...
        num2str(max(max(abs(solution-correct_solution))))])
end
```

When exercise1.m is run in the workspace, the following output is displayed to the command window:

```
A =

     1     1     1     2
     1     3     2     1
    -1     1     3     1
     1     0     2     4


b =

     2
     0
     3
     2


solution =

  -14.6667
    8.3333
   -9.6667
    9.0000
```

exercise2.m :

```
% APPM3021 Lab 1, Exercise 2

clc
clear all

A = [8, 1, 6;...
     3, 5, 7;...
     4, 9, 2]

B = [5, 2, 1;...
     1, 4, 3;...
     3, 4, 3]

solution = gaussMultipleSystems(A,B)                      % Here is the function
```

```
% Output and check
correct_solution = A\B;
if ~isequal(solution,correct_solution)
    warning(['Solution is inaccurate, by a max difference of ',...
        num2str(max(max(abs(solution-correct_solution))))])
end
```

When exercise2.m is run in the workspace, the following output is displayed to the command window:

```
A =

     8     1     6
     3     5     7
     4     9     2


B =

     5     2     1
     1     4     3
     3     4     3


solution =

    0.7833   -0.0278   -0.0944
    0.0333    0.3889    0.3222
   -0.2167    0.3056    0.2389
```

exercise3.m :

```
% APPM3021 Lab 1, Exercise 3

clc
clear all

A = [8, 1, 6;...
     3, 5, 7;...
     4, 9, 2]

b = [3;9;5]

[L, U] = LUFactorization(A)                              % Here is the function

% Check the function works
lu_check = L*U;
if ~isequal(A,lu_check)
    warning(['Function is inaccurate, by a max difference of ',...
        num2str(max(max(abs(A - lu_check))))])
    disp(' ')
end

% Solve the matrix using LU decomposition
% Ax=b , A=LU, so Ax=LUx=b
% Ux=y <--- Ly=b

y = gaussElimination(L,b);
solution = backSubstitution(U,y)

% Output and check
check = A\b;
if ~isequal(solution,check)
    warning(['Solution is inaccurate, by a max difference of ',...
        num2str(max(max(abs(solution-check))))])
end
```

When exercise3.m is run in the workspace, the following output is displayed to the command window:

```
A =

     8     1     6
     3     5     7
     4     9     2
```

```
b =

     3
     9
     5


L =

    1.0000         0         0
    0.3750    1.0000         0
    0.5000    1.8378    1.0000


U =

    8.0000    1.0000    6.0000
         0    4.6250    4.7500
         0         0   -9.7297


solution =

   -0.5389
    0.5444
    1.1278
```

Questions

Question 1a)

```
% APPM3021 Lab 1, Question 1a

clc
clear all

A = [ 2,  1, -1,  2;...
      4,  5, -3,  6;...
     -2,  5, -2,  6;...
      4, 11, -4,  8]

b = [5; 9; 4; 2]

% Gauss Elimination without partial pivoting
% Using forward elimination and back substitution
solution = gaussElimination(A,b)

% Output and check
correct_solution = A\b;
if ~isequal(solution,correct_solution)
    warning(['Solution is inaccurate, by a max difference of ',...
        num2str(max(max(abs(solution-correct_solution))))])
end
```

When question1a.m is run in the workspace, the following output is displayed to the command window:

```
A =

     2     1    -1     2
     4     5    -3     6
    -2     5    -2     6
     4    11    -4     8


b =

     5
     9
     4
     2
```

```
solution =

     1
    -2
     1
     3
```

Question 1b)

```
% APPM3021 Lab 1, Question 1b

% clc
% clear all

A = [ 3,  1, -1;...
      1, -4,  2;...
     -2, -1,  5]

b = [3; -1; 2]

%   Gaussian elimination method for solving a single system of equations i.e. Ax = b
%   Using back elimination and forward substitution, without partial pivoting
%   This function returns the front, top triangular values as 0

if ~isSolvable(A)                                  % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

[M,y] = backElimination(A,b);
M
solution = forwardSubstitution(M, y)

% Output and check
correct_solution = A\b;
if ~isequal(solution,correct_solution)
    warning(['Solution is inaccurate, by a max difference of ',...
        num2str(max(max(abs(solution-correct_solution))))])
end
```

When question1b.m is run in the workspace, the following output is displayed to the command window:

```
A =

     3     1    -1
     1    -4     2
    -2    -1     5


b =

     3
    -1
     2


M =

    2.6000         0         0
    1.8000   -3.6000         0
   -2.0000   -1.0000    5.0000


solution =

    1.1538
    0.5000
    0.4000
```

Question 1c)

```
% APPM3021 Lab 1, Question 1c
```

```
clc
clear all

A = [ 1, -1,  2, -1;...
      2, -2,  3, -3;...
      1,  1,  1,  0;...
      1, -1,  4,  3]

B = [ -8, -10, -100;...
     -20, -20, -250;...
      -2,  -2,  -25;...
       4,   8,   80]

solution = gaussMultipleSystems(A,B)                % Here is the function

% Output and check
check = A\B;
if ~isequal(solution,check)
    warning(['Solution is inaccurate, by a max difference of ',...
        num2str(max(max(abs(solution-check))))])
end
```

When question1c.m is run in the workspace, the following output is displayed to the command window:

```
A =

     1    -1     2    -1
     2    -2     3    -3
     1     1     1     0
     1    -1     4     3


B =

    -8   -10  -100
   -20   -20  -250
    -2    -2   -25
     4     8    80


solution =

   -7.0000   12.0000  -57.5000
    3.0000   -5.0000   22.5000
    2.0000   -9.0000   10.0000
    2.0000    9.0000   40.0000
```

Question 1d)

```
% APPM3021 Lab 1, Question 1d

clc
clear all

A = [ 1, -1,  2, -1;...
      2, -2,  3, -3;...
      1,  1,  1,  0;...
      1, -1,  4,  3]

B = [ -8, -10, -100;...
     -20, -20, -250;...
      -2,  -2,  -25;...
       4,   8,   80]

[n,m] = size(B);
Y = zeros(n,m);
solution = Y;

[L, U] = LUFactorization(A)

% Solve the matrix using LU decomposition
% Ax=b , A=LU, so AX=LUX=B
% UX=Y <--- LY=b
```

```
for i = 1:m
Y(:,i) = gaussElimination(L,B(:,i));
solution(:,i) = backSubstitution(U,Y(:,i));
end

solution

% Output and check
check = A\B
if ~isequal(solution,check)
    warning(['Solution is inaccurate, by a max difference of ',...
        num2str(max(max(abs(solution-check))))])
end
```

When question1d.m is run in the workspace, the following output is displayed to the command window:

```
A =

     1    -1     2    -1
     2    -2     3    -3
     1     1     1     0
     1    -1     4     3


B =

    -8   -10  -100
   -20   -20  -250
    -2    -2   -25
     4     8    80


L =

     1     0     0     0
     2     1     0     0
     1     2     1     0
     1     0     2     1


U =

     1    -1     2    -1
     0     0    -1    -1
     0     0     1     3
     0     0     0    -2


solution =

    20   -33   145
     0     0     0
   -10    11   -80
     8    -1    85


check =

   -7.0000   12.0000  -57.5000
    3.0000   -5.0000   22.5000
    2.0000   -9.0000   10.0000
    2.0000    9.0000   40.0000
```

Functions and Code

isSolvable.m :

```
function x = isSolvable( A )
% Checks if input matrix is square and non-singular

x = true;
n = size(A);
if n(1) ~= n(2)
    disp('Matrix is not square')
    x = false;
    return
end

if det(A)==0
    disp('Matrix is singular')
    x = false;
    return
end

end
```

swapRow.m :

```
function X = swapRow(A,row_1,row_2)
%   Swaps row_1 with row_2 in matrix A

temp_row = A(row_1,:);                                   % store temp_row
A(row_1,:) = A(row_2,:);                                 % assign new row_1
A(row_2,:)= temp_row;                                    % assign new row_2
X = A;                                                   % return matrix

return
```

backSubstitution.m :

```
function x = backSubstitution(A,b)
%   Solves for variables and substitutes them (upwards from the bottom)
%   in a upper triangular matrix (forward eliminated system of equations)


% if ~isSolvable(A)                                        % check is matrix is square and non-
singular
%     error(strcat('Matrix is not solvable'))
% end

n = length(b);
x = zeros(n,1);                                          % initialise solution vector
if A(n,n) == 0
    error('Divide by zero: unable to solve.');
    else x(n) = b(n) / A(n,n);                           % solution to variable in bottom row
end

for i = (n-1):-1:1                                       % work ascending from the last row up
    value = b(i);                                        % find the factor
    for j = (i+1):n                                      % finish rest of entries in row
        value = value -(A(i,j) .* x(j));
    end
    if A(i,i) ~= 0
        x(i) = value / A(i,i);                           % solve for variable
    end
end
```

forwardSubstitution.m :

```
function x = forwardSubstitution(A,b)
%   Solves for variables and substitutes them (downwards from the top)
%   in a upper triangular matrix (forward eliminated system of equations)
```

```
if ~isSolvable(A)                                      % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

n = length(b);
x = zeros(n,1);                                        % initialise solution vector
x(1) = b(1) / A(1,1);                                  % solution to variable in top row
for i = 2:n                                            % work ascending from the second row down
    value = b(i);                                      % find the factor
    for j = (i+1):n                                    % finish rest of entries in row
        value = value - (A(i,j) .* x(j));
    end
    x(i) = value / A(i,i);                             % solve for variable
end
```

forwardElimination.m :

```
function [X,y] = forwardElimination(A,b)
%   Forward elimination method, takes a Matrix and vector
%   Puts Matrix A in upper triangular form

if ~isSolvable(A)                                      % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

n = length(A);
for i = 1:(n-1)                                        % for each row
    for j = (i+1):n                                    % the next row (below)
        if A(i,i) == 0
            error('Naive Gaussian does not support pivoting. Unable to solve;');
        else factor = A(j,i) / A(i,i);                 % find the factor
        end
        for k = i:n                                    % finish rest of entries in row
            A(j,k)=A(j,k)-(factor*A(i,k));             % set entry in A
        end
         b(j) = b(j) - ( factor .* b(i));              % set entry in b
    end
end
X = A;                                                 % Output assignments
y = b;
end
```

backElimination.m :

```
function [X,y] = backElimination(A,b)
%   "Back" elimination method, takes a Matrix and vector
%   Puts Matrix A in upper, reverse triangular form

if ~isSolvable(A)                                    % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

n = length(A);
entry = 0;
for i = n:-1:2                                       % for each row
    entry = entry + 1;
    for j = (i-1):-1:1                                    % row above
        if A(i,i) == 0
            error('Naive Gaussian does not support pivoting. Unable to solve;');
        else factor = A(j,i) / A(i,i);                     % find the factor
        end
        for k = entry:n                              % finish rest of entries in row
            A(j,k)=A(j,k)-(factor*A(i,k));                % set entry in A
        end
        b(j) = b(j) - ( factor .* b(i));                  % set entry in b

    end
X = A;                                               % Output assignments
y = b;
end
```

forwardEliminationWithPivoting.m :

```
function [X,y] = forwardEliminationWithPivoting(A,b)
%   Forward elimination method, takes a Matrix and vector
%   Uses partial pivoting
%   Puts Matrix A in upper triangular form

if ~isSolvable(A)                                   % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

n = length(b);
for i = 1:(n-1)                                   % for each row
    for p = (i+1):n                               % check for need to do partial-pivoting
        if (A(i,i)<A(p,i))                        % pivot is smaller than current entry
            A = swapRow(A,i,p);                   % swap rows in A
            b = swapRow(b,i,p);                   % swap row in b
        end
    end
    for j = (i+1):n                               % for each pivot along the main diagonal
        if A(i,i) == 0
            error('Divide by zero. Unable to solve;');
        else m = A(j,i) / A(i,i);                 % find the factor
        end

        for k = i:n                               % finish rest of entries in row
            A(j,k)=A(j,k)-(m*A(i,k));             % set entry in A
        end
        b(j) = b(j) - ( m .* b(i));               % set entry in b
    end
end
X = A;                                              % Output assignments
y = b;
end
```

gaussElimination.m :

```
function x = gaussElimination(A,b)
%   Gaussian elimination method for solving a single system of equations i.e. Ax = b
%   Using forward elimination and back substitution, without partial pivoting

if ~isSolvable(A)                                   % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

[M,y] = forwardElimination(A,b);
x = backSubstitution(M, y);

end
```

gaussEliminationAltered.m

```
function x = gaussEliminationAltered(A,b)
%   Gaussian elimination method for solving a single system of equations i.e. Ax = b
%   Using back elimination and forward substitution, without partial pivoting
%   This function returns the front, top triangular values as 0

if ~isSolvable(A)                                   % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

[M,y] = backElimination(A,b);
disp(M)
x = forwardSubstitution(M, y)

end
```

gaussMultipleSystems.m :

```
function X = gaussMultipleSystems( A,B )
%   Gaussian elimination method for solving multiple system of equations i.e. AX = B
%   Using forward elimination and back substitution, with partial pivoting
```

```
if ~isSolvable(A)                                % check is matrix is square and non-singular
    error(strcat('Matrix is not solvable'))
end

[n,m] = size(B);
Y = zeros(n,m);
M = zeros(size(A));

for i = 1:m
    [M,Y(:,i)] = forwardEliminationWithPivoting(A,B(:,i));
end

X = zeros(n,m);
for i = 1:m
    X(:,i) = backSubstitution(M, Y(:,i));
end


end
```

gaussEliminationLUFactorization

```
function [L,U] = LUFactorization(A)
%   Permuted LU-factorization splits a matrix into Upper and Lower matrices
%   using Doolitle Decomposition

if ~isSolvable(A)                                    % check is matrix is square and non-
singular
    error(strcat('Matrix is not solvable'))
end

n = length(A);

L = eye(n);                                          % L diagonal entries will be 1
U = zeros(n);                                        % pre-allocate upper matrix U

for i = 1:n                                          % Loop through from second row
        % Lower matrix calculation
        for j = 1:(i-1)                              % Loop through columns
            L(i,j) = A(i,j);                         % Use current A(i,j) value
            for k = 1:(j-1)                          % for each entry (lower ?)
                L(i,j)=L(i,j)-double(L(i,k)*U(k,j));    % double maintains precision of inner
 product
            end
            if U(j,j) ~= 0
                L(i,j)= L(i,j)/U(j,j);               % completes the entry's calculation
 equation
            end
        end

        % Upper matrix calculation
        for j = i:n                                  % Loop through columns
            U(i,j) = A(i,j);                         % Use current A(i,j) value
            for k = 1:(i-1)                          % for each entry in the row & column
                U(i,j)=U(i,j)-(double(L(i,k)*U(k,j)));  % assign entry (upper ?)
            end
        end
end
end
```