Contents

- House keeping
- Downloading Data
- Problem 1
- Problem 2
- Problem 3
- Problem 4
- Problem 5
- Relative Error
- Display
- Fucntions

House keeping

```
clear all; close all; clc;

%{
    CSCI: 3656
    Author: Connor O'Reilly
    Email: coor1752@colorado.edu
    Last Edited: 09/30/2021
%}

%error for matrix computations due to mat5 have been turned off on my
%machine will be explained later
```

Downloading Data

```
A{1} = readmatrix('mat1-1.txt');
A{2} = readmatrix('mat2-1.txt');
A{3} = readmatrix('mat3-1.txt');
A{4} = readmatrix('mat4-1.txt');
A{5} = readmatrix('mat5-1.txt');
```

Problem 1

Generate a right-hand-side b of all ones of appropriate size for all five matrices

```
 %sizes of matrices
s = zeros(5,2);
s(1,:) = size(A{1}); s(2,:) = size(A{2}); s(3,:) = size(A{3}); s(4,:) = size(A{4}); s(5,:) = size(A{5});

%right hand-sides
b{1} = ones(s(1,2),1); b{2} = ones(s(2,2),1); b{3} = ones(s(3,2),1); b{4} = ones(s(4,2),1); b{5} = ones(s(5,2),1);
```

Problem 2

```
%solvng Ax = b for all matrices and running a timing study

%n: number of runs to compute average from, change if becomes to slow
%t_avg: variable to store time sum to compute average
num_trials_b = 30;
t_avg = zeros(5,1);

for i = 1:5
    for j = 1:num_trials_b

        tic
        truth{i} = A{i}\b{i};
        t_avg(i) = t_avg(i) + toc/num_trials_b;

    end
end
```

Problem 3

```
%function created that solves Ax=b using either LU or cholesky is included
%in function section of script, following is the timing study

num_trials_l = 30;
t_avg_lu = zeros(5,1);
chol = zeros(5,1);

for i = 1:5
    for j = 1:num_trials_l

        tic
        [LU{i}, chol(i)] = LU_or_Chol(A{i}, b{i});
        t_avg_lu(i) = t_avg_lu(i) + toc/num_trials_b;

    end
end
```

Problem 4

```
%function created that solves Ax = b using the jacobi method is included in
%the function section of this script, following is the timing study.

num_trials_j = 30;
jacobi = cell(1,5);
t_avg_jacobi = zeros(5,1);

for i = 1:5
    for j = 1:num_trials_j

        tic
        jacobi{i} = jacobi_csci3656(A{i}, b{i}, 50);
        t_avg_jacobi(i) = t_avg_jacobi(i) + toc/num_trials_j;

    end
end
```

Problem 5

```
%function created that solves Ax = b using the Gauss-Seidel method is included in
%the function section of this script, following is the timing study.

num_trials_g = 30;
gauss = cell(1,5);
t_avg_gauss = zeros(5,1);

for i = 1:5
    for j = 1:num_trials_g

        tic
        gauss{i} = Gauss_Seidel(A{i}, b{i}, 50);
        t_avg_gauss(i) = t_avg_gauss(i) + toc/num_trials_g;

    end
end
```

Relative Error

```
rel3 = zeros(5,1); rel4 = zeros(5,1); rel5 = zeros(5,1);
for i = 1:5

    % Problem 3
    rel3(i) = norm(truth{i} - LU{i})/norm(truth{i});
    % Problem 4
    rel4(i) = norm(truth{i} - jacobi{i})/norm(truth{i});
    % Problem 5
    rel5(i) = norm(truth{i} - gauss{i})/norm(truth{i});

end
```

Display

```
%question 2
%run time averages for backslash operation
fprintf('Average run time for backslash operation on provided matrices for %i trials: \n',num_trials_b);
fprintf('mat1: %0.7f [ms]\n', t_avg(1) * 1000);
fprintf('mat2: %0.7f [ms]\n', t_avg(2) * 1000);
fprintf('mat3: %0.7f [ms]\n', t_avg(3) * 1000);
fprintf('mat4: %0.7f [ms]\n', t_avg(4) * 1000);
fprintf('mat5: %0.7f [ms]\n\n', t_avg(5) * 1000);

%question 3
%run time averages for LU and Gauss
LogicalStr = {'false', 'true'};
fprintf('Average run time for LU Decomposition or Cholesky Factorization operation on provided matrices for %i trials: \n',num_trials_l);
fprintf('mat1: %0.7f [ms], Gauss-Seidel used: %s\n', t_avg_lu(1) * 1000,LogicalStr{chol(1)+1});
fprintf('mat2: %0.7f [ms], Gauss-Seidel used: %s\n', t_avg_lu(2) * 1000,LogicalStr{chol(2)+1});
fprintf('mat3: %0.7f [ms], Gauss-Seidel used: %s\n', t_avg_lu(3) * 1000,LogicalStr{chol(3)+1});
fprintf('mat4: %0.7f [ms], Gauss-Seidel used: %s\n', t_avg_lu(4) * 1000,LogicalStr{chol(4)+1});
fprintf('mat5: %0.7f [ms], Gauss-Seidel used: %s\n\n', t_avg_lu(5) * 1000,LogicalStr{chol(5)+1});

%question 4
%run time averages for Jacobi method
fprintf('Average run time for Jacobi method on provided matrices for %i trials: \n',num_trials_j);
fprintf('mat1: %0.7f [ms]\n', t_avg_jacobi(1) * 1000);
fprintf('mat2: %0.7f [ms]\n', t_avg_jacobi(2) * 1000);
fprintf('mat3: %0.7f [ms]\n', t_avg_jacobi(3) * 1000);
fprintf('mat4: %0.7f [ms]\n', t_avg_jacobi(4) * 1000);
fprintf('mat5: %0.7f [ms]\n\n', t_avg_jacobi(5) * 1000);

%question 5
%run time averages for GS method
fprintf('Average run time for Gauss-Seidel method on provided matrices for %i trials: \n',num_trials_j);
fprintf('mat1: %0.7f [ms]\n', t_avg_gauss(1) * 1000);
fprintf('mat2: %0.7f [ms]\n', t_avg_gauss(2) * 1000);
fprintf('mat3: %0.7f [ms]\n', t_avg_gauss(3) * 1000);
fprintf('mat4: %0.7f [ms]\n', t_avg_gauss(4) * 1000);
fprintf('mat5: %0.7f [ms]\n\n', t_avg_gauss(5) * 1000);

%error methods
%relative errors for matrices and methods used
for i = 1:5
    LogicalStr = {'LU decomposition:' , 'Cholesky factorization:'};
    fprintf('Relative error for each method using matrix %i : \n', i);
    fprintf('%s: %e\n',LogicalStr{chol(i) + 1},rel3(i));
    fprintf('Jacobi Method: %e\n', rel4(i));
    fprintf('Gauss_Seidel: %e\n\n', rel5(i));
end
```

```
Average run time for backslash operation on provided matrices for 30 trials:
mat1: 0.2490233 [ms]
mat2: 0.9574800 [ms]
mat3: 0.2585167 [ms]
mat4: 0.0266500 [ms]
mat5: 0.1530833 [ms]

Average run time for LU Decomposition or Cholesky Factorization operation on provided matrices for 30 trials:
mat1: 2.5141000 [ms], Gauss-Seidel used: true
mat2: 4.7886333 [ms], Gauss-Seidel used: true
mat3: 0.3503600 [ms], Gauss-Seidel used: false
mat4: 0.1785733 [ms], Gauss-Seidel used: true
mat5: 0.2742467 [ms], Gauss-Seidel used: false

Average run time for Jacobi method on provided matrices for 30 trials:
mat1: 3.6610300 [ms]
mat2: 28.2574300 [ms]
mat3: 13.6085000 [ms]
mat4: 0.2178700 [ms]
mat5: 0.2384900 [ms]

Average run time for Gauss-Seidel method on provided matrices for 30 trials:
mat1: 3.7924667 [ms]
mat2: 28.9973300 [ms]
mat3: 13.9908167 [ms]
mat4: 0.1964667 [ms]
```

```
mat5: 0.2195967 [ms]

Relative error for each method using matrix 1 :
Cholesky factorization:: 0.000000e+00
Jacobi Method: 3.855978e-07
Gauss_Seidel: 1.742567e-11

Relative error for each method using matrix 2 :
Cholesky factorization:: 0.000000e+00
Jacobi Method: 8.366790e-16
Gauss_Seidel: 2.428222e-15

Relative error for each method using matrix 3 :
LU decomposition:: 0.000000e+00
Jacobi Method: 3.651120e+306
Gauss_Seidel: NaN

Relative error for each method using matrix 4 :
Cholesky factorization:: 0.000000e+00
Jacobi Method: 5.693926e+05
Gauss_Seidel: 4.093036e-03

Relative error for each method using matrix 5 :
LU decomposition:: NaN
Jacobi Method: NaN
Gauss_Seidel: NaN
```

Fucntions

```
% Problem 3
%solves Ax = b using either lu decomposition of cholesky factorization
%depending on if matrix is positive definite or not

%to check whether a matrix is symmetric positive definite the following
%webpage was used
%url: https://www.mathworks.com/help/matlab/math/determine-whether-matrix-is-positive-definite.html
%method 2 was used
function [exe, chol_bool] = LU_or_Chol(mat, RHS)
%{
    Purpose: solves Ax = b using LU decomposition or cholesky factorization
    depending on whether the matrix is symmetric or not

    Inputs:
        mat: A  matrix
        RHS: right hand side

    Outputs:
        exe: matrix of computed solution x
        chol_cool: boolean determining if LU decomposition or Cholesky
        factorization was used
%}

    tf = issymmetric(mat);
    chol_bool = false;
    if(tf)
        d = eig(mat);
        chol_bool  = all(d > 0);
    end

    if(chol_bool)
        %if matrix is positive definite use Cholesky factorization to solve
        %Ax = b
        R = chol(mat);
        exe = R\(R'\RHS);

    else
        %if matrix is not positive definite use LU decomposition
        [L, U, P] = lu(mat);
        y = L\(P*RHS);
        exe = U\y;

    end

end
```

```
%Problem 4


function [exe] = jacobi_csci3656(mat, RHS, maxit)
%{
    Purpose: implementation of jacobi method to solve Ax = b
    Inputs:
        mat: A matrix
        RHS: right hand side matrix
        maxit: maximum iteratiions
    outputs:
        exe: solution matrix

%}

D = diag(diag(mat));
%need to remove diagonal elements on returns from tril and triu because
%they include diagonal elements of A not satisfying A = D + L + U

low = tril(mat) - D;
high = triu(mat) - D;
x0 = ones(size(RHS));
i = 0;

while (i < maxit)

    c = RHS - ( low + high ) * x0;
    x0 = D\c;
    i = i + 1;

end

exe = x0;

end

% Problem 5

function [exe] = Gauss_Seidel(mat, RHS, maxit)
%{
    Purpose: matlab implementation of Gauss-Seidel method to solve Ax = b
    Inputs:
        mat: A matrix
        RHS: right hand side matrix
        maxit: maximum number of iterations
    Outputs:
        exe: solution matrix
%}

D = diag(diag(mat));
low = tril(mat) - D;
high = triu(mat)- D;
x0 = ones(size(RHS));
i = 0;

while(i < maxit)
    c = RHS - high*x0;
    x0 = (D+low)\c;
    i = i + 1;
end

exe = x0;

end
```

Published with MATLAB® R2021a