Name: ____

1. (10 points) Let $a_{ij} = \max\{i, j\}$ with $1 \leq i, j \leq n$ be the coffiecients of the linear system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_2 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n. \end{aligned} \tag{1}$$

Furthermore, suppose $b_i \equiv 1$ for $i = 1, 2, \ldots, n$. In class, we discussed about the MATLAB implemention of the Gaussian elimination algorithm. The associated MATLAB function is `GEpivot.m`. Use this MATLAB function to solve the above linear system for $n = 2, 5, 10$ and $n = 20$. Attach your MATLAB **driver** and include MATLAB output.

Solution: The MATLAB driver for this problem is:

```
1  clearvars; close all; clc; format short;
2
3   n = 2;                      % Dimension of the system.
4   b = ones(n,1);              % Create a vector containing 1's.
5   i = n;
6   j = n;
7   A = max((1:i)', (1:j)); % Create associated matrix.
8
9  % "Call" Gaussian Elimination:
10 [x,lu,piv] = GEpivot(A,b);
11
12 % Print Solution vector:
13 x'
```

and by changing `n` in line 3 of the script, we can compute the solution vectors for the respective cases. In particular, we have:

- $n = 2$:

```
ans =

         0    0.5000
```

- $n = 5$:

```
ans =

   -0.0000    0.0000   -0.0000    0.0000    0.2000
```

- $n = 10$:

```
ans =

    0.0000   -0.0000    0.0000         0         0 ...
         0   -0.0000    0.0000         0    0.1000
```

- $n = 20$:

```
ans =

  Columns 1 through 11

    0.0000   -0.0000    0.0000   -0.0000         0 ...
        0.0000   -0.0000    0.0000   -0.0000         0 ...
        0.0000

  Columns 12 through 20

   -0.0000    0.0000   -0.0000         0    0.0000 ...
       -0.0000    0.0000   -0.0000    0.0500
```

2. (10 points) Repeat Question 1, but use the modifed coefficient matrix given by

$$a_{ij} = \min\{i, j\}.$$

Solution: The MATLAB driver in this case is precisely the same as in Question one with the only replacement in line 7:

```
A = min((1:i)', (1:j)); % Create the modified matrix.
```

Similarly, we have

- $n = 2$:

```
ans =

     1     0
```

- $n = 5$:

```
ans =

     1     0     0     0     0
```

- $n = 10$:

```
ans =

     1     0     0     0     0     0     0     0     0     0
```

- $n = 20$:

```
ans =

Columns 1 through 19

     1     0     0     0     0     0     0     0     0     ...
       0     0     0     0     0     0     0     0     0 ...
          0

  Column 20

     0
```

3. (30 points) Write a MATLAB function that solves a general linear system

$$A\mathbf{x} = \mathbf{b},$$

by using **forward** and **backward** substitutions. Store your function as `my_lin_solver.m` whose first line should read

```
function [ x ] = my_lin_solver( A, b )
```

Inside this function, you must use the *LU* decomposition provided by the MATLAB function `lu_doolittle` that was given in class. Of course, you could either use the MATLAB's built-in function `lu` for this purpose as well!

Then, test your code with the 3×3 system:

$$\begin{aligned} 3x_1 + x_2 + 4x_3 &= 6, \\ x_2 - 2x_3 &= -3, \\ x_1 + 2x_2 - x_3 &= -2. \end{aligned}$$

The exact solution is $\mathbf{x} = [1, -1, 1]^T$. Then, use your function in order to solve the 4×4 system:

$$\begin{aligned} x_1 + x_2 + x_4 &= 2, \\ 2x_1 + x_2 - x_3 + x_4 &= 1, \\ 4x_1 - x_2 - 2x_3 + 2x_4 &= 0, \\ 3x_1 - x_2 - x_3 + x_4 &= -3. \end{aligned}$$

Compute the l_2-norm of the residual $\|\mathbf{b} - A\hat{\mathbf{x}}\|_2$ where $\hat{\mathbf{x}}$ is the solution computed for the 4×4 system. Attach **all** your codes and provide MATLAB output.

Solution: The MATLAB function `my_lin_solver.m` in question is:

```
function [x] = my_lin_solver( A, b )

% Executable statements:
    n = length(b);  % Length of rhs vector.
    x = b;               % Pre-allocate x.
    y = b;               % Pre-allocate y.

% Home-made lu decomposition MATLAB function:
[A,L,U] = lu_doolittle(A);

% Forward substitution L*y=b:
y(1) = b(1) / L(1,1);
for k = 2:n
  y(k) = ( b(k) - L(k,1:k-1)*y(1:k-1) ) / L(k,k);
end

```

```
% Backward substitution U*x = y:
x(n) = y(n) / U(n,n);
for k = n-1:-1:1
  x(k) = ( y(k) - U(k,k+1:n)*x(k+1:n) ) / U(k,k);
end

end
```

The outputs for the 3×3 and 4×4 systems are given respectively:

```
>> A = [3,1,4;0,1,-2;1,2,-1];
>> b = [6;-3;-2];
>> x = my_lin_solver(A,b);
>> x

x =

    1.000000000000000
   -1.000000000000000
    1.000000000000000


>> A = [1,1,0,1;2,1,-1,1;4,-1,-2,2;3,-1,-1,1];
>> b = [2;1;0;-3];
>> x = my_lin_solver(A,b);
>> x

x =

  -2.666666666666667
   0.666666666666667
  -1.666666666666667
   4.000000000000000

>> norm(b-A*x,2)

ans =

     1.986027322597818e-15
```

4. (20 points) Write a MATLAB function called `tridiag.m` in order to solve the linear

system $A\mathbf{x} = \mathbf{f}$ where A is an $n \times n$ **tridiagonal matrix** of the form of

$$A = \begin{bmatrix} a_1 & c_1 & & & \\ b_2 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ & & & b_n & a_n \end{bmatrix}.$$

Its first line should read

```
function [ x ] = tridiag( a, b, c, f )
```

The **inputs** are the n-dimensional vectors: $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{f}$, and the **output** is the **solution vector x**. Test your code with the 5×5 system with $a_i = 2$, $b_i = -1$, $c_i = -1$, and rhs vector $\mathbf{f} = [1, 0, 0, 0, 1]^T$. The exact solution is $\mathbf{x} = [1, 1, 1, 1, 1]^T$. Attach **all** your codes and provide MATLAB output.

Solution: The —tridiag.m— function is depicted in the following:

```
function [ x ] = tridiag( a, b, c, f )
%
% This function solves a linear system of the form of
%
%                     A * x = f
%
% where A is a tridiagonal matrix. In particular, it
% employs the LU decomposition algorithm in this case
% and subsequently the system is solved by virtue of
% forward and backward substitutions.
%
% Inputs:
%
% 1) a: vector containing the main diagonal elements.
% 2) b: >>        >>          the sub-diagonal   >> .
% 3) c: >>        >>          the super-diagonal >> .
%
% Output:
%
% 1) The solution vector x.
%

% Pre-allocate vectors:
n = length(a);   % Dimension of the matrix.
L = zeros(n,1); % Auxiliary vector (multipliers).
x = zeros(n,1); % Solution vector.
```

```
y = zeros(n,1); % Auxiliary vector for L * y = f.

% Step 1: LU - decomposition:
for k = 1:n-1
            L(k+1) = b(k) / a(k);
            a(k+1) = a(k+1) - L(k+1) * c(k);
end

% Step 2: Forward substitution, i.e., solve L * y = f:
 y(1) = f(1);
for k = 2:n
    y(k) = f(k) - L(k) * y(k-1);
end

% Step 3: Backward substitution, i.e., solve U * x = y:
 x(n) = y(n) / a(n);
for k = n-1:-1:1
    x(k) = ( y(k) - c(k) * x(k+1) ) / a(k);
end

end
```

The script for the test problem reads:

```
  n = 5;            % Dimensions.
one = ones(n,1);    % n-dimensional column vector.
  f = [1;0;0;0;1]; % Right-hand-side (rhs) vector.
 x  = trisolve( 2 * one, -one, -one, f );
```

with the corresponding MATLAB output:

```
>> x

x =

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

5. (30 points) Consider the second-order, non-homogeneous ordinary differential equation (ODE):

$$u'' - u = x, \tag{2}$$

where $u = u(x)$ satisfies the boundary conditions: $u(0) = u(1) = 0$. Problems of this sort [cf. (2)] together with boundary conditions on the unknown function $u(x)$ are called **boundary value problems** (BVPs), while the ODE given is known as the *Helmholtz equation*.

In class, we derived the so-called second-order **centered**, **finite difference approximation** of the second derivative:

$$u''(x_0) \approx \frac{u(x_0+h) - 2u(x_0) + u(x_0-h)}{h^2} + O(h^2).$$

Use this approximation and the MATLAB function `tridiag.m` in order to solve the BVP of Eq. (2) with $n = 24$ points in $[0,1]$ (see *Hints*, for details). Furthermore, if the exact solution to Eq. (2) is given by

$$u_{\text{exact}}(x) = \frac{e}{e^2-1}\left(e^x - e^{-x}\right) - x,$$

plot the **numerical** and **exact** solutions on the **same** figure using a different marker (say, open circles and solid line, respectively) and include a legend. Finally, calculate the l_2-norm of the absolute error: $\|u_{\text{exact}} - u_{\text{numerical}}\|_2$. Include your code, any figure and MATLAB output.

Hints:

(a) Divide the interval $[0,1]$ into $n+1$ equal subintervals and set $x_i = ih$, $i = 0, 1, \ldots, n{+}1$ such that $(n{+}1)h = 1$ holds. This way, we create an one-dimensional computational grid (or mesh).

(b) Then, we look for an approximate solution $u(x_i) \doteq u_i$ with $i = 1, \ldots, n$ using the boundary conditions $u_0 = u_{n+1} \equiv 0$.

(c) To do so, the BVP at the discrete level is written as a **difference equation**:

$$\begin{gathered}\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} - u_i = x_i, \Rightarrow \\ u_{i+1} - \left(2 + h^2\right) u_i + u_{i-1} = h^2 x_i, \quad i = 1, \ldots, n.\end{gathered}$$

Note that the latter equation is just a linear system of the form of $A\mathbf{x} = \mathbf{f}$ with A being a **tridiagonal matrix**.

Solution: The MATLAB driver to solve the BVP (2) using the MATLAB function `tridiag.m` is:

```
clearvars; close all; clc; format long;

% Geometry:
  n = 24;                          % Number of points on the grid.
```

```
  h = 1 / (n+1);                % Lattice spacing h.
  x = [0:h:1]';                 % x_{i}, i = 1,.., n + 2.
                                % Note x_{1} = 0 and x_{n+2} = 1.

% Create the vectors a, b, c, and f and feed them to the
% tridiagonal solver:
one = ones(n,1);                % n-dimensional unit vector.
  a = -( 2 + h^2 ) * one;       % Main diagonal elements.
  b = one;                      % Sub-diagonal elements.
  c = one;                      % Super-diagonal elements.
  f = h^2 * x(2:end-1);         % Rhs of the system.
  u = tridiag( a, b, c, f );    % Call the tridiagonal solver.
  u = [0;u;0];                  % Add the boundary conditions.

% Define the exact solution as a function uex:
uex = @(x) exp(1) * ( exp(x) - exp(-x) ) / ( exp(1)^2 - 1 ) - x;

% Plot solutions:
figure;
plot(x,uex(x),'-k','linewidth',3);
hold on;
plot(x,u,'or','linewidth',2,'markersize',12);
xlabel('$x$','interpreter','latex');
hg = legend('$\textrm {Exact}$','$\textrm {Numerical}$');
set(hg,'FontSize',20,'fontname','times','interpreter','latex');
str = ['$\|u_{\textrm {exact}}-u_{\textrm ...
    {numerical}}\|_{2}=$',...
num2str(norm(uex(x)-u,2))];
ht = title(str);
set(ht,'Interpreter','latex');
set(gca,'fontsize',24,'fontname','times');
```

The numerical and exact solutions are shown in Fig. 1 together with the numerical value of the l_2-norm in question (see, the title therein).

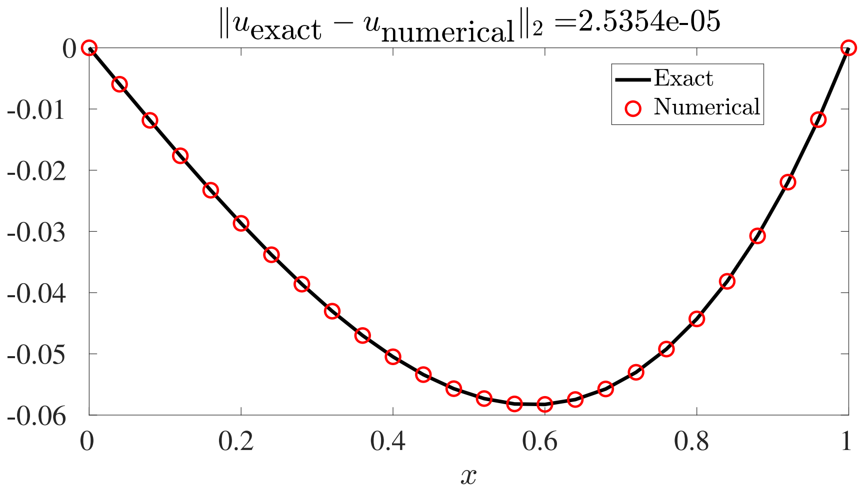

Figure 1: The numerical and exact solutions from Question 4 plotted against x.

6. (15 points) Consider the matrix

$$A = \begin{bmatrix} 1 & c \\ c & 1 \end{bmatrix}, \quad |c| \neq 1,$$

and find its condition number $\text{cond}(A)$ **by hand**. When does A become **ill-conditioned**? If we are supposed to solve $A\mathbf{x} = \mathbf{b}$, what does the ill-conditioning of A say about the linear system? How is $\text{cond}(A)$ related to $\det(A)$?

Solution: By definition, the condition number (here, we pick the ∞ norm) is

$$\text{cond}(A) = \|A\|_\infty \, \|A^{-1}\|_\infty,$$

where

$$\|A\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|.$$

In our case, we have:

$$\|A\|_\infty = \max_{1 \le i \le 2} \{1 + |c|, 1 + |c|\} \Rightarrow \boxed{\|A\|_\infty = 1 + |c|}$$

Next, we find the inverse of A (via a known formula for 2×2 matrices) which is given by

$$A^{-1} = \frac{1}{1 - c^2} \begin{bmatrix} 1 & -c \\ -c & 1 \end{bmatrix},$$

such that

$$\|A^{-1}\|_\infty = \frac{1}{1-c^2}\max_{1\le i\le 2}\{1+|c|, 1+|c|\} \Rightarrow \boxed{\|A^{-1}\|_\infty = \frac{1+|c|}{1-c^2}}.$$

This way, the condition number becomes

$$\text{cond}(A) = \|A\|_\infty\,\|A^{-1}\|_\infty \Rightarrow \boxed{\text{cond}(A) = \frac{(1+|c|)^2}{1-c^2}},$$

suggesting the matrix is well-conditioned since $|c| \neq 1$. However, it should be noted that if $|c| \approx 1$, say $|c| = 0.9999999$, the condition number becomes very large, and thus the matrix is ill-conditioned (at that particular value of c, we have $\text{cond}(A) = 10^7$!). In that case, the calculations become very sensitive while solving the associated linear system. Finally, and as per the last question of this problem, the condition number and determinant of A are related according to

$$\boxed{\text{cond}(A) = \frac{(1+|c|)^2}{\det(A)}}.$$

7. (15 points) Consider the following matrix, rhs vector and two approximate solutions

$$A = \begin{bmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0.8642 \\ 0.1440 \end{bmatrix}, \quad \mathbf{x}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 0.9911 \\ -0.4870 \end{bmatrix},$$

respectively.

(a) (2 points) Show **by hand** that $\mathbf{x} = [2, -2]^T$ is the exact solution of $A\mathbf{x} = \mathbf{b}$.

(b) (3 points) Compute the error and residual vectors in MATLAB for $\mathbf{x}_1$ and $\mathbf{x}_2$.

(c) (5 points) Use MATLAB to find $\|A\|_\infty$, $\|A^{-1}\|_\infty$ and the condition number $\text{cond}(A)$ in the ∞ norm. Note that in MATLAB the inverse of a matrix A is `inv(A)` while the condition number is available as a built-in command (try `help cond` for more details).

(d) (5 points) We proved that the relative error in the solution is bounded by

$$\frac{\|\mathbf{e}\|}{\|\mathbf{x}\|} \le \text{cond}(A)\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|},$$

where $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$ and $\hat{\mathbf{r}}$ are the error and residual vectors, respectively with $\mathbf{r} = \mathbf{b} - A\hat{\mathbf{x}}$. Verify this result for the two approximate solutions $\mathbf{x}_1$ and $\mathbf{x}_2$ given by using the ∞ norm.

Solution:

(a) Direct computation reveals that the vector given **is** an exact solution!

(b) Using MATLAB, we have following script and output:

```
% Matrix and vectors given:
   A = [1.2969,0.8648;0.2161,0.1441];
   b = [0.8642;0.1440];
  x1 = [0;1];
  x2 = [0.9911;-0.4870];
 xex = [2;-2];

% Error and residual vector for the first solution:
  e1 = xex - x1;
  r1 = b - A * x1;

% Same thing but for the second solution:
  e2 = xex - x2;
  r2 = b - A * x2;
```

```
>> e1, r1

e1 =

     2
    -3


r1 =

   1.0e-03 *

  -0.600000000000045
  -0.100000000000017

>> e2, r2

e2 =

   1.008900000000000
  -1.513000000000000
```

```
r2 =

   1.0e-07 *

   0.100000001612699
  -0.100000000224920
```

(c) Using MATLAB again, we obtain

```
>> A_inf = norm(A,'inf'), Ainv_inf = norm(inv(A), 'inf')

A_inf =

   2.161700000000000


Ainv_inf =

     1.513000002352261e+08

>> A_condinf = cond(A, 'inf')

A_condinf =

     3.270652105084882e+08
```

It can be discerned from the above results that while the norm of A is small, the norm of its inverse, i.e., A^{-1} is quite large, resulting in $\|A^{-1}\|\,\|A\| = \text{cond}(A) \approx 3.27 \times 10^8$.

(d) Finally, we validate the bound for the two solutions:

```
>> norm(e1,'inf')/norm(x,'inf'), A_condinf*norm(r1,'inf')/norm(b,'inf')

ans =

   1.499999997300285


ans =
```

```
     2.270760545071831e+05

>> norm(e2,'inf')/norm(x,'inf'), A_condinf*norm(r2,'inf')/norm(b,'inf')

ans =

   0.756499998638443


ans =

   3.784600969486987
```

That is, the bound **is satisfied in both cases**!

Date: March 5, 2020
Mathematics Department, California Polytechnic State University, San Luis Obispo, CA 93407-0403, USA
Email address: echarala@calpoly.edu