Homework 3

Math 3607, Summer 2021

Spenser Smith

Table of Contents

Problem 1.

This problem asks for a script that modifies the backsub and forelim function that we were given in pp 36 of Module 2 Lectures Slides. It also asks us to formulate a new function called ltinverse that provides the inverse of a lower triangular matrix.

```
format rat %necessary in order to get fractions

L1 = [2 0 0; 8 -7 0; 4 9 -27];
X1 = ltinverse(L1);
fprintf('The inverse of the first matrix is: \n')
```

```
The inverse of the first matrix is:
```

```
disp(X1)
```

```
     1/2           0             0
     4/7          -1/7           0
    50/189        -1/21         -1/27
```

```
L2 = [1 0 0 0; 1/3 1 0 0; 0 1/3 1 0; 0 0 1/3 1];
X2 = ltinverse(L2);
fprintf('The inverse of the second matrix is: \n')
```

```
The inverse of the second matrix is:
```

```
disp(X2)
```

```
     1             0             0             0
    -1/3           1             0             0
     1/9          -1/3           1             0
    -1/27          1/9          -1/3           1
```

Problem 2.

This problem asks for a script that completes the function myplu on pp 56 of Module 2 Lecture Slides. Mathworks Section 2.7 was accessed for the creation of this function. It also asks us to create a random integer matrix of size 500 by 50 and find the Upper and Lower Triangular matrices as well as the permutation matrix.

```
A=randi(500,500)
```

```
A =
     107        150        382        477        246        128         59        265
     427        271         17        163        447        483        390        149
     454        373        476        310        287         96        414        181
     178         12        464        272        171        305        476        401
       2         60        448         33        327        394        381         26
      50         21        474        304        224        331        340        304
     485        243        380        307        351         23        419        192
      ⋮
```

```
[L,U,P] = myplu(A)
```

```
L =
       1              0              0              0              0              0              0              0
      64/497          1              0              0              0              0              0              0
      18/497        144/1777         1              0              0              0              0              0
      33/497        277/5994       127/321          1              0              0              0              0
     487/497      -1345/1777      -694/2447       497/1000          1              0              0              0
      99/497        394/419        479/575       1735/6206       -711/1678          1              0              0
      37/71         234/419        769/2511       533/3194       1114/5877      -384/5167          1              0
      ⋮
U =
     497            351            350            158            140            449            488            417
       0         112398/251       -218/71        4177/26       14831/36       7516/127      14527/44       7701/53
       0              0        53633/110      -4325/444      89689/486      14628/35        2888/7       16480/57
       0              0              0        23762/49       -6208/95       6123/673     -95241/676      -3470/67
       0              0              0              0        39244/53      -4654/25       16934/67      32617/20
       0              0              0              0              0      -19783/37       -3463/8       -2895/16
       0              0              0              0              0              0      -25811/44     -13283/10
      ⋮
P =
       0              0              0              0              0              0              0              0
       0              0              0              0              0              0              0              0
       0              0              0              0              0              0              0              0
       0              0              0              0              0              0              0              0
       0              0              0              0              0              0              0              0
       0              0              0              0              0              0              0              0
       0              0              0              0              0              0              0              0
      ⋮
```

Problem 3.

This problem asks for a script that computes the determinant of a matrix A by calling a function called 'determinant.' Then, it asks us to find the relative error between our function and the built-in MATLAN function 'det' for a series of magic matrices for n = 3,4,...,7.

```
% Part A
```

if $A = LU$

then $\det(A) = \det(LU) = \det(L)\det(U)$

but we know $\det(L) = \prod_{i=1}^{n} l_{ii}$

$\det(U) = \prod_{i=1}^{n} u_{ii}$

and since the diagonal of L is always just 1's

then

$$\boxed{\det(A) = 1 \cdot \prod_{i=1}^{n} u_{ii}}$$

```
% Part B
A=randi(10,3,3)
```

```
A =
       7              2              2
       5              5              9
       4              1              1
```

```
X=determinant(A); %test to make sure function works
fprintf('The determinant is: %d \n',X)
```

```
The determinant is: 4.000000e+00
```

```
fprintf('n     my value       relative eror')
```

```
n     my value       relative eror
```

```
for n=3:7
    magic(n);
    Y=det(magic(n)); %actual MATLAB function to get det.
    X=determinant(magic(n)); %my function to get det.
    rel_err=abs((X-Y)/Y); %calculates relative error
    fprintf('%d     %5.2e      %f\n',n,X,rel_err)
end
```

```
3     -3.60e+02      0.000000
4     3.62e-13      0.294118
5     5.07e+06      0.000000
6     0.00e+00      1.000000
7     -3.48e+11      0.000000
```

Problem 4.

This problem asks us to find the most efficient (and correct) way to code a variety of problems and count the corresponding flops that each one requires.

```
% a) x = A*(B*(C*(D*b))) and it takes 8n^2 flops
%    because a matrix times vector is 2n^2 and
```

```
%    we are doing that 4 times.

% b) x = B*(A\b) and it takes 2n^3 for large n
%    because A\b takes 2n^3 and B times a vector is
%    again 2n^2 so for large values of n the number
%    of flops is approx 2n^3.

% d) x = (B)\(C+A)*b and it takes 2n^3 for large n
%    because B\(C+A) takes 2n^3 flops and that answer
%    times a vector only takes 2n^2 flops so for large
%    values of n the number of flops is approx 2n^3.
```

Problem 5.

This problem asks us to perform a variety of tasks that are centered around the norm and condition number of matrices.

a) No, b/c the norm is zero only when A is the zero matrix.

For example, consider $A = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$

which is singular but has positive norm.

$A^T = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$

$A^T A = \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix}$

$\begin{bmatrix} 2-\lambda & 4 \\ 4 & 8-\lambda \end{bmatrix}$

$(2-\lambda)(8-\lambda) - 16 = 0$

$16 - 10\lambda + \lambda^2 - 16 = 0$

$\lambda^2 - 10\lambda$

$\lambda(\lambda - 10) = 0$

$\lambda = 10$

$\sqrt{10}$ and is nonzero

b) is cond. # of sing. matrix nec. ∞?

Yes since $\dfrac{\max_{\|y\|_p=1} \|A^{-1}y\|_p}{\min_{\|y\|_p=1} \|A^{-1}y\|_p} = \text{cond. \#}$

and the $\max_{\|y\|_p=1} \|A^{-1}y\|_p$ of a singular matrix is ∞ thus the ratio will always be $\frac{\infty}{x}$ where x is any number. This gives a condition number that is always infinite.

c) is $K_p(A) = K_p(A^{-1})$ for nonsingular TRUE

yes, since $\text{cond}(A) = \|A\| \cdot \|A^{-1}\|$

then $\text{cond}(A^{-1}) = \|A^{-1}\| \cdot \|A\|$

which means that $K_p(A) = K_p(A^{-1})$.

d) any matrices that yield $K(A) < 1$

No, since we know it's $\frac{\max}{\min}$ the condition number will always be ≥ 1.

e) find $A \in \mathbb{R}^{10\times10}$ s.t. $\det(A) = 10^{-20}$ and $K_2(A) \geq 1$.

$$A = \begin{bmatrix} .01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .01 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .01 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .01 \end{bmatrix}$$

```
function X = backsub(U,B) % MODIFIED BACKSUB
% BACKSUB X = backsub(U,b)
% Solve an upper triangular linear system.
% Input:
% U upper triangular square matrix (n by n)
% B right-hand side matrix (n by p)
% Output:
% X solution of UX=B (n by p matrix)
```

```
[n,p] = size(B);
X = zeros(n,p); % preallocate
    for j=1:p
        for i = n:-1:1 %since backward its n back to 1
        X(i,j) = ( B(i,j) - U(i,i+1:n)*X(i+1:n,j) ) / U(i,i);
        end
    end
end



% MODIFIED FORELIM
function X = forelim(L,B)
% FORELIM X = forelim(L,B)
% Solve a lower triangular linear system.
% Input:
% L lower triangular square matrix (n by n)
% B right-hand side matrix (n by p)
% Output:
% X solution of LX=B (n by p matrix)
[n,p]=size(B);
X=zeros(n,p); %preallocate
    for j=1:p
        for i=1:n %since forward its 1 to n
            X(i,j) = ( B(i,j) - L(i,1:i-1)*X(1:i-1,j) ) / L(i,i);
        end
    end
end



function X = ltinverse(L)
% LTINVERSE I = ltinverse(L)
% Solves the inverse of a lower triangular linear system.
% Input:
% L lower triangular square matrix (n by n)
% Output:
% I solution of the inverse of L (n by n matrix)
    n=length(L);
    I=eye(n); %grabs identity matrix
    X = forelim(L,I); %calls modified forelim to get inverse
end



function [L,U,P] = myplu(A)
% MYPLU [L,U,P] = myplu(A)
% Solves the PLU factorization for a given n by n matrix A.
% Input:
% A starting matrix (n by n)
% Output:
% L lower triangular matrix (n by n)
% U upper triangular matrix (n by n)
% P permutation matrix (n by n)
```

```
[n,n] = size(A);
P=eye(n);
    for k = 1:n-1
    % Find largest element below diagonal in k-th column
    [r,m] = max(abs(A(k:n,k)));
    m = m+k-1;
    % Skip elimination if column is zero
        if (A(m,k) ~= 0)
        % Swap pivot row
            if (m ~= k)
            A([k m],:) = A([m k],:);
            P([k m],:) = P([m k],:);
            end
        % Compute multipliers
        i = k+1:n;
        A(i,k) = A(i,k)/A(k,k);
        % Update the remainder of the matrix
        j = k+1:n;
        A(i,j) = A(i,j) - A(i,k)*A(k,j);
        end
    end
% Separate result
L = tril(A,-1) + eye(n,n);
U = triu(A);
end



function X = determinant(A)
    n = length(A);
    L = eye(n); % grabs identity matrix
    % Gaussian elimination
    for j = 1:n-1
        for i = j+1:n
        L(i,j) = A(i,j) / A(j,j); % row multiplier
        A(i,j:n) = A(i,j:n) - L(i,j)*A(j,j:n);
        end
    end
    U = triu(A);
    V = diag(U); %grabs diagonal values of U
    X = prod(V); %multiplies the diagonal values of U to get det.
end
```