Homework 5

Math 3607, Autumn 2021

Xiaoya Gao

Gao.1666

Table of Contents

Problem 1 (Improved triangular substitution)

Part(a)

Programs `backsub.m` and `forelim.m` are included at the end of the livescript.

Part(b)

If AX = I, then X = the inverse matrix of A. Then we can use the `forelim` function to find the inverse matrix of lower trangular matrix A by letting B be the n by n identty matrix.

Program `ltinverse` is included at the end of the live script. HW#5 Hint is used in solving this problem.

```
L1 = [2,0,0; 8,-7,0; 4,9,-27]; % L1 given
inv_L1 = [1/2,0,0; 4/7,-1/7,0; 50/189,-1/21,-1/27];  % inverse matrix given
Numinv_L1 = ltinverse(L1); % numerically calculated inverse of L1, using ltinverse
norm(inv_L1 - Numinv_L1) % gives the norm of the difference of the two
```

```
ans =
      0
```

```
L2 = [1,0,0,0; 1/3,1,0,0; 0,1/3,1,0; 0,0,1/3,1]; % L2 given
inv_L2 = [1,0,0,0; -1/3,1,0,0; 1/9,-1/3,1,0; -1/27,1/9,-1/3,1];
Numinv_L2 = ltinverse(L2); % numerically calculated inverse of L2, using ltinverse
norm(inv_L2 - Numinv_L2) % gives the norm of the difference of the two
```

```
ans =
      0
```

The norm of the difference seems to be zero for both lower triangular matrices. That's the way provided by the HW#5 hints. Comparing the numerical solution with the given exact solutions, we can find that they are basically the same.

```
format rational % puts the decimal elements in rational form; mentioned in lecture 1

Numinv_L1
```

```
Numinv_L1 =
       1/2             0              0
       4/7            -1/7            0
      50/189          -1/21          -1/27
```

```
Numinv_L2
```

```
Numinv_L2 =
       1               0              0              0
      -1/3             1              0              0
       1/9            -1/3            1              0
      -1/27            1/9           -1/3            1
```

Problem 2 (Triangular substitution and stability)

Part(a)

$$AX = \begin{bmatrix} 1 & -1 & 0 & \alpha-\beta & \beta \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

$$= \begin{bmatrix} x_1 - x_2 + (a-\beta)x_4 + \beta x_5 \\ 0x_1 + x_2 - x_3 + 0x_4 + 0x_5 \\ 0x_1 + 0x_2 + x_3 - x_4 + 0x_5 \\ 0x_1 + 0x_2 + 0x_3 + x_4 - x_5 \\ 0x_1 + 0x_2 + 0x_3 + 0x_4 + x_5 \end{bmatrix}$$

$$= \begin{bmatrix} x_1 - x_2 + (a-\beta)x_4 + \beta x_5 \\ x_2 - x_3 \\ x_3 - x_4 \\ x_4 - x_5 \\ x_5 \end{bmatrix}$$

Using matrix multiplication. So from $AX = B$ we have

$$\begin{cases} x_1 - x_2 + (a-\beta)x_4 + \beta x_5 = a \\ x_2 - x_3 = 0 \\ x_3 - x_4 = 0 \\ x_4 - x_5 = 0 \\ x_5 = 1 \end{cases}$$

From the system we get that $x_2 = x_3 = x_4 = x_5 = 1$.

The system of equations becomes

$$\begin{cases} x_1 + 1 + a - \beta + \beta = a \\ 1 - 1 = 0 \\ 1 - 1 = 0 \\ 1 - 1 = 0 \\ 1 = 1 \end{cases}$$

Here x_1 must satisfy that $x_1 - 1 + a - \beta + \beta = a$. So $x_1 = 1$.

Therefore, $X = (1, 1, 1, 1, 1)^T$ is the solution to the system.

Part(b)

```
for i = 10.^[1:12] % i = beta value
    a = [1,-1,0,(0.1-i),i; 0,1,-1,0,0; 0,0,1,-1,0; 0,0,0,1,-1; 0,0,0,0,1];
    x = a \ b; % compute x = a^-1 b

    j = abs(x(1,1)-1);
    fprintf(' %3.2e  %5.4f \n', i, j);
end
```

```
1.00e+01  0.0000
1.00e+02  0.0000
1.00e+03  0.0000
1.00e+04  0.0000
1.00e+05  0.0000
```

```
1.00e+06  0.0000
1.00e+07  0.0000
1.00e+08  0.0000
1.00e+09  0.0000
1.00e+10  0.0000
1.00e+11  0.0000
1.00e+12  0.0000
```

From (a) we can find that the elements of x must satisfy that $x_2 = x_3 = x_4 = x_5 = 1$,

so the euqation involving x_1 becomes $x_1 - 1 + \alpha - \beta + \beta = \alpha$, and after simplifying we get $x_1 - 1 = 0$.

Therefore, no matter what value α, β has, we will get $x_1 = 1$.

Thus the table above shows that `abs(x(1,1)-1)=0` for all β.

Problem 3 (Vectorizing `mylu.m`)

Part(a)

The modified function is placed at the end of the script.

After making the changes, `i` becomes the vector `[j+1, j+2, j+3, ..., n]`.

Every element `L(i,j)` and `A(i,j)` is accessed the same as being accessed in the for-loop.

To verify that the function works properly, we test the changed part with an random matirx A.

I wrote the following script to verify that the chaged function works the same.

```
type VerifyChange.m
```

```
format short

A1 = [1,2,3; 4,5,6; 7,8,9];  % random test matrix A1

% part of the function (unchanged)
n1 = length(A1);
L1 = eye(n1);
for j1 = 1:n1-1
    for i1 = j1+1:n1
        L1(i1,j1) = A1(i1,j1) / A1(j1,j1);
        A1(i1,j1:n1) = A1(i1,j1:n1) - L1(i1,j1)*A1(j1,j1:n1);
    end
end



A2 = [1,2,3; 4,5,6; 7,8,9];  % same random test matrix A2

% part of the function (changed)
n2 = length(A2);
L2 = eye(n2);
for j2 = 1:n2-1
    i2 = j2+1:n2;
    L2(i2,j2) = A2(i2,j2) / A2(j2,j2);
```

```
    A2(i2,j2:n2) = A2(i2,j2:n2) - L2(i2,j2)*A2(j2,j2:n2);

end
U1 = triu(A1)
U2 = triu(A2)
L1
L2
```

```
VerifyChange
```

```
U1 = 3×3
     1     2     3
     0    -3    -6
     0     0     0
U2 = 3×3
     1     2     3
     0    -3    -6
     0     0     0
L1 = 3×3
     1     0     0
     4     1     0
     7     2     1
L2 = 3×3
     1     0     0
     4     1     0
     7     2     1
```

We can see that the outputs are the same before and after the part of the function is modified. Thus we have varified that the function can work properly after we vectorize the group of operations.

Part(b)

In the iteration with $j=3$. the vector i becomes $[4\ \ 5]$. $(n=5)$.

$L([4\,5],3) = A([4\,5],3)\,/\,A(3,3)$ assigns $L(4,3) = \dfrac{A(4,3)}{A(3,3)}$.

and assigns $L(5,3) = \dfrac{A(5,3)}{A(3,3)}$ first.

$A(i,j{:}n)$ becomes $A([4\,5],[3\,4\,5])$, since $[j{:}n]=[3\,4\,5]$.

$A([4\,5],[3\,4\,5]) = A([4\,5],[3\,4\,5]) - L([4\,5],3)*A(3,[3\,4\,5])$ assigns

$A(4,3) = A(4,3) - L(4,3)*A(3,3)$,

$A(4,4) = A(4,4) - L(4,3)*A(3,4)$.

$A(4,5) = A(4,5) - L(4,3)*A(3,5)$.

$A(5,3) = A(5,3) - L(5,3)*A(3,3)$.

$A(5,4) = A(5,4) - L(5,3)*A(3,4)$,

$A(5,5) = A(5,5) - L(5,3)*A(3,5)$.

Problem 4 (Application of LU factorization)

Part(a)

We have learned in linear algebra that $\det(A) = \det(L)\det(U)$.

Also the determinant of triangular matrices $= t_{11}t_{22}\cdots t_{nn}$, for $T \in \mathbb{R}^{n\times n}$.

$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix}$ $\det(L) = 1$ using the formula above. It's actually 1 if we calculate $\det(L)$ using the cofactor and submatrix method.

$\det(L) = 1 \cdot \det \begin{vmatrix} 1 & 0 & 0 \\ l_{23} & 1 & 0 \\ \cdots & \cdots & 1 \end{vmatrix} + 0^{n-1}$, since the cofactors are all 0.

And also since the cofactors of the $n-1$ submatrices are all 1, we get $\boxed{\det(L) = 1^{n-1} + 0^{n-1} = l_{11} l_{22} l_{33} \cdots l_{nn} = 1.}$

$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$ use the last row as the cofactors for all submatrices, we get $\boxed{\det(U) = u_{11} u_{22} u_{33} \cdots u_{nn} = \prod_{i=1}^{n} u_{ii}.}$

Since $\det(L) = 1$ and $\det(U) = \prod_{i=1}^{n} u_{ii}$, we get that $\boxed{\det(A) = \det(L)\det(U) = \prod_{i=1}^{n} u_{ii}.}$

Part(b)

```
for n = 3:7
    M = magic(n);
    detM = determinant(M); % detM calculated using my function
    rel_err = (detM - det(M))/det(M); % relative error compare to det(M)
    if detM < 0
    fprintf('%d  %4.3e  %4.3f \n', n, detM, rel_err);
    else
    fprintf('%d  %5.4e  %4.3f \n', n, detM, rel_err);
    end

end
```

```
3  -3.600e+02  -0.000
4  3.6238e-13  -0.294
5  5.0700e+06  -0.000
6  0.0000e+00  -1.000
7  -3.481e+11  0.000
```

Problem 5 (Proper usage of `lu`)

$A = LU$, $Ax = b$, $x = A^{-1}b = (LU)^{-1}b = U^{-1}L^{-1}b$.

$x = U\backslash(L\backslash b)$ is the correct code to calculate $x = A^{-1}b$. that

$U\backslash(L\backslash b) = U\backslash(L^{-1}b) = U^{-1}L^{-1}b = A^{-1}b$,

whereas $\boxed{U\backslash L\backslash b = (U^{-1}L)\backslash b = (U^{-1}L)^{-1}b = L^{-1}Ub,}$

Since concecutive \ has left precedence.

Problem 6 (FLOP Counting)

Part(a)

```
x = A *(B *(C *(D*b)));
```

Here, u = D*b: ~$2n^2$ flops, C*(D*b) = C*u: ~$2n^2$ flops. So similarily, B*(C*(D*b)): ~$2n^2$ flops, and A*(B*(C*(D*b))): ~$2n^2$ flops.

In total, it takes <u>~$8n^2$ flops.</u>

Part(b)

```
[L,U,P] = lu(A);
x = B(U \(L \(P*b)));
```

Here lu(A): ~(2/3)n^3 flops, B*(U \(L \(P*b)): ~$8n^2$ flops. In total, it takes <u>~(2/3)n^3 flops.</u>

Part(c)

```
[L,U,P] = lu(C+A);
x = B(U \(L \(P*b)));
```

Here lu(C+A): ~(2/3)n^3 flops(neglecting the lower power terms), B*(U \(L \(P*b)): ~$8n^2$ flops. In total, it takes <u>~(2/3)n^3 flops.</u>

Problem 7 (Matrix norms)

Part(a)

$$\|A\|_1 = \max_{1\le j\le 2} \sum_{i=1}^{2} |a_{ij}| = \max_{1\le j\le 2}(1+0,\ 2+3) \boxed{= 5.}$$

$$\|A\|_2 = \sqrt{\lambda_{max}(A^T A)}.$$

Here $A^T A = \begin{bmatrix} 1 & 0 \\ 2 & 3 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 1+0 & 2+0 \\ 2+0 & 4+9 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 13 \end{bmatrix}.$

Finding the largest eigenvalue: $|A^T A - \lambda I| = \begin{vmatrix} 1-\lambda & 2 \\ 2 & 13-\lambda \end{vmatrix} = \lambda^2 - 14\lambda + 13 - 4 = 0.$

So the largest eigenvalue is $\lambda = \frac{14+\sqrt{160}}{2} = 7+2\sqrt{10}.$

$$\boxed{\|A\|_2 = \sqrt{7+2\sqrt{10}}.}$$

$$\|A\|_\infty = \max_{1\le i\le 2} \sum_{j=1}^{2} |a_{ij}| = \max_{1\le i\le 2}(1+2,\ 0+3) \boxed{= 3.}$$

$$\|A\|_F = \left(\sum_{i=1}^{2}\sum_{j=1}^{2} |a_{ij}|^2\right)^{1/2} = (1+13)^{1/2} = \boxed{\sqrt{14}.}$$

Part(b)

```
function X = MatrixNorm(A, j)
% MatrixNorm   computes matrix norms
% Usage:
%   mat_norm(A, 1) returns the 1-norm of A
%   mat_norm(A, 2) is the same as mat_norm(A)
%   mat_norm(A, 'inf') returns the infinity-norm of A
%   mat_norm(A, 'fro') returns the Frobenius norm of A

if j == 1
    X = max(sum(abs(A))); % largest value of the column sum

elseif j == 2
    t = A(:)'*A(:); % stores A^T A
    m = max(eig(t)); % finds the maximum eigenvalue
    X = sqrt(m);

elseif j == inf
    X = max(sum(abs(A),2)); % largest value of the row sum

elseif j == fro
    t = A(:)'*A(:); % stores A^T A
    X = sqrt(sum(diag));
end

end
```

Problem 1 functions

Function X = backsub(A,B):

```
function X = backsub(A,B)
```

```
% BACKSUB X = backsub(A,B)
% Solves and upper triangular linear system.
% Input: A (upper triangular square matrix, n-by-n)
%        B (right-hand side matirx, n-by-p)
% Output: X (solution of AX = B, n-by-p)

p = length(A); % number of columns
n = numel(A)/p; % number of rows
X = zeros(n,p); % preallocate

for j = (p:-1:1)
    for i = (n:-1:1)
        X(i,j) = (B(i,j) - A(i,i+1:n) * X(i+1:n,j)) / (A(i,i));
    end
end

end
```

Function X = forelim(A,B):

```
function X = forelim(A,B)
% FORELIM X = forelim(A,B)
% Solves and lower triangular linear system.
% Input: A (lower triangular square matrix, n by n)
%        B (right-hand side matrix, n by p)
% Output: X (solution of AX = B, n by p)

[n,p] = size(B); % gets the dimension of B
X = zeros(n,p); % preallocates the solution matirx

for j = (1:p)
    for i = (1:n)
        X(i,j) = (B(i,j) - A(i,1:i) * X(1:i,j)) / A(i,i);
    end
end

end
```

Function invL = ltinverse(L):

```
function invL = ltinverse(L)
% LTINVERSE invL = ltinverse(L)
% Computes the inverse matrix of the lower triangular matrix.
% Input: L (lower triangular square matrix, n-by-n)
% Output: invL (soution of A*invL=I, n-by-n)

n = length(L); % number of columns(/rows)

I = zeros(n,n); % preallocate right-hand side identity matrix, n-by-n
```

```
for i = (1:n)
    for j = (1:n)
        if j == i
            I(i,j) = 1; % assigns 1 to every i,i entry of the matirx
        end
    end
end

invL = forelim(L,I);
end
```

Problem 3 function

function [L,U] = mylu(A)

```
function [L,U] = mylu(A)
% MYLU   LU factorization
% Input: A (square matrix)
% Output: L (unit lower triangular)
%         U (unit upper triangular, LU=A)

    n = length(A);
    L = eye(n);   % ones on diagonal
    % Gaussian elimination
    for j = 1:n-1
        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
     U = triu(A);
end
```

Problem 4 function

```
function detA = determinant(A)
% DETERMINANT detA = determinant(A)
% finds the determinant of an input matirx A
% Input: A (square matrix, n by n)
% Output: detA (scalar, determinant of A)

[~,U] = mylu(A); % gets the upper triangular matrix

% computes the cumulative product of U's diagonals
proU = cumprod(diag(U));
detA = proU(end,1); % gets the cumulative product

end
```