Foundation of Computational Mathematics

Homework 1

ChenChen Zhou

1 Homework description

Problem review:

The case where $A \in R^{n \times n}$ is nonsingular and the system Ax = b is to be solved via LU factorization.

In this homework, we:

A Write a routine to do LU factorization without pivoting, partial pivoting or complete pivoting. Then using factorization result we solve system Ax = b. (Matlab file *'factorize.m'*)

B Write a another routine use output of routine above to test the correctness of our LU factorize routine and check the answer for solution of system Ax = b. (Matlab file *'TestRoutine.m'*)

C Use several small and large matrixs as input and then use test routine in B to test.

D Made a conlusion that our routine works well for 3 cases respectively based on test result.

2 Instruction for routine test

We have 6 matlab M-files:

A *"factorize.m"*

```
function [A,b,P,Q,flag,N,B,f]=factorize(A,flag,b)
```

We use A,b,N,flag as inputs, where A,b are parameters of system equation Ax = b and flag is the LU method we use (flag = 1, 2, 3 means we do LU factorization without pivoting, partial pivoting or complete pivoting respectively).

Among the Outputs, A is the LU factorized matrix (both saved in A). b is the solution of system Ax=b(we rewrite b). P,Q is the permutation Matrix we used and P(i)=j means we interchange i,j row, Q(i)=j means we interchange i,j column. flag and N are method number and dimension of A. B is the copy of original A. f is the copy of original b.

B *"TestRoutine.m"*

function error=TestRoutine(LU,x,P,Q,flag,N,B,f)

TestRoutine will use outputs of "factorize" as it's inputs.

Output error is the max value of 7 different error estimation error1,……, error7. Where,

error1 is the relative error between actual solution and routine solution $\frac{\|x-\tilde{x}\|}{\|x\|}$,where the norm is 2-Norm.

error2 is also $\frac{\|x-\tilde{x}\|}{\|x\|}$ but we take infinite norm.

error3 is the relative error of residual $\frac{\|b-A\tilde{x}\|}{\|b\|}$, we take 2-Norm.

error4 is also $\frac{\|b-A\tilde{x}\|}{\|b\|}$ but we take infinite norm.

error5 is the relative error of LU between PAQ ,thus $\frac{\|PAQ-LU\|}{\|A\|}$,which is measured the accuracy of the LU factorization. If we do without pivoting, P,Q and identity matrix, if we do partial pivoting, Q is identity matrix. We take 2-Norm.

error6 is also $\frac{\|PAQ-LU\|}{\|A\|}$ but we take 1-Norm this time.

error7 is still $\frac{\|PAQ-LU\|}{\|A\|}$ now we take infinite norm.

So, TestRoutine will test for $\frac{\|x-\tilde{x}\|}{\|x\|}$, $\frac{\|b-A\tilde{x}\|}{\|b\|}$, $\frac{\|PAQ-LU\|}{\|A\|}$ in different norm to check the correctness of function "factorize".

C *'lsolver'*

function f = lsolver(A,f)

This is a function used to solve system Ax=f, A is a lower triangle matrix,where diagnal is all 1 and denote as 0.

D *'usolver'*

function f = usolver(A,f)

This is a function used to solve system Ax=f, A is a upper triangle matrix.

E *'Swap'*

function b= Swap(P,b,N)

This is a function used to act permutation matrix P to column vector b, N is the size of b and P.

F *'TestCases'*

This is a function used to generate test matrixs and call routine "factorize" and "testroutine" automatically to test those matrixs.

If we get error is very small for each of the test matrix, we can say our routine works well for many cases.

How do we run the test

For each tested case, if we input matrix C_{nxn} and vector b_{nx1} ,then choose flag to run

[LU,b,P,Q,flag,N,B,f]=factorize(C,flag,b);

TestRoutine(LU,b,P,Q,flag,N,B,f);

in Matlab.

The output will be the error defined to be the max of the seven errors (error1 to error7) in TestRoutine M-files.

3 Test result and conclusion

In method without pivoting, partial pivoting or complete pivoting, we will test 10 matrixs for each of them. We will choose 2 small matrix, 10 50x50 uniformly distributed random matrixs and 10 100*100 uniformly distributed random matrix.

Particularly, those random distributed matrix have different domains of distribution,which means they have great probability to be good choices. And they will hardly fail our routine.

The result is posted below:

case	flag	style	size	error
1	1	class example	3x3	4.59E-16
2	2	class example	3x3	7.52E-16
3	3	class example	3x3	8.17E-16
4	1	class example	4x4	3.90E-16
5	2	class example	4x4	1.77E-15
6	3	class example	4x4	2.24E-15
7	1	uniform distributed	50x50	4.92E-13
8	2	uniform distributed	50x50	1.53E-13
9	3	uniform distributed	50x50	1.06E-13
10	1	uniform distributed	50x50	3.62E-14
11	2	uniform distributed	50x50	1.03E-14
12	3	uniform distributed	50x50	1.07E-14
13	1	uniform distributed	50x50	4.89E-14
14	2	uniform distributed	50x50	3.88E-15
15	3	uniform distributed	50x50	4.69E-15
16	1	uniform distributed	50x50	7.88E-14
17	2	uniform distributed	50x50	6.94E-15
18	3	uniform distributed	50x50	1.65E-14
19	1	uniform distributed	50x50	3.09E-13
20	2	uniform distributed	50x50	2.57E-14
21	3	uniform distributed	50x50	9.99E-15
22	1	uniform distributed	50x50	1.45E-13
23	2	uniform distributed	50x50	4.83E-15
24	3	uniform distributed	50x50	5.65E-15
25	1	uniform distributed	50x50	3.14E-12
26	2	uniform distributed	50x50	1.58E-14
27	3	uniform distributed	50x50	1.11E-14
28	1	uniform distributed	50x50	8.59E-14
29	2	uniform distributed	50x50	7.26E-15
30	3	uniform distributed	50x50	3.27E-15
31	1	uniform distributed	50x50	1.56E-13

32	2	uniform distributed	50x50	2.72E-14
33	3	uniform distributed	50x50	2.22E-14
34	1	uniform distributed	50x50	1.49E-12
35	2	uniform distributed	50x50	1.11E-14
36	3	uniform distributed	50x50	1.54E-14
37	1	uniform distributed	100x100	1.29E-12
38	2	uniform distributed	100x100	4.01E-14
39	3	uniform distributed	100x100	3.74E-14
40	1	uniform distributed	100x100	1.51E-12
41	2	uniform distributed	100x100	1.33E-14
42	3	uniform distributed	100x100	1.06E-14
43	1	uniform distributed	100x100	2.64E-11
44	2	uniform distributed	100x100	1.19E-13
45	3	uniform distributed	100x100	3.81E-14
46	1	uniform distributed	100x100	9.22E-13
47	2	uniform distributed	100x100	7.27E-15
48	3	uniform distributed	100x100	9.09E-15
49	1	uniform distributed	100x100	5.52E-11
50	2	uniform distributed	100x100	7.58E-13
51	3	uniform distributed	100x100	7.44E-13
52	1	uniform distributed	100x100	1.42E-11
53	2	uniform distributed	100x100	3.90E-14
54	3	uniform distributed	100x100	3.30E-14
55	1	uniform distributed	100x100	5.09E-11
56	2	uniform distributed	100x100	8.29E-14
57	3	uniform distributed	100x100	6.70E-14
58	1	uniform distributed	100x100	4.76E-13
59	2	uniform distributed	100x100	3.28E-14
60	3	uniform distributed	100x100	8.38E-15
61	1	uniform distributed	100x100	1.57E-12
62	2	uniform distributed	100x100	9.28E-15
63	3	uniform distributed	100x100	6.60E-15
64	1	uniform distributed	100x100	1.18E-11
65	2	uniform distributed	100x100	2.57E-14
66	3	uniform distributed	100x100	1.57E-14

Based on the result shown above, we can see that for all of our tested cases, the error is very small and so the routine for that case can be regarded as success.

Since our error is defined as the max of error1 to error 7 in TestRoutine M-file, which means each of the error is smaller than the error and so they are all very small.

And Since we computed 3 types of error $\frac{\|x-\tilde{x}\|}{\|x\|}$, $\frac{\|b-A\tilde{x}\|}{\|b\|}$, $\frac{\|PAQ-LU\|}{\|A\|}$, so if the error is small then it means both the LU factoriztion and system solution for Ax=b are successful.

So,we can now draw a conlusion that since for all the tested cases so far we did not fail any of them, and 60 of them are ramdomly generated, then our routine works well for both LU factoriztion and solving of system Ax=b.

Extra test for fail situation:

A

If we choose

C=[0 1 3 1;10 5 12 3;5 10 23 5;15 6 19 7];

b=[1 9 7 5]';

[LU,b,P,Q,flag,N,B,f]=factorize(C,1,b);

TestRoutine(LU,b,P,Q,flag,N,B,f);

```
>> C=[0 1 3 1;10 5 12 3;5 10 23 5;15 6 19 7]
b=[1 9 7 5]';
[LU,b,P,Q,flag,N,B,f]=factorize(C,1,b);
TestRoutine(LU,b,P,Q,flag,N,B,f);
??? Error using ==> factorize
divided by zero
```

Then notice c(1,1)=0, we get the error and program will exit.

B

Now if we input:

C=[0.00000001 1 3 1;10 5 12 3;5 10 23 5;15 6 19 7];

b=[1 9 7 5]';

[LU,b,P,Q,flag,N,B,f]=factorize(C,1,b);

TestRoutine(LU,b,P,Q,flag,N,B,f);

```
>> C=[0.00000001 1 3 1:10 5 12 3:5 10 23 5:15 6 19 7];
b=[1 9 7 5]';
[LU,b,P,Q,flag,N,B,f]=factorize(C,1,b);
TestRoutine(LU,b,P,Q,flag,N,B,f);
divided by too small number
```

We get . That means we get a warning that we divided too small error in case without pivoting. That will enlarge the error of the output, so we can see our program can warn us about that.

4 Matlab M-files

A *"factorize.m"*

```
function [A,b,P,Q,flag,N,B,f]=factorize(A,flag,b)
a=size(A);
if a(1)==a(2)
    N=a(1);
else
  error('not a square matrix');return;
end
f=b;
P=ones(1,N-1);
Q=ones(1,N-1);
e=0.00001;

%%%%%%%%%%%%%%%%%%%%%%%%
if flag==1
B=A;
for j=1:N-1
    if A(j,j)==0
         error('divided by zero');return;
    elseif abs(A(j,j))<=e
        fprintf('divided by too small number \n');
    end;
    for i=(j+1):N
       p=A(i,j)/A(j,j);
       A(i,j)=p;
       for k=(j+1):N
         A(i,k)=A(i,k)-p*A(j,k);
       end
    end
end
b=usolver(A,lsolver(A,b));
end

%%%%%%%%%%%%%%%%%%%%%%%%
if flag==2
B=A;
for j=1:N-1
        max=abs(A(j,j));
        max_i=j;
        for h=(j+1):N
            if abs(A(h,j))>max
              max=abs(A(h,j));
              max_i=h;
            end
        end

        P(j)=max_i;
        t=1;
        while (t<=N)
          temp=A(j,t);
          A(j,t)=A(max_i,t);
```

```
          A(max_i,t)=temp;
          t=t+1;
        end

        if A(j,j)==0
         error('divided by zero');return;
        elseif abs(A(j,j))<=0.00001
        fprintf('divided by too small number \n');
        end;

    for i=(j+1):N
      p=A(i,j)/A(j,j);
      A(i,j)=p;
      for k=(j+1):N
         A(i,k)=A(i,k)-p*A(j,k);
      end
    end
end

b=usolver(A,lsolver(A,Swap(P,b,N)));
end

%%%%%%%%%%%%%%%%%%%%%%%
if flag==3
B=A;
for j=1:N-1
        max=abs(A(j,j));
        max_i=j;
        max_j=j;

        for h=j:N
           for g=j:N
            if abs(A(h,g))>max
              max=abs(A(h,g));
              max_i=h;
              max_j=g;
            end
           end
        end

        P(j)=max_i
        Q(j)=max_j
        t=1;
        while (t<=N)
          temp=A(j,t);
          A(j,t)=A(max_i,t);
          A(max_i,t)=temp;
          t=t+1;
        end

         t=1;
        while (t<=N)
          temp=A(t,j);
          A(t,j)=A(t,max_j);
          A(t,max_j)=temp;
```

```
          t=t+1;
        end

        if A(j,j)==0
         error('divided by zero');return;
        elseif abs(A(j,j))<=0.00001
        fprintf('divided by too small number \n');
        end;

    for i=(j+1):N
      p=A(i,j)/A(j,j);
      A(i,j)=p;
      for k=(j+1):N
         A(i,k)=A(i,k)-p*A(j,k);
      end
    end
end

b=usolver(A,lsolver(A,Swap(P,b,N)));
t=N-1;
while (t>=1)
    temp=b(t);
    b(t)=b(Q(t));
    b(Q(t))=temp;
    t=t-1;
end

end
```

B *"TestRoutine.m"*

```
function err=TestRoutine(LU,x,P,Q,flag,N,B,f)

error(1)=norm(x-inv(B)*f)/norm(inv(B)*f);
error(2)=norm(x-inv(B)*f,inf)/norm(inv(B)*f);
error(3)=norm(B*x-f)/norm(f);
error(4)=norm(B*x-f,inf)/norm(f);

for i=1:N
   for j=1:N
    M(i,j)=0;
       if i>j
         for k=1:j
             M(i,j)=M(i,j)+LU(i,k)*LU(k,j);
         end
       else
           for k=1:i-1
             M(i,j)=M(i,j)+LU(i,k)*LU(k,j);
           end
            M(i,j)=M(i,j)+LU(i,j);
       end
   end
end
```

```
for j=1:N
for i=1:N-1
    if P(i)>i
    temp=B(i,j);
    B(i,j)=B(P(i),j);
    B(P(i),j)=temp;
    end
end
end

for i=1:N
    for j=1:N-1
       if Q(j)>j
    temp=B(i,j);
    B(i,j)=B(i,Q(j));
    B(i,Q(j))=temp;
      end
    end
end

error(5)=norm(B-M)/norm(B);
error(6)=norm(B-M,1)/norm(B);
error(7)=norm(B-M,inf)/norm(B);
err=norm(error,inf)
```

C *'lsolver'*

```
function f = lsolver(A,f)
a=size(A);
    N=a(1);
for i=2:N
  for j=1:i-1
     f(i)=f(i)-A(i,j)*f(j);
  end
end
```

D *'usolver'*

```
function f = solver(A,f)
a=size(A);
    N=a(1);
f(N)=f(N)/A(N,N);
for i=(N-1):-1:1
  for j=N:-1:(i+1)
     f(i)=f(i)-A(i,j)*f(j);
  end
  f(i)=f(i)/A(i,i);
end
```

E *'Swap'*

```
function  b= Swap(P,b,N)
for i=1:N-1
    if P(i)>i
    temp=b(i);
    b(i)=b(P(i));
    b(P(i))=temp;
    end
end
```

F *'TestCases'*

```
% Test for small matrix 1
C=[3 17 10;2 4 -2;6 18 -12];
for flag=1:3
b=[1 8 5]';
[LU,b,P,Q,flag,N,B,f]=factorize(C,flag,b);
TestRoutine(LU,b,P,Q,flag,N,B,f);
end

%Test for small matrix 2
C=[5 1 3 1;10 5 12 3;5 10 23 5;15 6 19 7];
for flag=1:3
b=[1 2 3 4]';
[LU,b,P,Q,flag,N,B,f]=factorize(C,flag,b);
TestRoutine(LU,b,P,Q,flag,N,B,f);
end

%Test for random matrix

for k=1:10
C=random('unif',10-k,10*k,50,50);
b=random('unif',-50-k,200+k*20,50,1);
for flag=1:3
[LU,b,P,Q,flag,N,B,f]=factorize(C,flag,b);
TestRoutine(LU,b,P,Q,flag,N,B,f);
end
end

for k=1:10
C=random('unif',-1000,1000,100,100);
b=random('unif',500,1000-5*k*k,100,1);
for flag=1:3
[LU,b,P,Q,flag,N,B,f]=factorize(C,flag,b);
TestRoutine(LU,b,P,Q,flag,N,B,f);
end
end
```