Foundations of Computational Math

Homework 2

Chenchen Zhou

1 Homework description

In this homework, we:

A Write a routine to do factorization of a column full rank matrix A_{nxk} base on householder reflector:

$$H_k H_{k-1} \cdots H_1 A = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

B We use that routine to find a solution of the least square problem:

$$\min_{x \in \mathbb{R}^k} \|b - Ax\|_2$$

For each case 1. n=k 2. n>k and b belongs to R(A) 3. n>k and b is not in R(A).

We write a routine separately to test our factorization and the correctness of the solution to the least square problem, particularly, for both known solution and unknown solution.

2 Alogrithm

We do not need to store the whole matrix, for each H_k, we only need to store a (n-k+1)*1 vector.

We will store it on the lower triangle part of H. And when we act H_k on H, we will not change the upper triangle part of H. And we will not use the kth colunmn of H any more after we finished step k.

So at last we will get a upper triangle matrix R stored in the upper triangle part of H. The n*1 vector stands for H1, we will store it on a extra vector. The rest vectors which stand for H_2 to H_k will be stored in the lower triangle part of H.

So actually we only need space of nxk+n to output our results.

1 Input A,b. let H=A, A will be copied and used for test. H will be used for output.

2 Compute H_k .(Actually we compute the vector which could generate H_k)

3 Every time when we compute Hk,we will act it on H and b.

4 At last we get H with R stored in the upper triangle part. And $c=H_kH_{k-1}...H_1b$

5 Compute $Xmin=R^{-1}c$. rewrite it in b.

6 output H, b(the Xmin)

3 Instruction for routine test

We have 2 matlab M-files.

1 Householder.M

Func tion[H,b]=Householder(A,b).

$$\min_{x\in\mathbb{R}^k}\|b-Ax\|_2$$

Input A,b is the parameter of . Output [H,b], H contains upper triangle of R, b is Xmin.

2 hw2testcase.M.

Hw2testcase(flag,type)

This is used to test our 3 situations.

For each case we test for known and unknown solution. And tested for at least 5 matrixs.

If flag=1, we test for n=k.

If flag=2, we test for n>k, b belongs to R(A). So it is consistent overdetermine equations

If flag=3, we test for n>k, b does not belong to R(A). So there is only Xmin to minimize our 2-norm of residual of the equation and no exact solution.

If type=1, we test for known solution. If type=2, we test for unknown solution.

Now, for

(flag=1,type=1),

Since A is column full rank.

We set A,Xmin randomly at first, compute b=AXmin. Then solve X*min,And we will check if

2-Norm(Xmin-X*min)=0.

(flag=1, type=2),

Set A,b randomly ,compute Xmin. We will check if 2-Norm(AXmin-b)=0.

(flag=2, type=1),

We set A,Xmin randomly at first, compute b=AXmin. Then solve X*min,And we will check if

2-Norm(Xmin-X*min)=0.

(flag=2, type=2),

We set A randomly, then randomly compute x to set b=Ax, then b belongs to R(A).

Then We will check if 2-Norm(AXmin-b)=0.

(flag=3, type=1),

For know solution, we will do exactly as what the instructor required.

We will set Xmin, Set Q, compute A=QM,

then compute b=b1+b2 by setting b=a1v1+a2v2 and v1=QQtv, v2=v-v1 and adjust v1 and v2 to be consistent.

Then compute the difference of 2-norm(rmin) and 2-norm(b2). It should be zero.

(flag=3, type=2).

In this case, we will Set A,b randomly, then compute Xmin.

Let rmin=b-A*Xmin, we will check if rTA=0.

We will also randomly pick x to see if rmin is the smallest among all the 2-norm of residuals ||b-Ax||.

At last we will use matlab rounting "regress" to see if Xmin and rmin we solved are equal to the solution computed by matlab rountine.

Test.M

This is used to run test case and chech the correctness of R.

4 Test result and conclusion

Before we test the routine. We will call matlab rountine [C,R]=qr(A) to see if our R is correct or not. We will run Test.M.

For each case we will randomly test different A (nxk matrix). And compare the actual R with the R we get. Result is posted below:

n	k	2-norm(RR-R)
50	50	8.14E-13
100	100	1.92E-12
150	150	2.76E-12
200	200	5.29E-12
250	250	6.98E-12
50	30	5.33E-13
100	60	1.13E-12
150	90	2.06E-12
200	120	2.52E-12
250	150	3.63E-12

So we can see our R is correct. Householder factorization works.

Then we need to see if the Xmin we get is the right solution.

Run

```
hw2testcase(1,1)
hw2testcase(1,2)
hw2testcase(2,1)
hw2testcase(2,2)
hw2testcase(3,1)
hw2testcase(3,2)
```

For each case we test 5 matrix randomly. From 50*50 to 250*250 for n=k, Or from 60*50 to 300*250 for n>k.

If our output results contain numbers very big, we fail.Otherwise,if all the output number we get are very small, we can say our rountine works for every situation and for each situation we can get the right Xmin as our solution.

We post the results as below:

flag,type	1,1	1,2	2,1	2,2	3,1	3,2
case1	5.12E-12	4.08E-12	1.97E-12	6.64E-10	-3.41E-13	1.04E-09
case2	3.07E-10	6.42E-11	1.08E-11	4.65E-09	-4.55E-13	7.12E-09
case3	7.39E-11	4.23E-11	1.94E-11	8.19E-09	9.09E-13	1.12E-08
case4	7.74E-10	5.98E-11	2.41E-11	1.95E-08	3.64E-12	2.27E-08
case5	1.97E-09	6.48E-11	3.43E-11	2.80E-08	2.73E-12	3.45E-08

Each case stands for different size of matrixs.

So we can make a conclusion that our solution is accurate in a range that we can tolerate. So our rountine really works.

5 Matlab M-files

For some part of householder.M. We have different version of code, for coding we will use a simple version. For presenting we will use tedious version since we can't simply use matlab matrix multiplication to implement our algorithm.

Householder.m
Version1

```
function [H,b]=Householder(A,b)
S=size(A);
n=S(1);
k=S(2);

if n<k
error('matrix does not meet requirement'); return;
end

H=A;
h=ones(n,1);

gama=sign(H(1,1))*norm(H(:,1));
h=A(:,1)+gama*eye(n,1);
a=-2/(norm(h).^2);
b=(eye(n)+a*h*h')*b;   %% change
H=(eye(n)+a*h*h')*H;   %% change

kk=k;
if n==k
   kk=k-1;
end

for i=2:kk
   m=n-i+1;
   gama=sign(H(i,i))*norm(H(i:n,i));

   H(i:n,i-1)=H(i:n,i)+gama*eye(m,1);   %% change

   a=-2/(norm(H(i:n,i-1)).^2);

   b(i:n)=(eye(m)+a*H(i:n,i-1)*H(i:n,i-1)')*b(i:n);   %% change
   H(i:n,i:k)=(eye(m)+a*H(i:n,i-1)*H(i:n,i-1)')*H(i:n,i:k);  %% change

end

b(k)=b(k)/H(k,k);
for i=(k-1):-1:1
```

```
 for j=k:-1:(i+1)
    b(i)=b(i)-H(i,j)*b(j);
 end
 b(i)=b(i)/H(i,i);
end
```

Householder.m
Version2

```
function [H,b]=Householder(A,b)
S=size(A);
n=S(1);
k=S(2);
if n<k
error('matrix does not meet requirement'); return;
end

H=A;
% use h to store extra vector. and compute H1 then act it on H.
h=ones(n,1);
gama=sign(H(1,1))*norm(H(:,1));
h=A(:,1)+gama*eye(n,1);
a=-2/(norm(h).^2);

temp=b;
for p=1:n
temp(p)=b(p);
for t=1:n
temp(p)=temp(p)+a*h(p)*h(t)*b(t);
end
end
b=temp;

for q=1:k
   temp=H(:,q);
 for p=1:n
    temp(p)=H(p,q);
    for t=1:n
       temp(p)=temp(p)+a*h(p)*h(t)*H(t,q);
    end
 end
 H(:,q)=temp;
end

kk=k;
if n==k
   kk=k-1;
end

for i=2:kk
   m=n-i+1;
   gama=sign(H(i,i))*norm(H(i:n,i));
```

```
   H(i:n,i-1)=H(i:n,i)+gama*eye(m,1);
   a=-2/(norm(H(i:n,i-1)).^2);

%   b(i:n)=(eye(m)+a*H(i:n,i-1)*H(i:n,i-1)')*b(i:n);   %% change

   temp=b;
   for p=i:n
temp(p)=b(p);
for t=i:n
temp(p)=temp(p)+a*H(p,i-1)*H(t,i-1)*b(t);
end
   end

b=temp;

for q=i:k
   temp=H(:,q);
 for p=i:n
    temp(p)=H(p,q);
    for t=i:n
       temp(p)=temp(p)+a*H(p,i-1)*H(t,i-1)*H(t,q);
    end
 end
 H(:,q)=temp;
end
%  H(i:n,i:k)=(eye(m)+a*H(i:n,i-1)*H(i:n,i-1)')*H(i:n,i:k);  %% change
end

% since the upper triangle R contained in H, we compute b=inv(R)*b to get the Xmin stored in x.
b(k)=b(k)/H(k,k);
for i=(k-1):-1:1
 for j=k:-1:(i+1)
   b(i)=b(i)-H(i,j)*b(j);
 end
 b(i)=b(i)/H(i,i);
end
```

hw2testcase.m

```
function hw2testcase(flag,type)

% n=k
if flag==1

% know solution
    if type==1
        for i=1:5
A=random('unif',-200,200,50*i,50*i);
x=random('unif',-500,500,50*i,1);
b=A*x;
[H,y]=Householder(A,b);
norm(x-y(1:50*i))
```

```
        end

% don't know solution
    elseif type==2
for i=1:5
A=random('unif',-200,200,50*i,50*i);
b=random('unif',-500,500,50*i,1);
[H,x]=Householder(A,b);

norm(b-A*x(1:50*i))
end
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% n>k b belong to R(A)
elseif flag==2

% know solution
if type==1

    for i=1:5
A=random('unif',-200,200,60*i,50*i);
x=random('unif',-500,500,50*i,1);
b=A*x;
[H,y]=Householder(A,b);
norm(x-y(1:50*i))
        end
% don't know solution
elseif type==2

        for i=1:5
A=random('unif',-200,200,60*i,50*i);
b=A*random('unif',200,500,50*i,1);
[H,x]=Householder(A,b);
norm(b-A*x(1:50*i))
        end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%555
% n>k b does not belong to R(A)
elseif flag==3

    if type==1
% know solution
    for i=1:5

A=random('unif',-200,200,60*i,50*i);
Q=orth(A);

A=Q*random('unif',-200,200,50*i,50*i);
v=random('unif',-500,500,60*i,1);
v1=Q*Q'*v;
v2=v-v1;
if norm(v1'*v2)>0.0000001
```

```
    error('not good');return;
end
a1=1;
a2=1;

while norm(a1*v1)/norm(a2*v2)<1 ||  norm(a1*v1)/norm(a2*v2)>5
a1=rand();
a2=rand();
end
b1=a1*v1;
b2=a2*v2;
b=b1+b2;

[H,x]=Householder(A,b);

norm(b-A*x(1:50*i))-norm(b2)
        end

% don't know solution
    elseif type==2
        for i=1:5
A=random('unif',-200,200,60*i,50*i);
b=random('unif',-500,500,60*i,1);
[H,x]=Householder(A,b);
r=A*x(1:50*i)-b;
e1=norm(r'*A);
min=1;
for j=1:100
y=random('unif',-500,500,50*i,1);
if norm(b-A*y)<norm(r)
min=0;
end
end
if min==0
    error('not the least square solution'); return;
end

[z,bint,rr] = regress(b,A);
e2=abs(norm(rr)-norm(r));
e3=norm(z-x(1:50*i));

maxe=e1;
if e2>maxe
    maxe=e2;
end
if e3>maxe
        maxe=e3;
end
maxe

        end
    end
end
```

test.m

```
for j=1:2
    for i=1:5
     A=random('unif',-100,100+20*i,50*i,50*i-10*i*(j-1));
        b=random('unif',-200-20*i,200,50*i,1);
     [C,R]=qr(A);
     [H,b]=Householder(A,b);
       RR=triu(H);
       norm(RR-R)
    end
end

hw2testcase(1,1)
hw2testcase(1,2)
hw2testcase(2,1)
hw2testcase(2,2)
hw2testcase(3,1)
hw2testcase(3,2)
```