Advanced Numerical Analysis Homework 4

Michael Nelson

Throughout this homework, $\|\cdot\|$ denotes the ℓ_2-norm.

1 Problem 1

Exercise 1. Suppose the $m \times n$ matrix A has the form

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$$

where A_1 is a nonsingular matrix of dimension $n \times n$ and A_2 is an arbitrary matrix of dimension $(m-n) \times n$. Prove that $\|A^+\| \leq \|A_1^{-1}\|$.

Solution 1. Note that A has full column-rank so all of its singular values are strictly positive. Let $\sigma_1 \geq \cdots \geq \sigma_n$ be the singular values of A. Then

$$\|A\| = \sigma_1, \quad \text{and} \quad \|A^+\| = \sigma_n^{-1}.$$

Next let $\tau_1 \geq \cdots \geq \tau_n$ be the singular values of A_1. Then $\|A_1\|^{-1} = \tau_n^{-1}$. Thus to show

$$\sigma_n^{-1} = \|A^+\| \leq \|A_1^{-1}\| = \tau_n^{-1},$$

it suffices to show that $\sigma_n \geq \tau_n$. This follows since

$$\sigma_n = \min_{x \neq 0} \frac{\|Ax\|}{\|x\|} \geq \min_{x \neq 0} \frac{\|A_1 x\|}{\|x\|} = \tau_n.$$

2 Problem 2

Exercise 2. Take $m = 50$ and $n = 12$. Using MATLAB's linspace, define t to be the m-vector corresponding to linearly spaced grid points from 0 to 1. Using MATLAB's vander and fliplr, define A to be the $m \times n$ matrix associated with least squares fitting on this grid by a polynomial of degree $n-1$. Take b to be the function $\cos(4t)$ evaluated on the grid. Now, calculate and print (to sixteen-digit precision) the least squares coefficient vector x by six methods:

1. Formation and solution of the normal equations, using MATLAB's \,
2. QR factorization computed by mgs;
3. QR factorization computed by house;
4. QR factorization computed by MATLAB's qr;
5. $x = A\backslash b$ in MATLAB;
6. SVD, using MATLAB's svd;
7. The calculations above will produce six lists of twelve coefficients. In each list, shade with red pen the digits that appear to be wrong (affected by rounding error). Comment on what differences you observe. Do the normal equations exhibit instability? You do not have to explain your observations.

Solution 2. We first set up our problem in MATLAB as the problem instructed:

```
m = 50;
n = 12;
t = linspace(0,1,m)';
A = fliplr(vander(t));
A = A(:,1:n);
b = cos(4*t);
x = zeros(n,6);
```

Now that the problem is set up, we solve (1-7):

1. We work in MATLAB below:

```
Bval = A'*A;
Rval = chol(Bval);
x(:,1) = Rval\(Rval'\(A'*b));
```

2. We work in MATLAB below:

```
[Qval,Rval] = mgs(A);
x(:,2) = backsubs(Rval,Qval'*b);
```

3. We work in MATLAB below:

4. We work in MATLAB below:

```
[Qval,Rval] = qr(A);
x(:,4) = backsubs(Rval,Qval'*b);
```

5. We work in MATLAB below:

```
x(:,5) = A\b ;
```

6. We work in MATLAB below:

```
[Uval,Sval,Vval] = svd(A,0);
c = Uval'*b;
for i=1:n
   c(i) = c(i)/Sval(i,i);
end;
x(:,6) = Vval*c;
```

7. We work in MATLAB below:

```
Bval = A'*A;
Rval = chol(Bval);
x(:,1) = Rval\(Rval'\(A'*b));
```

7. The normal equations exhibit instability.

3 Problem 3

Exercise 3. Implement Householder reduced QR factorization with column pivoting. At step k, consider columns k through n of the current A (has been up- dated in previous steps), find the column j ($k \leq j \leq n$) such that $\|A(k:m:j)\| = \max_{k \leq l \leq n} \|A(k:m:l)\|$, and switch columns k and j. Similar to GEPP, we need a permutation matrix P to record column swapping, such that $AP = QR$ numerically. Generate a new test matrix as follows:

```
U = randn(1024,10);
A4 = U*randn(10,15);
```

1. Test your code on A_2, A_3, and A_4, compare your upper triangular matrices with those generated by MATLAB's command [Q,R,P] = qr(A,0);
2. Show that the diagonal elements of R are monotonically decreasing in modulus.
3. Comment on the use of this algorithm to extract a set of numerically linearly independent columns from a matrix with numerically linearly dependent columns.

Solution 3.

4 Problem 4

Exercise 4.

Solution 4.

Appendix

Classical Gram-Schmidt

```
function [Q,R] = gs(A)
[m,n] = size(A);
Q = A;
R = zeros(n);
Q(:,1) = Q(:,1) / norm(Q(:,1)) ;
R(1,1) = Q(:,1)'*Q(:,1);
for j = 2:m
    for i = 1:j-1
       R(i,j) = Q(:,i)'*Q(:,j);
    end
    for i = 1:j-1
       Q(:,j) = Q(:,j) - R(i,j)*Q(:,i);
    end
    Q(:,j) = Q(:,j) / norm(Q(:,j)) ;
    R(j,j) = Q(:,j)'*Q(:,j) ;
    end
end
```

Modified Gram-Schmidt

```
function [Q,R] = mgs(A)
[m,n] = size(A);
Q = zeros(m,n);
R = zeros(n,n);
for j = 1:n
    Q(:,1) = A(:,1) / norm(A(:,1)) ;
    R(1,1) = Q(:,1)'*(A:,1);
    for j = 2:m
        for i = 1:j-1
           R(i,j) = Q(:,i)'*A(:,j);
        end
        Q(:,j) = A(:,j)
        for i = 1:j-1
```

```
            Q(:,j) = Q(:,j) - R(i,j)*Q(:,i);
        end
        Q(:,j) = Q(:,j) / norm(Q(:,j)) ;
        R(j,j) = Q(:,j)'*A(:,j) ;
    end
end
```

Double Modified Gram-Schmidt

```
function [Q,R] = qrmgs2(A)

[Q1,R1] = qrmgs(A);
[Q,R2] = qrmgs(Q1);
R = R2*R1;
```

Householder Factorization

```
function [v,R] = house(A)
[m,n] = size(A);
v = zeros(m,n);
for j = 1:n
    x = A(j:m,j);
    v(j:m,j) = x + sign(x(1))*norm(x)*eye(m-j+1,1);
    v(j:m,j) = v(j:m,j)/norm(v(j:m,j));
    A(j:m,j:n) = A(j:m,j:n) - 2*v(j:m,j)*(v(j:m,j)'*A(j:m,j:n));
end
R = A(1:n,:);
```