Math/CS 466/666 Programming Project 2

QR Factorization

Your work should be presented in the form of a typed report using clear and properly punctuated English. Where appropriate include full program listings and output. If you choose to work in a group of two, please turn in independently prepared reports.

1. A matrix $Q \in \mathbf{R}^{n\times n}$ is called an orthogonal matrix if $Q^tQ = I$. An orthogonal matrix Q and an upper triangular matrix R is called a QR factorization of A if $A = QR$. The Frobenius norm of a matrix A, denoted by $\|A\|_F$, is defined as

$$\|A\|_F = \Big(\sum_{i=1}^n \sum_{j=1}^n |A_{i,j}|^2\Big)^{1/2}.$$

The $n \times n$ Hilbert H matrix is defined as the matrix with entries

$$H_{ij} = 1/(i+j-1) \qquad \text{for} \qquad i,j = 1,2,\ldots,n.$$

(i) Suppose A is a non-singular $n\times n$ matrix with columns denoted by u_i. Apply Gram–Schmidt orthogonalization to the u_i's to obtain an orthonormal basis v_i. Let Q be the matrix with columns consisting of the vectors v_i and let $R = Q^tA$. Prove the matrix Q is orthogonal and that R is upper triangular.

Clearly $Q \in R^{n\times n}$. Since the v_i's form an orthonormal basis then

$$(v_i, v_j) = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{for } i \neq j. \end{cases}$$

It follows that

$$Q^tQ = \begin{bmatrix} v_1^t \\ \hline \vdots \\ \hline v_n^t \end{bmatrix} \begin{bmatrix} v_1 & | & \ldots & | & v_n \end{bmatrix} = \begin{bmatrix} (v_1,v_1) & \cdots & (v_1,v_n) \\ \vdots & \ddots & \vdots \\ (v_n,v_1) & \cdots & (v_n,v_n) \end{bmatrix} = I.$$

Therefore Q is an orthogonal matrix.

To see that R is upper triangular recall the Gram–Schmidt algorithm ensures that

$$\operatorname{span}\{\, u_i : i = 1,\ldots,k\,\} = \operatorname{span}\{\, v_i : i = 1,\ldots,k\,\} \qquad \text{for} \qquad k = 1,\ldots,n.$$

Therefore $(v_i, u_j) = 0$ for $i > j$. The fact that R is upper triangular now follows from

$$R = Q^tA = \begin{bmatrix} v_1^t \\ \hline \vdots \\ \hline v_n^t \end{bmatrix} \begin{bmatrix} u_1 & | & \ldots & | & u_n \end{bmatrix} = \begin{bmatrix} (v_1,u_1) & \cdots & (v_1,u_n) \\ \vdots & \ddots & \vdots \\ (v_n,u_1) & \cdots & (v_n,u_n) \end{bmatrix}. \qquad (*)$$

(ii) Suppose Q is an orthogonal matrix. Show that $QQ^t = I$.

Since $Q \in R^{n\times n}$ then $Q^tQ = I$ implies the columns of Q are orthonormal vectors. Since there are n columns, the columns of Q must form an orthonormal basis of $\mathbf{R}^n$.

Suppose $y \in \mathbf{R}^n$. As the columns of Q form an orthonormal basis, there is $x \in \mathbf{R}^n$ such that $y = Qx$. It follows that $Q^ty = Q^tQx = Ix = x$. Thus, $Q^ty = 0$ implies $x = 0$ and consequently that $y = Qx = 0$. Therefore $Q^ty = 0$ if any only if $y = 0$.

Now, since $I = Q^tQ$ then $Q^tI = Q^t = IQ^t = (Q^tQ)Q^t = Q^t(QQ^t)$. Consequently

$$Q^t(I - QQ^t)x = 0 \qquad \text{for every} \qquad x \in R^n.$$

As $y = 0$ is the only value of y such that $Q^ty = 0$, it follows that $(I - QQ^t)x = 0$ for every $x \in R^n$. This is identical as saying $QQ^t = I$.

(iii) [Extra Credit and Math/CS 666] Show that H is non-singular for any n.

Define

$$(f, g) = \int_0^1 f(x)g(x)dx \qquad \text{and} \qquad \|f\| = \sqrt{(f, f)}.$$

Note for any two polynomials p and q that $\|p - q\| = 0$ if and only if $p = q$. Therefore, the above definition yields a norm and inner product on the space of polynomials.

Let $a \in \mathbf{R}^n$ and consider the polynomial

$$p(x) = a_1 + a_2x + \cdots + a_nx^{n-1} = \sum_{i=1}^{n} a_ix^{i-1}.$$

Now

$$\begin{aligned}\|p\|^2 &= \int_0^1 p(x)p(x)dx = \int_0^1 \Big(\sum_{i=1}^{n} a_ix^{k-1}\Big)\Big(\sum_{j=1}^{n} a_jx^{j-1}\Big)dx \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\int_0^1 a_ia_jx^{i+j-2}dx = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{a_ia_j}{i+j-1}dx = a \cdot Ha.\end{aligned}$$

Therefore, if $Ha = 0$ then $\|p\| = 0$. Since this is a norm, that would imply $p = 0$, which in turn implies $a = 0$. It follows that H is injective and consequently invertible.

2. Let A be a matrix with columns given by u_i for $i = 1, \ldots, n$. Consider the following modification of the Gram–Schmidt orthogonalization algorithm:

$$\begin{aligned}
&t_{1,1} = u_1 && v_1 = t_{1,1}/\|t_{1,1}\| \\
&t_{1,2} = u_2 && \\
&t_{2,2} = t_{1,2} - (v_1, t_{1,2})v_1 && v_2 = t_{2,2}/\|t_{2,2}\| \\
&t_{1,3} = u_3 && \\
&t_{2,3} = t_{1,3} - (v_1, t_{1,3})v_1 && \\
&t_{3,3} = t_{2,3} - (v_2, t_{2,3})v_2 && v_3 = t_{3,3}/\|t_{3,3}\| \\
&t_{1,4} = u_4 && \\
&t_{2,4} = t_{1,4} - (v_1, t_{1,4})v_1 && \\
&t_{3,4} = t_{2,4} - (v_2, t_{2,4})v_2 && \\
&t_{4,4} = t_{3,4} - (v_3, t_{3,4})v_3 && v_4 = t_{4,4}/\|t_{4,4}\| \\
&\quad\vdots && \\
&t_{1,n} = u_n && \\
&t_{2,n} = t_{1,n} - (v_1, t_{1,n})v_1 && \\
&t_{3,n} = t_{2,n} - (v_2, t_{2,n})v_2 && \\
&\quad\ldots && \\
&t_{n,n} = t_{n-1,n} - (v_{n-1}, t_{n-1,n})v_{n-1} && v_n = t_{n,n}/\|t_{n,n}\|.
\end{aligned}$$

Let Q be the matrix with columns v_i and let R be the matrix with entries

$$R_{i,j} = \begin{cases} (v_i, t_{i,j}) & \text{for } i < j \\ \|t_{i,i}\| & \text{for } i = j \\ 0 & \text{for } i > j. \end{cases}$$

(i) Show that $\|t_{i,i}\| = (v_i, t_{i,i})$ and $(v_i, t_{i,j}) = (v_i, u_j)$ for $i \le j$.

Since $v_i = t_{i,i}/\|t_{i,i}\|$ by definition it follows that

$$(v_i, t_{i,i}) = \frac{(t_{i,i}, t_{i,i})}{\|t_{i,i}\|} = \frac{\|t_{i,i}\|^2}{\|t_{i,i}\|} = \|t_{i,i}\|.$$

Suppose there exists $i < j$ such that $v_i \cdot v_j \ne 0$. Let j_0 to be the smallest value for which such an i exists and further choose i_0 to be the smallest value of i such that $v_i \cdot v_{j_0} \ne 0$. Write $j_0 = i_0 + k$.

Since

$$\begin{aligned}
(v_i, t_{i+1,i+1}) &= \big(v_i, t_{i,i+1} - (v_i, t_{i,i+1})v_i\big) \\
&= (v_i, t_{i,i+1}) - (v_i, t_{i,i+1})(v_i, v_i) = 0,
\end{aligned}$$

then the fact that v_{i+1} is proportional to $t_{i+1,i+1}$ implies $(v_i, v_{i+1}) = 0$. We proceed using induction on k to show that $(v_i, v_{i+k}) = 0$ for $i = 1, \ldots, n-k$ when $k = 1, \ldots, n-1$.

Suppose $(v_i, v_{i+j}) = 0$ for all $i = 1, \ldots, n-j$ where $j = 1, \ldots, k$ and $k < n-1$. By hypothesis

$$\begin{aligned}
(v_i, t_{i+j+1,i+k+1}) &= \big(v_{i+j}, t_{i+j,i+k+1} - (v_{i+j}, t_{i+j,i+k+1})v_{i+j}\big) \\
&= (v_i, t_{i+j,i+k+1}) - (v_i, t_{i+j,i+k+1})(v_i, v_{i+j}) = (v_i, t_{i+j,i+k+1})
\end{aligned}$$

holds for $j = 1, \ldots, k$ where $i = 1, \ldots, n-k-1$. Therefore

$$(v_i, t_{i+k+1,i+k+1}) = (v_i, t_{i+1,i+k+1}).$$

Since

$$\begin{aligned}(v_i, t_{i+1,i+k+1}) &= \big(v_i, t_{i,i+k+1} - (v_i, t_{i,i+k+1})v_i\big)\\ &= (v_i, t_{i,i+k+1}) - (v_i, t_{i,i+k+1})(v_i, v_i) = 0\end{aligned}$$

the fact that $t_{i+k+1,i+k+1}$ is proportional to v_{i+k+1} implies $(v_{i+k+1}, v_i) = 0$, which completes the induction. In particular, the v_i's are orthonormal.

Since $t_{1,j} = u_j$, then $(v_1, t_{1,j}) = (v_1, u_j)$ for $j = 1, \ldots, n$. Now, if $1 < k \le i \le j$ then

$$\begin{aligned}(v_i, t_{k,j}) &= \big(v_i, t_{k-1,j} - (v_{k-1}, t_{k-1,j})v_{k-1}\big)\\ &= (v_i, t_{k-1,j}) - (v_{k-1}, t_{k-1,j})(v_i, v_{k-1}) = (v_i, t_{k-1,j}).\end{aligned}$$

Proceeding inductively obtains that $(v_i, t_{i,j}) = (v_i, t_{1,j}) = (v_i, u_j)$ for every $i \le j$.

(ii) Prove the matrix Q defined above is orthogonal and that $A = QR$.

We have already shown that that v_i is an orthonormal basis of $\mathbf{R}^n$ and that Q is an orthogonal matrix. In fact, since $(v_i, t_{i,j}) = (v_i, u_j)$, then the modified Gram–Schmidt algorithm is mathematically identical to the original Gram–Schmidt algorithm. Therefore the Q obtained by the modified algorithm is the the same as for the original algorithm. We also have $\|t_{i,i}\| = (v_i, t_{i,i}) = (v_i, u_j)$. Thus,

$$R_{i,j} = \begin{cases} (v_i, u_j) & \text{for } i \le j \\ 0 & \text{for } i > j. \end{cases}$$

From $(*)$ it follows that $R = Q^t A$. Finally, the fact that $QQ^t = I$ implies

$$QR = Q(Q^t A) = (QQ^t)A = IA = A.$$

(iii) Let H be the Hilbert matrix and write a program that computes Q and R such that $H = QR$ using the modified Gram–Schmidt algorithm described above.

We begin with a simple matrix library similar to the one created in the computing lab. Note that some functions have been renamed and other have been added. The library header file is

```
 /* p2lib.h--Simple Matrix Libraries for Programming Project 2
    Written November 15, 2016 by Eric Olson for Math/CS 466/666 */

#ifndef P2LIB_H
#define P2LIB_H

extern void matprintA(int n,double A[n][n]);
extern void matprintAt(int n,double A[n][n]);
extern double vecXtY(int n,double X[n],double Y[n]);
extern void matCisABt(int n,double C[n][n],double A[n][n],double B[n][n]);
extern void matCisAtB(int n,double C[n][n],double A[n][n],double B[n][n]);
extern void matCisAB(int n,double C[n][n],double A[n][n],double B[n][n]);
extern double frobI(int n,double A[n][n]);
extern double frobD(int n,double A[n][n],double B[n][n]);
extern void matset(int n,double A[n][n],double B[n][n]);
extern void mattrans(int n,double A[n][n]);
extern void matRQisA(int n,double R[n][n],double Q[n][n],double A[n][n]);

#endif
```

while the code for the library is

```
 /* p2lib.c--Simple Matrix Libraries for Programming Project 2
    Written November 15, 2016 by Eric Olson for Math/CS 466/666 */

```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "p2lib.h"

void matprintA(int n,double A[n][n]) {
    for(int i=0;i<n;i++) {
        for(int j=0;j<n;j++){
            printf("%8g%c",A[i][j],
            j==n-1?';':',');
        }
    printf("\n");
    }
}
void matprintAt(int n,double A[n][n]) {
    for(int i=0;i<n;i++) {
        for(int j=0;j<n;j++){
            printf("%18.10e%c",A[j][i],
            j==n-1?';':',');
        }
    printf("\n");
    }
}
double vecXtY(int n,double X[n],double Y[n]) {
    double r=0;
    for(int i=0;i<n;i++) r+=X[i]*Y[i];
    return r;
}
void matCisABt(int n,double C[n][n],double A[n][n],double B[n][n]) {
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) C[i][j]=0;
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) for(int k=0;k<n;k++)
        C[i][j]+=A[i][k]*B[j][k];
}
void matCisAtB(int n,double C[n][n],double A[n][n],double B[n][n]) {
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) C[i][j]=0;
    for(int k=0;k<n;k++) for(int i=0;i<n;i++) for(int j=0;j<n;j++)
        C[i][j]+=A[k][i]*B[k][j];
}
void matCisAB(int n,double C[n][n],double A[n][n],double B[n][n]) {
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) C[i][j]=0;
    for(int i=0;i<n;i++) for(int k=0;k<n;k++) for(int j=0;j<n;j++)
        C[i][j]+=A[i][k]*B[k][j];
}
double frobI(int n,double A[n][n]) {
    double r=0;
    for(int i=0;i<n;i++) for(int j=0;j<n;j++){
        register double x=A[i][j];
        if(i==j) x-=1;
        r+=x*x;
    }
    return sqrt(r);
}
double frobD(int n,double A[n][n],double B[n][n]) {
    double r=0;
    for(int i=0;i<n;i++) for(int j=0;j<n;j++){
        register double x=A[i][j]-B[i][j];
        r+=x*x;
    }
    return sqrt(r);
}
void matset(int n,double A[n][n],double B[n][n]) {
    for(int i=0;i<n;i++) for(int j=0;j<n;j++)
        A[i][j]=B[i][j];
}
void mattrans(int n,double A[n][n]) {
    for(int i=0;i<n;i++) for(int j=0;j<i;j++) {
        double t=A[i][j];
        A[i][j]=A[j][i];
        A[j][i]=t;
    }
}
void matRQisA(int n,double R[n][n],double Q[n][n],double A[n][n]) {

```

/* Since the C Programming Language stores matrices in row-major order, we switch the order of the indices and work with tranposed matrices in the code. In this way the column operations of the Gram–Schmidt algorithm become row operations for the transposed matrices. Thus, the code below actually performs the factorization $Q^tR^t = H^t$ or equivalently $RQ = H$. */

```

    for(int i=0;i<n-1;i++) for(int k=i+1;k<n;k++) R[i][k]=0;
    for(int i=0;i<n;i++){
        double r;
        for(int j=0;j<n;j++) Q[i][j]=A[i][j];
        r=vecXtY(n,Q[i],Q[i]);
        for(int k=0;k<i;k++) {
            r=vecXtY(n,Q[k],Q[i]);
            R[i][k]=r;
            for(int j=0;j<n;j++) Q[i][j]-=r*Q[k][j];
        }
        r=sqrt(vecXtY(n,Q[i],Q[i]));
        R[i][i]=r;
        for(int j=0;j<n;j++) Q[i][j]/=r;
    }
}
```

The main program now consists of a routine to generate the Hilbert matrix and perform the QR decomposition for the desired values of n.

```
 /* p2iii.c--Compute QR decomposition of the Hilbert Matrix
    Written November 15, 2016 by Eric Olson for Math/CS 466/666 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "p2lib.h"

void mathilb(int n,double H[n][n]) {
    for(int i=0;i<n;i++) for(int j=0;j<n;j++){
        H[i][j]=1.0/(1.0+i+j);
    }
}

#define N 4
double A[N][N],Q[N][N],R[N][N];
int main(){
    printf("#p2iii\n");
    mathilb(N,A);
    mattrans(N,A);
    matRQisA(N,R,Q,A);
    printf("Q=[\n");
    matprintAt(N,Q);
    printf("]\nR=[\n");
    matprintAt(N,R);
    printf("]\n");
    return 0;
}
```

(iv) Use the above program to find an orthogonal matrix Q and an upper triangular matrix R such that $H = QR$ when $n = 4$.

The output of the program is

```
#p2iii
Q=[
  8.3811635492e-01, -5.2264837396e-01,  1.5397276152e-01, -2.6306682088e-02;
  4.1905817746e-01,  4.4171332392e-01, -7.2775380737e-01,  3.1568018506e-01;
  2.7937211831e-01,  5.2882138625e-01,  1.3950552218e-01, -7.8920046265e-01;
  2.0952908873e-01,  5.0207166632e-01,  6.5360920576e-01,  5.2613364176e-01;
]
R=[
  1.1931517553e+00,  6.7049308394e-01,  4.7493260112e-01,  3.6983547090e-01;
  0.0000000000e+00,  1.1853326749e-01,  1.2565509463e-01,  1.1754199276e-01;
  0.0000000000e+00,  0.0000000000e+00,  6.2217740601e-03,  9.5660929494e-03;
  0.0000000000e+00,  0.0000000000e+00,  0.0000000000e+00,  1.8790487206e-04;
]
```

(v) For the matrices Q and R found using the above program compute

$$\|H - QR\|_F, \qquad \|Q^tQ - I\|_F \qquad \text{and} \qquad \|QQ^t - I\|_F$$

when $n = 4$, 6, 8, 10, 12 and 20.

Slight changes to the code from the previous section resulted in the program

```c
 /* p2iv.c--Compute QR decomposition of the Hilbert Matrix
    Written November 15, 2016 by Eric Olson for Math/CS 466/666 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "p2lib.h"

void mathilb(int n,double H[n][n]) {
    for(int i=0;i<n;i++) for(int j=0;j<n;j++){
        H[i][j]=1.0/(1.0+i+j);
    }
}

int main(){
    printf("#p2iv\n");
    int nvals[]={4, 6, 8, 10, 12, 20};
    printf("#%3s %22s %22s %22s\n","n","|QtQ-I|_F","|QQt-I|_F","|H-QR|_F");
    for(int i=0;i<sizeof(nvals)/sizeof(int);i++){
        int N=nvals[i];
        double A[N][N],Q[N][N],R[N][N],T[N][N];
        mathilb(N,A);
        mattrans(N,A);
        matRQisA(N,R,Q,A);
        mattrans(N,Q);
        mattrans(N,R);
        matCisAtB(N,T,Q,Q);
        double d1=frobI(N,T);
        matCisABt(N,T,Q,Q);
        double d2=frobI(N,T);
        matCisAB(N,T,Q,R);
        double d3=frobD(N,T,A);
        printf("%4d %22.14e %22.14e %22.14e\n",N,d1,d2,d3);
    }
    return 0;
}
```

The resulting output was

```
#p2iv
#  n             |QtQ-I|_F             |QQt-I|_F              |H-QR|_F
   4  4.05023225521046e-13  4.05051714551953e-13  2.77555756156289e-17
   6  2.72353370005502e-10  2.72353367319302e-10  6.35960131078450e-17
   8  7.33787259380259e-07  7.33787259425052e-07  1.11022302462516e-16
  10  2.55016714252341e-04  2.55016714252345e-04  1.16936318589005e-16
  12  4.38888436635874e-01  4.38888436635874e-01  1.08388987728284e-16
  20  2.37604463513680e+00  2.37604463513680e+00  1.69079929654581e-16
```

3. The LAPACK routine DGEQRFP computes the QR factorization of a matrix. After factorization, the upper triangular part of the matrix A is overwritten with R. The matrix Q may be obtained from DORGQR routine.

(i) Use these subroutines or equivalent ones to find an orthogonal matrix Q and an upper triangular matrix R such that $H = QR$ when $n = 4$.

The program

```
 /* p3i.c--Compute QR decomposition of the Hilbert Matrix using LAPACK
    Written November 15, 2016 by Eric Olson for Math/CS 466/666 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <lapacke.h>
#include "p2lib.h"

void mathilb(int n,double H[n][n]) {
    for(int i=0;i<n;i++) for(int j=0;j<n;j++){
        H[i][j]=1.0/(1.0+i+j);
    }
}

#define N 4
double Q[N][N],R[N][N],TAU[N];
int main(){
    printf("#p3i\n");
    mathilb(N,Q);
    mattrans(N,Q);
    LAPACKE_dgeqrfp(LAPACK_COL_MAJOR,N,N,Q[0],N,TAU);
    for(int i=0;i<N;i++) for(int j=0;j<N;j++){
        if(j<=i) R[i][j]=Q[i][j];
        else R[i][j]=0;
    }
    LAPACKE_dorgqr(LAPACK_COL_MAJOR,N,N,N,Q[0],N,TAU);
    printf("Q=[\n");
    matprintAt(N,Q);
    printf("]\nR=[\n");
    matprintAt(N,R);
    printf("]\n");
    return 0;
}
```

produces the output

```
#p3i
Q=[
  8.3811635492e-01, -5.2264837396e-01,  1.5397276152e-01, -2.6306682088e-02;
  4.1905817746e-01,  4.4171332392e-01, -7.2775380737e-01,  3.1568018506e-01;
  2.7937211831e-01,  5.2882138625e-01,  1.3950552218e-01, -7.8920046265e-01;
  2.0952908873e-01,  5.0207166632e-01,  6.5360920576e-01,  5.2613364176e-01;
]
R=[
  1.1931517553e+00,  6.7049308394e-01,  4.7493260112e-01,  3.6983547090e-01;
  0.0000000000e+00,  1.1853326749e-01,  1.2565509463e-01,  1.1754199276e-01;
  0.0000000000e+00,  0.0000000000e+00,  6.2217740601e-03,  9.5660929494e-03;
  0.0000000000e+00,  0.0000000000e+00,  0.0000000000e+00,  1.8790487206e-04;
]
```

(ii) Is the QR factorization found using LAPACK the same as the factorization found using the modified Gram–Schmidt algorithm in the previous question?

The output appears to be the same.

(iii) For the matrices Q and R found using LAPACK compute

$$\|H - QR\|_F, \qquad \|Q^tQ - I\|_F \qquad \text{and} \qquad \|QQ^t - I\|_F$$

when $n = 4$, 6, 8, 10, 12 and 20.

The program

```
 /* p3iii.c--Compute QR decomposition of the Hilbert Matrix using LAPACK
    Written November 15, 2016 by Eric Olson for Math/CS 466/666 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <lapacke.h>
#include "p2lib.h"

void mathilb(int n,double H[n][n]) {
    for(int i=0;i<n;i++) for(int j=0;j<n;j++){
        H[i][j]=1.0/(1.0+i+j);
    }
}

int main(){
    printf("#p3iii\n");
    int nvals[]={4, 6, 8, 10, 12, 20};
    printf("#%3s %22s %22s %22s\n","n","|QtQ-I|_F","|QQt-I|_F","|H-QR|_F");
    for(int i=0;i<sizeof(nvals)/sizeof(int);i++){
        int N=nvals[i];
        double A[N][N],Q[N][N],R[N][N],T[N][N],TAU[N];
        mathilb(N,A);
        mattrans(N,A);
        memcpy(Q,A,sizeof(double)*N*N);
        LAPACKE_dgeqrfp(LAPACK_COL_MAJOR,N,N,Q[0],N,TAU);
        for(int i=0;i<N;i++) for(int j=0;j<N;j++){
            if(j<=i) R[i][j]=Q[i][j];
            else R[i][j]=0;
        }
        LAPACKE_dorgqr(LAPACK_COL_MAJOR,N,N,N,Q[0],N,TAU);
        mattrans(N,Q);
        mattrans(N,R);
        matCisAtB(N,T,Q,Q);
        double d1=frobI(N,T);
        matCisABt(N,T,Q,Q);
        double d2=frobI(N,T);
        matCisAB(N,T,Q,R);
        double d3=frobD(N,T,A);
        printf("%4d %22.14e %22.14e %22.14e\n",N,d1,d2,d3);
    }
    return 0;
}
```

produces the output

```
#p3iii
#  n             |QtQ-I|_F              |QQt-I|_F               |H-QR|_F
   4  1.37677167724569e-15  1.29470816722684e-15  3.10316769155909e-16
   6  1.24564258906621e-15  1.16912542563685e-15  1.86190061493545e-16
   8  9.30497652132227e-16  8.82796670053343e-16  2.45915026490370e-16
  10  1.00435845246017e-15  1.06611484391197e-15  4.53742119551398e-16
  12  2.65012986360284e-15  2.55560000106286e-15  7.21311288194086e-16
  20  2.69041862515974e-15  2.61831256177488e-15  4.10085107424621e-16
```

(iv) Compare the accuracy of the results obtained using LAPACK with the accuracy of the results obtained in using the modified Gram–Schmidt algorithm.

The accuracy of the LAPACK routines are significantly better than the modified Gram–Schmidt algorithm.

(v) [Extra Credit and Math/CS 666] Look up the the algorithm used by LAPACK and explain how it works. What is special about the Hilbert matrix? Compare the speed and accuracy of the two methods when finding the QR factorization for random $n \times n$ matrices where $n = 10, 50, 100, 500, 1000$ and 2000.

The LAPACK routine is based on the method of Householder reflections. A description of this algorithm can be found in

[1] *Wikipedia, The Free Encyclopedia*, "QR Decomposition," accessed on November 19, 2016, `https://en.wikipedia.org/wiki/QR_decomposition`.

[2] *First Steps in Numerical Analysis*, R.J. Hosking, Hodder Arnold, 1978.

The basic idea is to construct a sequence of n Householder matrices of the form

$$H_k = I - 2w_k w_k^t \qquad \text{where} \qquad \|w\| = 1$$

such that

$$H_1 A = \begin{bmatrix} \alpha & * & \cdots & * \\ 0 & & & \\ \vdots & & A_1 & \\ 0 & & & \end{bmatrix}$$

where $\alpha = -\text{signum}(A_{1,1})\|u_1\|$ and u_1 is the first column of A. Note that $w_1 \in \mathbf{R}^n$ is a unit vector of length n with n degrees of freedom which needs to be chosen to create $n-1$ zeros upon multiplication. The sign of α is chosen by multiplying w_1 by ± 1 as needed. Note that the choice of sign for α increases the stability of the algorithm by maximizing the entry in the upper-left corner of the matrix H_1.

This procedure is then repeated inductively to find a vector $w_{k+1} \in \{0\}^k \times \mathbf{R}^{n-k}$ such that the the last $n-k$ components of w_{k+1} denoted by $\tilde{w}_{k+1} \in \mathbf{R}^{n-k}$ satisfy $\|\tilde{w}_{k+1}\| = 1$ and such that $(I - 2\tilde{w}_{k+1}\tilde{w}_{k+1}^t)A_k$ is an $(n-k) \times (n-k)$ matrix with zeros in all but the top corner of the first column. The QR decomposition may then be obtained by taking $Q = H_n H_{n-1} \cdots H_2 H_1$.

Some special things about the Hilbert matrix were found in

[3] *Wikipedia, The Free Encyclopedia*, "Hilbert matrix," accessed on November 19, 2016, `https://en.wikipedia.org/wiki/Hilbert_matrix`.

In particular, the Hilbert matrix is known to be non-singular, symmetric, positive definite and to have an inverse matrix consisting only of integer entries. Moreover, the condition number of the $n \times n$ Hilbert matrix is such that

$$\text{cond}(H) \sim K \frac{(1+\sqrt{2})^{4n}}{\sqrt{n}}.$$

Thus, the Hilbert matrix is ill-conditioned for relatively small values of n. This is why the Gram–Schmidt process failed to find a useful QR decomposition for $n = 20$.

The code

```
 /* p4.c--Compute QR decomposition of a Random Matrix in two ways
    Written November 15, 2016 by Eric Olson for Math/CS 466/666 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <lapacke.h>
#include <sys/time.h>
#include <sys/resource.h>
#include "p2lib.h"

static double tic_time;
void tic() {
    struct timeval tp;
    gettimeofday(&tp,0);
    tic_time=tp.tv_sec+tp.tv_usec*1.e-6;
}
double toc() {
    struct timeval tp;
    gettimeofday(&tp,0);
    return tp.tv_sec+tp.tv_usec*1.e-6-tic_time;
}

void matrand(int n,double A[n][n]) {
    srand(1234);
    for(int i=0;i<n;i++) for(int j=0;j<n;j++){
        A[i][j]=2.0*rand()/RAND_MAX-1.0;
    }
}

int main(){
    printf("#p4\n");
    setrlimit(RLIMIT_STACK,&(const struct rlimit){
        RLIM_INFINITY,RLIM_INFINITY});
    int nvals[]={10,50,100,500,1000,2000};
    printf("#%3s %18s %18s %18s %18s\n",
        "n","GS sec","GS |QtQ-I|_F",
        "LAPACK sec","LAPACK |QtQ-I|_F");
    for(int i=0;i<sizeof(nvals)/sizeof(int);i++){
        int N=nvals[i];
        double A[N][N],Q[N][N],R[N][N],T[N][N],TAU[N];
        matrand(N,A);
        mattrans(N,A);
```

```c
        tic();
        matRQisA(N,R,Q,A);
        double t1=toc();
        matCisABt(N,T,Q,Q);    // QtQ since column major
        double d1=frobI(N,T);

        tic();
        memcpy(Q,A,sizeof(double)*N*N);
        LAPACKE_dgeqrfp(LAPACK_COL_MAJOR,N,N,Q[0],N,TAU);
        for(int i=0;i<N;i++) for(int j=0;j<N;j++){
            if(j<=i) R[i][j]=Q[i][j];
            else R[i][j]=0;
        }
        LAPACKE_dorgqr(LAPACK_COL_MAJOR,N,N,N,Q[0],N,TAU);
        double t2=toc();
        matCisABt(N,T,Q,Q);    // QtQ since column major
        double d2=frobI(N,T);
        printf("%4d %18.10e %18.10e %18.10e %18.10e\n",N,t1,d1,t2,d2);
    }
    return 0;
}
```

produces the output

```
#p4
#  n            GS sec       GS |QtQ-I|_F        LAPACK sec   LAPACK |QtQ-I|_F
  10  5.9604644775e-06   1.7127857513e-14   2.6202201843e-04   1.2667751215e-15
  50  2.6392936707e-04   1.8314443720e-14   4.3702125549e-04   4.3156599481e-15
 100  1.7039775848e-03   5.5393586389e-13   1.2831687927e-03   8.0321821104e-15
 500  2.9938697815e-01   8.5169821271e-13   6.5408945084e-02   2.9375320935e-14
1000  2.2616238594e+00   3.6279737321e-12   4.3951296806e-01   4.9525951276e-14
2000  1.8388961077e+01   9.3367459484e-12   3.2365288734e+00   8.5636611542e-14
```

For reference, the above timings were performed using an AMD Athlon 64 X2 Dual Core 5400+ Processor with DDR2 800 RAM.

While the Modified Gram–Schmidt Method is faster for $n = 10$ and 50, for larger values of n the LAPACK routines are faster. In particular, LAPACK is 5.7 times faster when finding the QR decomposition for $n = 2000$. Although the rounding error is not so severe when working with random matrices and both methods yielded usable QR decompositions for all values of n, LAPACK achieved greater accuracy in all cases.