Math/CS 466/666: Programming Project 2

1. Let $A \in \mathbf{R}^{n\times n}$ and $C = AA^T$. If A is nonsingular does it follow that C must also be nonsingular? If so, explain why; if not, give a counter example.

Yes, it follows that C is also nonsingular. If A is non-singular than $\det(A) \neq 0$ and since $\det(A^T) = \det(A)$ then also $\det(A^T) \neq 0$. Consequently,

$$\det(C) = \det(AA^T) = \det(A)\det(A^T) \neq 0$$

shows that C is non-singular.

2. Show that $C^{-1} = (A^{-1})^T A^{-1}$ and conclude that $\|A^{-1}\|_2 = \sqrt{\rho(C^{-1})}$.

Since the transpose of the inverse is the inverse of the transpose we obtain

$$C^{-1} = (AA^T)^{-1} = (A^T)^{-1}A^{-1} = (A^{-1})^T A^{-1}.$$

We proved in class that

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

Now, replacing A by A^{-1} in the above equality yields

$$\|A^{-1}\|_2 = \sqrt{\rho\big((A^{-1})^T A^{-1}\big)} = \sqrt{\rho(C^{-1})}.$$

3. Consider the matrix $B = A^T A$. Do B^{-1} and C^{-1} have the same eigenvalues? If so, explain why; if not, give a counter example.

Let λ_i and ξ_i be the eigenvalues and eigenvectors of the matrix B such that

$$A^T A\xi_i = \lambda_i \xi_i \qquad \text{for} \qquad i = 1, 2, \ldots n.$$

Since B and C are invertible, it follows that A is invertible. Upon defining $\eta_i = A\xi_i$ we conclude that the η_i are linearly independent. Now

$$C\eta_i = AA^T A\xi_i = AB\xi_i = \lambda_i A\xi = \lambda_i \eta_i$$

shows that λ_i and η_i are eigenvalues and eigenvectors of the matrix C. Since the correspondence between ξ_i and η_i is one-to-one, it follows that B and C have the same eigenvalues.

Multiplying the equation $\lambda_i \xi_i = B\xi_i$ by B^{-1} and dividing by λ_i yields

$$B^{-1}\xi_i = \lambda_i^{-1}\xi_i \qquad \text{for} \qquad i = 1, 2, \ldots n.$$

Therefore B^{-1} has eigenvectors ξ_i with eigenvalues λ_i^{-1}. Similarly $C^{-1}\eta_i = \lambda_i^{-1}\eta_i$ for $i = 1, 2, \ldots n$. Therefore B^{-1} and C^{-1} have the same eigenvalues.

4. Modify the code in `matnorm2` by replacing `multAx` with `plusolve` to create a routine `invmatnorm2` which approximates $\|A^{-1}\|_2$. You will also need to add a call to `plufact` somewhere before the main loop. Test your routine using the matrix

$$A = \begin{bmatrix} 7 & 1 & 7 \\ 5 & 7 & 7 \\ 3 & 7 & 8 \end{bmatrix} \qquad \text{to show that} \qquad \|A^{-1}\|_2 \approx 0.73677.$$

Please include full program source code and output with your report.

I added code for the `invmatnorm2` routine

```
double invmatnorm2(int n,double A[n][n]){
    double B[n][n],*P[n],y[n],yk[n];
    bzero(B,sizeof(double)*n*n);
    for(int k=0;k<n;k++){                  // B = A^T A
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                B[i][j]+=A[k][i]*A[k][j];
            }
        }
    }
    for(int i=0;i<n;i++){                  // Choose x ∈ R^n randomly
        y[i]=2.0*random()/RAND_MAX+1.0;    // and store x in y for now
    }
    plufact(n,B,P);
    double q=0,qk;
    for(int k=1;k<100*n;k++){
        plusolve(n,B,P,yk,y);              // y_k = B^{-k}x/||B^{1-k}x||_2
        qk=vecnorm2(n,yk);
        for(int j=0;j<n;j++){
            y[j]=yk[j]/qk;                 // Overwrite y by y_k/||y_k||_2
        }
        if(fabs(qk-q)<5e-15*qk){           // Converge to 15 digits where
            return sqrt(qk);               // ||A^{-1}||_2 ≈ (||B^{-k}x||_2/||B^{1-k}x||_2)^{1/2}
        }
        q=qk;
    }
    fprintf(stderr,"invmatnorm2: Failed to converge!\n");
    return sqrt(qk);
}
```

to the matrix library files `matrixlib.c` and `matrixlib.h` from class. The final versions of these files are included in the appendix of this programming assignment for reference. The main program which calls this routine for the test matrix given above was

```
1 #include <stdio.h>
```

```
#include <stdlib.h>
#include "matrixlib.h"

double A[3][3]={{7,1,7},{5,7,7},{3,7,8}};
#define N 3

int main(){
    printf("Programming Project 2 Question 4.\n");
    printf("N=%d\n",N);
    printf("A=\n"); matprint(N,N,A);
    printf("||A^-1||_2=%.10g\n",invmatnorm2(N,A));
    return 0;
}
```

The resulting output is

```
Programming Project 2 Question 4.
N=3
A=
7 1 7
5 7 7
3 7 8
||A^-1||_2=0.7367677975
```

which shows that $\|A^{-1}\|_2 \approx 0.73677$.

5. Write a routine `matcond2` that computes the condition number

$$\text{cond}_2(A) = \|A^{-1}\|_2 \|A\|_2.$$

Approximate the condition number of the matrix given in the previous problem. Please include source code and output with your report.

Since $\text{cond}_2(A) = \|A\|_2\|A^{-1}\|_2$ and code to approximate $\|A\|_2$ and $\|A^{-1}\|_2$ have already been written, then only a few modifications to the previous code are needed to compute the condition number. I added code for the `invmatnorm2` routine

```
double matcond2(int n,double A[n][n]){
    return matnorm2(n,A)*invmatnorm2(n,A);
}
```

to the matrix library files `matrixlib.c` and `matrixlib.h` from class. The final versions of these files are included in the appendix of this programming assignment for reference. The main to call this routine for the the test matrix is

```
#include <stdio.h>
#include <stdlib.h>
#include "matrixlib.h"
```

```
4
5  double A[3][3]={{7,1,7},{5,7,7},{3,7,8}};
6  #define N 3
7
8  int main(){
9      printf("Programming Project 2 Question 5.\n");
10     printf("N=%d\n",N);
11     printf("A=\n"); matprint(N,N,A);
12     printf("||A||_2=%.10g\n",matnorm2(N,A));
13     printf("||A^-1||_2=%.10g\n",invmatnorm2(N,A));
14     printf("cond(A)=%.10g\n",matcond2(N,A));
15     return 0;
16 }
```

which produces the output

```
Programming Project 2 Question 5.
N=3
A=
7 1 7
5 7 7
3 7 8
||A||_2=17.7148954
||A^-1||_2=0.7367677975
cond(A)=13.05176446
```

6. Find $\|A\|_2$, $\|A^{-1}\|_2$ and $\text{cond}_2(A)$ for the special matrix associated with your netid available for download from the course website.

My matrix was given by

$$A = \begin{bmatrix} 0 & 5 & 5 & -7 & -1 & -3 & 7 \\ -8 & 2 & 8 & -5 & 6 & 7 & -5 \\ 5 & 8 & 3 & -7 & 0 & -9 & -1 \\ 9 & 5 & 6 & -3 & -4 & 7 & -5 \\ 3 & 8 & -9 & -7 & 3 & 6 & -5 \\ -8 & -6 & -7 & -7 & 5 & -9 & -3 \\ 1 & 8 & 1 & -3 & 7 & -6 & -1 \end{bmatrix}.$$

Changing the definition of A in the previous code obtains

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "matrixlib.h"
4
5  double A[7][7] = {
6      {  0,  5,  5, -7, -1, -3,  7 },
```

```
    { -8,  2,  8, -5,  6,  7, -5 },
    {  5,  8,  3, -7,  0, -9, -1 },
    {  9,  5,  6, -3, -4,  7, -5 },
    {  3,  8, -9, -7,  3,  6, -5 },
    { -8, -6, -7, -7,  5, -9, -3 },
    {  1,  8,  1, -3,  7, -6, -1 }};
#define N 7

int main(){
    printf("Programming Project 2 Question 5.\n");
    printf("N=%d\n",N);
    printf("A=\n"); matprint(N,N,A);
    printf("||A||_2=%.10g\n",matnorm2(N,A));
    printf("||A^-1||_2=%.10g\n",invmatnorm2(N,A));
    printf("cond(A)=%.10g\n",matcond2(N,A));
    return 0;
}
```

which produces the output

```
Programming Project 2 Question 5.
N=7
A=
0 5 5 -7 -1 -3 7
-8 2 8 -5 6 7 -5
5 8 3 -7 0 -9 -1
9 5 6 -3 -4 7 -5
3 8 -9 -7 3 6 -5
-8 -6 -7 -7 5 -9 -3
1 8 1 -3 7 -6 -1
||A||_2=22.55950187
||A^-1||_2=0.6838453902
cond(A)=15.42721136
```

Appendix: Complete Listing for Matrix Library

The file `matrixlib.h` is given by

```
#ifndef MATRIXLIB_H
#define MATRIXLIB_H

#include <complex.h>

extern void matprint(int m,int n,double A[m][n]);
extern void vecprint(int m,double X[m]);
extern void multAx(int m,int n,
    double A[m][n],double X[n],double B[m]);
extern void multATx(int m,int n,
    double A[m][n],double X[m],double B[n]);
extern void algob(int m,int n,
    double A[m][n],double X[n],double B[m]);
extern void cmatprint(int m,int n,complex A[m][n]);
extern void cvecprint(int m,complex X[m]);
extern void cmultAx(int m,int n,
    complex A[m][n],complex X[n],complex B[m]);
extern void calgob(int m,int n,
    complex A[m][n],complex X[n],complex B[m]);
extern void lufact(int n,double A[n][n]);
extern void backsub(int n,double U[n][n],
    double x[n],double y[n]);
extern void lusolve(int n,double LU[n][n],
    double x[n],double b[n]);
extern void plusolve(int n,double LU[n][n],double *P[n],
    double x[n],double b[n]);
extern void plufact(int n,double A[n][n],
        double *P[n]);
extern double vecnorm2(int n,double x[n]);
extern double matnorm2(int n,double A[n][n]);
extern double matcond2(int n,double A[n][n]);
extern double invmatnorm2(int n,double A[n][n]);
extern double vecnorm1(int n,double x[n]);
extern double matnorm1(int n,double A[n][n]);

#endif
```

The file `matrixlib.c` is given by

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <complex.h>
#include <math.h>
#include "matrixlib.h"

void matprint(int m,int n,double A[m][n]){
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            printf("%g ",A[i][j]);
        }
        printf("\n");
    }
}
void vecprint(int m,double X[m]){
    matprint(m,1,(double (*)[1])X);
}
void multAx(int m,int n,
    double A[m][n],double X[n],double B[m]){
    bzero(B,sizeof(double)*m);
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            B[i]+=A[i][j]*X[j];
        }
    }
}
void multATx(int m,int n,
    double A[m][n],double X[m],double B[n]){
    bzero(B,sizeof(double)*n);
    for(int j=0;j<m;j++){
        for(int i=0;i<n;i++){
            B[i]+=A[j][i]*X[j];
        }
    }
}
void algob(int m,int n,
    double A[m][n],double X[n],double B[m]){
    bzero(B,8*m);
    for(int j=0;j<n;j++){
        for(int i=0;i<m;i++){
            B[i]+=A[i][j]*X[j];
```

```
44          }
45      }
46  }
47  void cmatprint(int m,int n,complex A[m][n]){
48      for(int i=0;i<m;i++){
49          for(int j=0;j<n;j++){
50              printf("(%g %g) ",A[i][j]);
51          }
52          printf("\n");
53      }
54  }
55  void cvecprint(int m,complex X[m]){
56      cmatprint(m,1,(complex (*)[1])X);
57  }
58  void cmultAx(int m,int n,
59      complex A[m][n],complex X[n],complex B[m]){
60      bzero(B,sizeof(complex)*m);
61      for(int i=0;i<m;i++){
62          for(int j=0;j<n;j++){
63              B[i]+=A[i][j]*X[j];
64          }
65      }
66  }
67  void calgob(int m,int n,
68      complex A[m][n],complex X[n],complex B[m]){
69      bzero(B,8*m);
70      for(int j=0;j<n;j++){
71          for(int i=0;i<m;i++){
72              B[i]+=A[i][j]*X[j];
73          }
74      }
75  }
76
77  void lufact(int n,double A[n][n]){
78      for(int i=0;i<n-1;i++){
79          for(int j=i+1;j<n;j++){
80              double alpha=A[j][i]/A[i][i];
81              for(int k=i+1;k<n;k++){
82                  A[j][k]-=alpha*A[i][k];
83              }
84              A[j][i]=alpha;
85          }
86      }
87  }
```

```
88 void backsub(int n,double U[n][n],
89     double x[n],double y[n]){
90     for(int i=n-1;i>=0;i--){  // Ux=y
91         x[i]=y[i];
92         for(int j=i+1;j<n;j++){
93             x[i]-=U[i][j]*x[j];
94         }
95         x[i]/=U[i][i];
96     }
97 }
98 void lusolve(int n,double LU[n][n],
99     double x[n],double b[n]){
100    double y[n];
101    for(int i=0;i<n;i++){  // Ly=b
102        y[i]=b[i];
103        for(int j=0;j<i;j++){
104            y[i]-=LU[i][j]*y[j];
105        }
106    }
107    backsub(n,LU,x,y);
108 }
109 void plusolve(int n,double LU[n][n],double *P[n],
110    double x[n],double b[n]){
111    double y[n];
112    for(int i=0;i<n;i++){  // Ly=b
113        y[i]=b[(P[i]-&LU[0][0])/n];
114        for(int j=0;j<i;j++){
115            y[i]-=P[i][j]*y[j];
116        }
117    }
118    for(int i=n-1;i>=0;i--){  // Ux=y
119        x[i]=y[i];
120        for(int j=i+1;j<n;j++){
121            x[i]-=P[i][j]*x[j];
122        }
123        x[i]/=P[i][i];
124    }
125 }
126 void plufact(int n,double A[n][n],
127        double *P[n]){
128    for(int i=0;i<n;i++){
129        P[i]=&A[i][0];
130    }
131    for(int i=0;i<n-1;i++){
```

```
132         for(int j=i+1;j<n;j++){
133             if(fabs(P[i][i])<fabs(P[j][i])){
134                 double *t=P[i]; P[i]=P[j]; P[j]=t;
135             }
136         }
137         for(int j=i+1;j<n;j++){
138             double alpha=P[j][i]/P[i][i];
139             for(int k=i+1;k<n;k++){
140                 P[j][k]-=alpha*P[i][k];
141             }
142             P[j][i]=alpha;
143         }
144     }
145 }
146
147 double matnorm2(int n,double A[n][n]){
148     double B[n][n],y[n],yk[n];
149     bzero(B,sizeof(double)*n*n);
150     for(int k=0;k<n;k++){                  // B = A^T A
151         for(int i=0;i<n;i++){
152             for(int j=0;j<n;j++){
153                 B[i][j]+=A[k][i]*A[k][j];
154             }
155         }
156     }
157     for(int i=0;i<n;i++){                  // Choose x ∈ R^n randomly
158         y[i]=2.0*random()/RAND_MAX+1.0;    // and store x in y for now
159     }
160     double q=0,qk;
161     for(int k=1;k<100*n;k++){
162         multAx(n,n,B,y,yk);                // y_k = B^k x/||B^{k-1} x||_2
163         qk=vecnorm2(n,yk);
164         for(int j=0;j<n;j++){
165             y[j]=yk[j]/qk;                 // Overwrite y by y_k/||y_k||_2
166         }
167         if(fabs(qk-q)<5e-15*qk){           // Converge to 15 digits where
168             return sqrt(qk);               // ||A||_2 ≈ (||B^k x||_2/||B^{k-1} x||_2)^{1/2}
169         }
170         q=qk;
171     }
172     fprintf(stderr,"matnorm2: Failed to converge!\n");
173     return sqrt(qk);
174 }
175
```

```
double invmatnorm2(int n,double A[n][n]){
    double B[n][n],*P[n],y[n],yk[n];
    bzero(B,sizeof(double)*n*n);
    for(int k=0;k<n;k++){                  // B = A^T A
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                B[i][j]+=A[k][i]*A[k][j];
            }
        }
    }
    for(int i=0;i<n;i++){                  // Choose x ∈ R^n randomly
        y[i]=2.0*random()/RAND_MAX+1.0;    // and store x in y for now
    }
    plufact(n,B,P);
    double q=0,qk;
    for(int k=1;k<100*n;k++){
        plusolve(n,B,P,yk,y);              // y_k = B^{-k}x/||B^{1-k}x||_2
        qk=vecnorm2(n,yk);
        for(int j=0;j<n;j++){
            y[j]=yk[j]/qk;                 // Overwrite y by y_k/||y_k||_2
        }
        if(fabs(qk-q)<5e-15*qk){           // Converge to 15 digits where
            return sqrt(qk);               // ||A^{-1}||_2 ≈ (||B^{-k}x||_2/||B^{1-k}x||_2)^{1/2}
        }
        q=qk;
    }
    fprintf(stderr,"invmatnorm2: Failed to converge!\n");
    return sqrt(qk);
}

double matcond2(int n,double A[n][n]){
    return matnorm2(n,A)*invmatnorm2(n,A);
}

double vecnorm2(int n,double x[n]){
    double r=0;
    for(int i=0;i<n;i++){
        double t=x[i];
        r+=t*t;
    }
    return sqrt(r);
}

double vecnorm1(int n,double x[n]){
```

```
220     double r=0;
221     for(int i=0;i<n;i++){
222         r+=fabs(x[i]);
223     }
224     return r;
225 }
226 
227 double matnorm1(int n,double A[n][n]){
228     double r=0;
229     for(int j=0;j<n;j++){
230         double s=0;
231         for(int i=0;i<n;i++){
232             s+=fabs(A[i][j]);
233         }
234         if(s>r) r=s;
235     }
236     return r;
237 }
238 
```