Math/CS 466/666 Homework 1 Solutions

1. Consider the vectors and matrices

$$u = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} 1 & -1 & 3 \\ 4 & 0 & 2 \\ -1 & 2 & 0 \end{bmatrix}$$

(i) Find $\|u\|_1$ and $\|A\|_1$ and verify that $\|Au\|_1 \leq \|A\|_1 \|u\|_1$.

Compute $\|u\|_1$ as

$$\|u\|_1 = \sum_{n=1}^{3} |u_n| = |1| + |2| + |3| = 6.$$

Compute $\|A\|_1$ as

$$\begin{aligned} \|A\|_1 &= \max_{1 \leq j \leq 3} \sum_{i=1}^{3} |a_{ij}| \\ &= \max\{\, |1| + |4| + |-1|, |-1| + |0| + |2|, |3| + |2| + |0| \,\} \\ &= \max\{6, 3, 5\} = 6. \end{aligned}$$

Compute Au as

$$Au = \begin{bmatrix} 1 & -1 & 3 \\ 4 & 0 & 2 \\ -1 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 - 2 + 9 \\ 4 + 0 + 6 \\ -1 + 4 + 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 10 \\ 3 \end{bmatrix}.$$

Therefore

$$\|Au\|_1 = \sum_{n=1}^{3} [Au]_n = |8| + |10| + |3| = 21.$$

The inequality $\|Au\|_1 \leq \|A\|_1 \|u\|_1$ is verified in this case, since

$$\|A\|_1 \|u\|_1 = 6 \cdot 6 = 36 \geq 21.$$

(ii) Find $\|u\|_\infty$ and $\|A\|_\infty$ and verify that $\|Au\|_\infty \leq \|A\|_\infty \|u\|_\infty$.

Compute $\|u\|_\infty$ as

$$\|u\|_\infty = \max\{\,|1|, |2|, |3|\,\} = 3.$$

Compute $\|A\|_\infty$ as

$$\begin{aligned}\|A\|_\infty &= \max_{1\leq i\leq 3} \sum_{j=1}^{3} |a_{ij}| \\ &= \max\{\,|1| + |-1| + |3|, |4| + |0| + |2|, |-1| + |2| + |0|\,\} \\ &= \max\{5, 6, 3\} = 6.\end{aligned}$$

Finally

$$\|Au\|_\infty = \left\| \begin{bmatrix} 8 \\ 10 \\ 3 \end{bmatrix} \right\|_\infty = \max\{|8|, |10|, |3|\} = 10.$$

The inequality $\|Au\|_\infty \leq \|A\|_\infty \|u\|_\infty$ is verified in this case, since

$$\|A\|_\infty \|u\|_\infty = 6 \cdot 3 = 18 \geq 10.$$

(iii) Compute $A^\dagger A$.

Computing $B = A^\dagger A$ we obtain

$$\begin{aligned} B &= \begin{bmatrix} 1 & 4 & -1 \\ -1 & 0 & 2 \\ 3 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 & -1 & 3 \\ 4 & 0 & 2 \\ -1 & 2 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1+16+1 & -1+0-2 & 3+8+0 \\ -1+0-2 & 1+0+4 & -3+0+0 \\ 3+8+0 & -3+0+0 & 9+4+0 \end{bmatrix} = \begin{bmatrix} 18 & -3 & 11 \\ -3 & 5 & -3 \\ 11 & -3 & 13 \end{bmatrix}. \end{aligned}$$

(iv) Prove the eigenvalues of $A^\dagger A$ are real and non-negative.

Recall that $A^\dagger = \overline{A^T}$ is the conjugate transpose of the matrix A. Therefore $(A^\dagger)^T = \bar{A}$. To see that λ is real and non-negative we write

$$\lambda \|\xi\|_2^2 = \lambda \xi \cdot \bar{\xi} = B\xi \cdot \bar{\xi} = A^\dagger A \xi \cdot \bar{\xi} = A\xi \cdot \overline{A\xi} = \|A\xi\|_2^2 \geq 0.$$

(v) Use the power method to find the largest eigenvalue λ_1 such that $A^\dagger Ax = \lambda_1 x$ for some non-zero eigenvector x.

The C program

```
/*  part5.c -- Power method to find largest eigenvalue of At*A
    Written October 3, 2014 by Eric Olson */

#include <stdio.h>
#include <string.h>

double A[3][3]={{1,-1,3},{4,0,2},{-1,2,0}};

void matprint(int n,double A[n][n]){
    for(int i=0;i<n;i++) for(int j=0;j<n;j++)
        printf("%g%c",A[i][j],j==n-1?'\n':' ');
}

double dotprod(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;
}

int main(){
    printf(
"part5 -- Power method to find largest eigenvalue of At*A\n"
"Written October 3, 2014 by Eric Olson\n");
    int n=sizeof(A[0])/sizeof(double);
    printf("\nA=\n"); matprint(n,A);
    double B[n][n];
    memset(B,0,n*n*sizeof(double));
    for(int k=0;k<n;k++) for(int i=0;i<n;i++) for(int j=0;j<n;j++)
        B[i][j]+=A[k][i]*A[k][j];
    printf("\nAt*A=\n"); matprint(n,B);
    double x[n],y[n];
    for(int i=0;i<n;i++) x[i]=1;
    printf("\n%5s %24s\n","k","lambda_max");
    for(int k=0;k<10;k++){
        memset(y,0,n*sizeof(double));
        for(int i=0;i<n;i++) for(int j=0;j<n;j++)
            y[i]+=B[i][j]*x[j];
        double xx=dotprod(n,x,x);
        double lambda=dotprod(n,y,x)/xx;
        for(int i=0;i<n;i++) x[i]=y[i]/xx;
        printf("%5d %24.15e\n",k+1,lambda);
    }
    return 0;
}
```

first multiplies $B = A^T A$ and then uses 10 iterations of the power method to find the largest eigenvalue of B. Note that `double` variable types are used throughout instead of `complex` because it is known from part (iv) that all eigenvalues are real.

The output of the program is

```
part5 -- Power method to find largest eigenvalue of At*A
Written October 3, 2014 by Eric Olson

A=
1 -1 3
4 0 2
-1 2 0

At*A=
18 -3 11
-3 5 -3
11 -3 13

    k               lambda_max
    1    1.533333333333333e+01
    2    2.701252236135957e+01
    3    2.755519034560535e+01
    4    2.756817719081595e+01
    5    2.756849552118013e+01
    6    2.756850365659541e+01
    7    2.756850387184427e+01
    8    2.756850387769260e+01
    9    2.756850387785459e+01
   10    2.756850387785914e+01
```

This implies that $\lambda_1 \approx 27.5685$.

(vi) Find $\|u\|_2$ and compute $\|A\|_2$ using the formula $\|A\|_2 = \sqrt{\lambda_1}$.

Compute $\|u\|_2$ as

$$\|u\|_2 = \Big(\sum_{n=1}^{3} |u_n|^2 \Big)^{1/2} = \big(1^2 + 2^2 + 3^2\big)^{1/2} = \sqrt{14} \approx 3.74.$$

Compute $\|A\|_2$ as

$$\|A\|_2 = \sqrt{\rho(A^\dagger A)} = \sqrt{\lambda_1} \approx \sqrt{27.5685} \approx 5.25.$$

(vii) Verify that $\|Au\|_2 \leq \|A\|_2 \|u\|_2$.

We have

$$\|Au\|_2 = \left\| \begin{bmatrix} 8 \\ 10 \\ 3 \end{bmatrix} \right\| = \sqrt{8^2 + 10^2 + 3^2} \approx 13.153.$$

The inequality $\|Au\|_2 \leq \|A\|_2 \|u\|_2$ is verified in this case, since

$$\|A\|_2 \|u\|_2 \approx (5.25)(3.74) \approx 19.63 \geq 13.153.$$

(viii) Use the inverse power method to find the smallest eigenvalue λ_3 of $A^\dagger A$.

We modify the program in part (v) to solve $Ay = x$ for y at each iteration rather than setting $y = Ax$ as before. The resulting program is

```
/*  part8.c -- Inverse power to find smallest eigenvalue of At*A
    Written October 3, 2014 by Eric Olson */

#include <stdio.h>
#include <string.h>
#include "solveAx.h"

double A[3][3]={{1,-1,3},{4,0,2},{-1,2,0}};

void matprint(int n,double A[n][n]){
    for(int i=0;i<n;i++) for(int j=0;j<n;j++)
        printf("%g%c",A[i][j],j==n-1?'\n':' ');
}

double dotprod(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;
}

int main(){
    printf(
"part8 -- Inverse power to find smallest eigenvalue of At*A\n"
"Written October 3, 2014 by Eric Olson\n");
    int n=sizeof(A[0])/sizeof(double);
    printf("\nA=\n"); matprint(n,A);
    double B[n][n];
    memset(B,0,n*n*sizeof(double));
    for(int k=0;k<n;k++) for(int i=0;i<n;i++) for(int j=0;j<n;j++)
        B[i][j]+=A[k][i]*A[k][j];
    printf("\nAt*A=\n"); matprint(n,B);
    double *PLU[n];
    PLUfact(n,B,PLU);
    double x[n],y[n];
    for(int i=0;i<n;i++) x[i]=1;
    printf("\n%5s %24s\n","k","lambda_min");
    for(int k=0;k<10;k++){
        PLUsolve(n,PLU,y,x);
        double xx=dotprod(n,x,x);
        double lambda=dotprod(n,y,x)/xx;
        for(int i=0;i<n;i++) x[i]=y[i]/xx;
        printf("%5d %24.15e\n",k+1,1/lambda);
    }
    return 0;
}
```

Note we have used the simple Gaussian elimination routine `solveAx.c` that was discussed in the Saturday computing workshop. It would also be possible to use the LAPACK linear algebra solver, however, for a 3×3 matrix the simple routine is just as fast.

The library header `solveAx.h` is given by

```
#ifndef _SOLVEAX_H
#define _SOLVEAX_H

extern void PLUfact(int n, double A[n][n], double *PLU[n]);
extern void PLUsolve(int n, double *PLU[n], double x[n], double b[n]);

#endif
```

and the Gaussian eliminations routines `solveAx.c` are

```
/*  solveAx.c -- Gaussian Elimination Using Partial Pivoting
    Written September 2014 by Eric Olson for Math/CS 466/666 */

#include <math.h>
#include "solveAx.h"

// Make PLU factorization of matrix A
void PLUfact(int n, double A[n][n], double *PLU[n]){
    for(int i=0;i<n;i++) PLU[i]=A[i];
    for(int j=0;j<n;j++){
        for(int i=j+1;i<n;i++){
            if(fabs(PLU[j][j])<fabs(PLU[i][j])) {
                double *t=PLU[i]; PLU[i]=PLU[j]; PLU[j]=t;
            }
        }
        for(int i=j+1;i<n;i++){
            double r=PLU[i][j]/PLU[j][j];
            for(int k=j;k<n;k++) PLU[i][k]-=r*PLU[j][k];
            PLU[i][j]=r;
        }
    }
}

// Solve PLU x = b using back substitution
void PLUsolve(int n, double *PLU[n], double x[n], double b[n]){
    double *A0=PLU[0];
    for(int i=1;i<n;i++) if(PLU[i]<A0) A0=PLU[i];
    /*  The row with the smallest memory address
        was the first row of the original matrix */
    for(int i=0;i<n;i++) x[i]=b[(PLU[i]-A0)/n];
    for(int i=1;i<n;i++){
        for(int j=0;j<i;j++) x[i]-=PLU[i][j]*x[j];
    }
    for(int i=n-1;i>=0;i--){
        for(int j=i+1;j<n;j++) x[i]-=PLU[i][j]*x[j];
        x[i]/=PLU[i][i];
    }
}
```

The output from the program is

```
part8 -- Inverse power to find smallest eigenvalue of At*A
Written October 3, 2014 by Eric Olson

A=
1 -1 3
4 0 2
-1 2 0

At*A=
18 -3 11
-3 5 -3
11 -3 13

    k               lambda_min
    1    6.849056603773585e+00
    2    4.036493427352660e+00
    3    3.905822413636077e+00
    4    3.855670334573589e+00
    5    3.820919652918088e+00
    6    3.797039814543667e+00
    7    3.780995554135172e+00
    8    3.770383178586949e+00
    9    3.763436242122769e+00
   10    3.758919608300943e+00
```

which implies that $\lambda_3 \approx 3.7589$.

(ix) Prove or disprove that the condition number

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \sqrt{\lambda_1/\lambda_3}.$$

This is true. By definition $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$. By part (vi) we already have that $\|A\|_2 = \sqrt{\lambda_1}$. It remains to show that $\|A^{-1}\|_2 = \sqrt{\lambda_3}$.

Define $C = (A^{-1})^\dagger A^{-1}$ and let σ_1 be the largest eigenvalue of C. Then again by part (vi) we have that $\|A^{-1}\|_2 = \sqrt{\sigma_1}$. It remains to show that $\sigma_1 = 1/\lambda_3$.

Since $(A^{-1})^\dagger = (A^\dagger)^{-1}$ we obtain that

$$C = (A^{-1})^\dagger A^{-1} = (A^\dagger)^{-1}A^{-1} = (AA^\dagger)^{-1}$$

Therefore if σ is an eigenvalue of C than $1/\sigma$ is an eigenvalue of $AA^\dagger$. It follows that the largest eigenvalue of C is the smallest eigenvalue of $AA^\dagger$. What remains is to show that the eigenvalues of $AA^\dagger$ and $A^\dagger A$ are exactly the same.

Recall the following properties of determinants:

- $\det(AB) = \det(A)\det(B)$,
- if A^{-1} exists then $\det(A) \neq 0$,
- $\det(A - \lambda I) = 0$ if and only if λ is an eigenvalue of A.

Since

$$\begin{aligned}\det(A^\dagger A - \lambda I) &= \det(A^\dagger A - \lambda A^{-1}A) = \det(A^\dagger - \lambda A^{-1})\det(A) \\ &= \det(A)\det(A^\dagger - \lambda A^{-1}) = \det(AA^\dagger - \lambda AA^{-1}) = \det(AA^\dagger - \lambda I)\end{aligned}$$

then λ is an eigenvalue of $A^\dagger A$ if and only if it is an eigenvalue of $AA^\dagger$.

Therefore, if λ_3 is the smallest eigenvalue of $A^\dagger A$, then it is the smallest eigenvalue of $AA^\dagger$ and consequently $1/\lambda_3$ is the largest eigenvalue of C. It follows that $\|A^{-1}\|_2 = 1/\sqrt{\lambda_3}$ and therefore we obtain $\kappa_2(A) = \sqrt{\lambda_1/\lambda_3}$.