数值分析计算实习报告

**第一题**

**姓名： 田康宁**

**学号： SY1815320**

**学院： 宇航学院**

**完成日期： 2018年11月2日**

# 题目分析

由题，系数矩阵A为高阶稀疏方阵，在上/下三角区域内含有大量0元素。题目要求为计算最大、最小、按模最小特征值，特定值附近的特征值以及该方阵的条件数和行列式。

1. 由于不要求计算所有特征值，不要求计算特征向量，因此不必采用QR分解这种可求全部特征值和特征向量的计算量较大的算法。最大特征值和最小特征值可以使用幂法结合平移技巧一起求出，然后通过大小判断得出结论。即，幂法求得按模最大的特征值λ，必为特征值最值之一，再将矩阵平移为A-λ，再用幂法，可求得另一特征值最值。按模最小可使用反幂法直接求得。
2. 特定值附近的特征值可以通过结合矩阵平移的反幂法求出。
3. 由第一小题结果可以求得矩阵A的条件数。特征值可使用三角分解获得的上三角阵的对角元计算求得。

# 程序设计方案

科学计算需要的是以最少的资源在最短的时间内获得符合精度要求的结果。在程序设计上，我摒弃了追求面向对象、用户友好、普适等习惯性的程序设计思路，为了提高效率，采用面向过程且极具针对性的程序设计思路。

## 数据结构

**系数矩阵**

题目要求不允许储存系数矩阵A中含有的0元素，因此不能采用二维数组方阵的形式储存矩阵A。由于矩阵A的元素分布很有规律，在牺牲算法适用性的条件下，仅采用一维数组a[ i ] 储存A的对角元，带状区域内的其他元素均为常数，因此使用单个数b，c保存其值，将矩阵A的结构特殊性与算法设计融合起来，使构造出的算法只能对该算例使用。

**求解容器**

分别使用两个一维数组lamda1和lamda2储存Q1和Q2的结果，使用两个单独的double变量储存Q3的条件数和行列式值。程序设计中预留了获得于特征值相对应的特征向量的接口，由于题目不要求储存特征向量，仅使用一个一维数组存放随特征值一起产生的特征向量。该一维数组的内容随每次求特征值计算刷新，不做存储。

## 主要方法

**幂法**

使用幂法求取矩阵A的最大、最小特征值。按照教材，算法如下：

**S1** 任取非零向量**u**

**S2** 对**u**归一化获得y（这里可以使用2范数（欧几里得范数）或无穷范数进行归一化）

**S3** 使用 **u** = A**y** 产生新的**u**

**S4** 由向量**u**与**y**的内积获得特征值的参考值β

**S5** 当邻步迭代获得的β相对误差大于误差限ε时，转到**S2**进行新一轮迭代运算

当β的相对变化小于误差限ε时，最新的β即为求得的矩阵A按模最大的特征值，最新的向量**u**即为该特征值对应的特征向量

在编写程序时，两种范数的归一化方法都进行了尝试。从运算时间和结果精度上看，并无明显差别。可能是由于计算规模尚不足以凸显两种算法的区别。

首先对矩阵A用幂法求得按模最大的特征值λ1。由于不确定λ1是最大（λmax）/最小（λmin）特征值，将A向λ1的反向平移，将谱范围移至同号区。平移后，矩阵A-λ1的特征值全部与λ1异号，同时A的另一特征值平移至模长为|λmax| + |λmin|，符号不变。此时对矩阵A-λ1使用幂法求得特征值λ2’。将λ2’反向平移至λ2’+λ1，所得即为矩阵A的另一特征值最值λ2。将λ1和λ2比较一次，即可判断λ1和λ501的具体值。

**反幂法**

使用反幂法求取矩阵A的按模最小的特征值λs和特定值附近的特征值。教材上的参考步骤如下：

**S1** 任取非零向量**u**

**S2** 对**u**归一化获得y（使用2范数（欧几里得范数）或无穷范数进行归一化）

**S3** 使用 A**u** = **y** 产生新的**u**

**S4** 由向量**u**与**y**的内积获得特征值的参考值β

**S5** 当邻步迭代获得的1/β相对误差大于误差限ε时，转到**S2**进行新一轮迭代运算

当β的相对变化小于误差限ε时，最新的1/β即为求得的矩阵A按模最大的特征值，最新的向量**u**即为该特征值对应的特征向量

为避免**S2**求取矩阵A的逆矩阵（矩阵求逆运算相对复杂，会大大增加算法的时间复杂度和空间复杂度），这里采用线性方程组求解的数值算法。由于迭代过程中矩阵A不变，只有**y**随迭代更新。适合采用LU三角分解方法，可以避免每次迭代对A进行分解运算，节省计算时间。又由于矩阵A是带状的稀疏方阵，适合采用三角分解法解带状线性方程组的算法，可以节省矩阵中大量0元素运算的时间。

**带状三角分解法**

详细过程参见教材，这里不再赘述。

# 程序代码

该程序所有代码均为手写，具体如下：

// Tian 2018-10-31

#include<iostream>

#include<fstream>

#include<math.h>

#include<iomanip>

using namespace std;

#define N 501

double vdv(double\* v1, double\* v2, int n);

double\* vdn(double\* v1, double\* v2, double c, int n);

double\* v2v(double\* v1, double\* v2, int n);

double min(double a, double b);

double max(double a, double b);

double norm(double\* v1, double\* v2, int n);

double powermethod(double\* a, double b, double c, double\* x, int n);

double antipowermethod(double\* a, double b, double c, double\* x, double\* det, int n);

double LUbounddecompose(double\* a, double bb, double cc, double c[][N], int n, int r, int s);

double\* LUboundsolve(double c [][N], double\* B, double\* x, int n, int r, int s);

int main()

{

// set the varation

int n = N;

double X[N] = {0};

double\* x = X;

double lamda1[3] = {0};

double lamda2[39] = {0};

double cond = 0;

double det = 0;

// cout<<"hi";

double a[N] = {0};

for(int i = 0; i < N; i++)

{

int k = i+1;

a[i] = ( 1.64 - 0.024 \* (k) ) \* sin(0.2\*k) - 0.64 \* exp(0.1/k);

}

double b = 0.16;

double c = -0.064;

// solve qestion 1-1

lamda1[0] = powermethod(a, b, c, x, n);

cout<<scientific<<setprecision(12)<<lamda1[0]<<'\n';

// solve qestion 1-2

for(int i = 0; i < N; i++)

{

a[i] += -1\*lamda1[0];

}

double temp = lamda1[0];

lamda1[1] = powermethod(a, b, c, x, n);

lamda1[1] += temp;

cout<<lamda1[1]<<'\n';

// solve qestion 1-3

for(int i = 0; i < N; i++)

{

a[i] += temp;

}

lamda1[2] = antipowermethod(a, b, c, x, &det, n);

// judge lamda max and lamda min

if( lamda1[0] > lamda1[1] )

{

double t = lamda1[0];

lamda1[0] = lamda1[1];

lamda1[1] = t;

}

cout<<lamda1[2]<<'\n';

// solve question 3

cout<<"det: "<<det<<'\n';

cond = fabs( lamda1[0] / lamda1[2] );

cout<<"cond2: "<<cond<<'\n';

// solve question 2

double s = 0;

for(int k = 0; k < 39; k++)

{

double u = lamda1[0] + (k+1) \* ( lamda1[1] - lamda1[0] )/40;

double a2[N] = {0};

for(int j = 0; j < n; j++)

{

a2[j] = a[j] - u;

}

lamda2[k] = antipowermethod(a2, b, c, x, &s, n);

lamda2[k] += u;

cout<<(k+1)<<" : "<<lamda2[k]<<'\n';

}

// print the result

fstream cfout("ans.txt", ios::out);

cfout<<"\nWork 1\nQ1\n\n";

cfout<<"lamda 1 : ";

cfout<<setw(25)<<scientific<<setprecision(12)<<lamda1[0]<<'\n';

cfout<<"lamda 501 : ";

cfout<<setw(25)<<scientific<<setprecision(12)<<lamda1[1]<<'\n';

cfout<<"lamda s : ";

cfout<<setw(25)<<scientific<<setprecision(12)<<lamda1[2]<<'\n';

cfout<<"\nQ2\n";

for(int i = 0; i < 39; i++)

{

cfout<<"lamda i "<<i+1<<" : ";

cfout<<setw(25)<<scientific<<setprecision(12)<<lamda2[i]<<'\n';

}

cfout<<"\nQ3\n";

cfout<<"det(A): ";

cfout<<setw(25)<<scientific<<setprecision(12)<<det<<'\n';

cfout<<"cond2(A): ";

cfout<<setw(25)<<scientific<<setprecision(12)<<cond<<'\n';

cfout.close();

return 0;

}

// infinite norm

double norm(double\* v1, int n)

{

double v2 = v1[0];

for(int i = 1; i < n; i++)

{

if( fabs(v1[i]) > fabs (v2) )

{

v2 = v1[i];

}

}

return v2;

}

// power method 2

double powermethod(double\* a, double b, double c, double\* x, int n)

{

// initiation

double U[N] = {1};

double\* u = U;

for(int i = 0; i < n; i++)

{

u[i] = 1;

}

double Y[N] = {0};

double\* y = Y;

double beta[2] = {0};

double h0 = 0;

double h1 = 0;

double t = 0;

for(; ; )

{

h0 = norm(u, n);

y = vdn(u, y, 1/fabs(h0), n);

// calculate u

u[0] = a[0] \* y[0] + b \* y[1] + c \* y[2];

u[1] = b \* y[0] + a[1] \* y[1] + b \* y[2] + c \* y[3];

u[n-2] = b \* y[n-1] + a[n-2] \* y[n-2] + b \* y[n-3] + c \* y[n-4];

u[n-1] = a[n-1] \* y[n-1] + b \* y[n-2] + c \* y[n-3];

for(int i = 2; i < n-2; i++)

{

u[i] = c \* ( y[i-2] + y[i+2] ) + b \* ( y[i-1] + y[i+1] ) + a[i] \* y[i];

}

h1 = norm(u, n);

t = beta[1];

beta[1] = ( h0/fabs(h0) )\*h1;

beta[0] = t;

// termination judgement

if( (fabs( beta[1] - beta[0]) / fabs(beta[1]) ) <= 1e-13 )

{

break;

}

}

v2v(u, x, n);

return beta[1];

}

double antipowermethod(double\* a, double b, double c, double\* x, double\* det, int n)

{

// initiation

double U[N] = {1};

double\* u = U;

const int r = 2, s = 2;

for(int i = 0; i < n; i++)

{

u[i] = 1;

}

double eta = 0;

double Y[N] = {0};

double\* y = Y;

double beta[2] = {0};

double t = 0;

double C[5][N] = {0};

int it = 0;

\*det = LUbounddecompose(a, b, c, C, n, r, s); // sol equation groups to avoid reverse matrix

for(; ; )

{

eta = pow(vdv(u, u, n), 0.5);

y = vdn(u, y, 1/eta, n);

LUboundsolve(C, y, u, n, r, s);

it++;

// cout<<it<<'\n';

t = beta[1];

beta[1] = vdv(y, u, n);

beta[0] = t;

// cout<<beta[1]<<'\n';

// termination judgement

if( (fabs( 1/beta[1] - 1/beta[0]) / fabs(1/beta[1]) ) <= 1e-12 )

{

break;

}

}

v2v(u, x, n);

return 1/beta[1];

}

// lu decompose

double LUbounddecompose(double\* a, double bb, double cc, double c [][N], int n, int r, int s)

{

// generate c

for(int i = 0; i < 5; i++)

{

for(int j = 0; j < N; j++)

{

c[i][j] = 0;

}

}

for(int i = 0; i < 5; i++)

{

switch(i)

{

case 0:

{

for(int j = 2; j < n; j++)

{

c[i][j] = cc;

}

break;

}

case 1:

{

for(int j = 1; j < n; j++)

{

c[i][j] = bb;

}

break;

}

case 2:

{

for(int j = 0; j < n; j++)

{

c[i][j] = a[j];

}

break;

}

case 3:

{

for(int j = 0; j < n-1; j++)

{

c[i][j] = bb;

}

break;

}

case 4:

{

for(int j = 0; j < n-2; j++)

{

c[i][j] = cc;

}

break;

}

}

}

//lu decompose

for(int k = 1; k <= n; k++)

{

for(int j = k; j <= min( (k+s), n ); j++)

{

double temp = 0;

for(int t = max( max(1, k-r), max(k-r, j-s) ) ; t <= k-1; t++)

{

temp += c[k-t+s][t-1] \* c[t-j+s][j-1];

}

c[k-j+s][j-1] = c[k-j+s][j-1] - temp;

}

if( k < n)

{

for(int i = (k+1); i <= min((k+r), n); i++ )

{

double temp = 0;

for(int t = max( max(1, (i-r)), max((i-r), (k-s) ) ); t <= k-1; t++ )

{

temp += c[i-t+s][t-1] \* c[t-k+s][k-1];

}

c[i-k+s][k-1] = (c[i-k+s][k-1] - temp) / c[s][k-1];

}

}

}

// cal the det

double det = 1;

for(int i = 0; i < n; i++)

{

det \*= c[s][i];

}

return det;

}

// lu bond decompose

double\* LUboundsolve(double c [][N], double\* B, double\* x, int n, int r, int s)

{

// solve lu

double b[N] = {0};

for(int i = 0; i < n; i++)

{

b[i] = B[i];

}

for(int i = 2; i <= n; i++)

{

double temp = 0;

for(int t = max(1, i-r); t <= i-1; t++ )

{

temp += c[i-t+s][t-1] \* b[t-1];

}

b[i-1] = b[i-1] - temp;

}

x[n-1] = b[n-1] / c[s][n-1];

for(int i = n-1; i >= 1; i--)

{

double temp = 0;

for(int t = i+1; t <= min((i+s), n-1); t++ )

{

temp += c[i-t+s][t-1] \* x[t-1];

}

x[i-1] = ( b[i-1] - temp ) / c[s][i-1];

}

return x;

}

// vector dot vector

double vdv(double\* v1, double\* v2, int n)

{

double sum = 0;

for(int i = 0; i < n; i++)

{

sum += v1[i] \* v2[i];

}

return sum;

}

// vector dot number

double\* vdn(double\* v1, double\* v2, double c, int n)

{

for(int i = 0; i < n; i++)

{

v2[i] = v1[i] \* c;

}

return v2;

}

// vector copy

double\* v2v(double\* v1, double\* v2, int n)

{

for(int i = 0; i < n; i++)

{

v2[i] = v1[i];

}

return v2;

}

double min(double a, double b)

{

if(a > b)

return b;

else

return a;

}

double max(double a, double b)

{

if(a > b)

return a;

else

return b;

}

# 结果说明

计算结果（以下内容完全来自程序输出的结果文件ans.txt，读者可自行编译代码，运行后与输出文件对照）

Work 1

Q1

lamda 1 : -1.070011361512e+01

lamda 501 : 9.724634099853e+00

lamda s : -5.557910794229e-03

Q2

lamda i 1 : -1.018293403315e+01

lamda i 2 : -9.585707425068e+00

lamda i 3 : -9.172672423928e+00

lamda i 4 : -8.652284007898e+00

lamda i 5 : -8.093483808675e+00

lamda i 6 : -7.659405407692e+00

lamda i 7 : -7.119684648691e+00

lamda i 8 : -6.611764339397e+00

lamda i 9 : -6.066103226595e+00

lamda i 10 : -5.585101052628e+00

lamda i 11 : -5.114083529812e+00

lamda i 12 : -4.578872176865e+00

lamda i 13 : -4.096470926260e+00

lamda i 14 : -3.554211215751e+00

lamda i 15 : -3.041090018133e+00

lamda i 16 : -2.533970311130e+00

lamda i 17 : -2.003230769563e+00

lamda i 18 : -1.503557611227e+00

lamda i 19 : -9.935586060075e-01

lamda i 20 : -4.870426738850e-01

lamda i 21 : 2.231736249575e-02

lamda i 22 : 5.324174742069e-01

lamda i 23 : 1.052898962693e+00

lamda i 24 : 1.589445881881e+00

lamda i 25 : 2.060330460274e+00

lamda i 26 : 2.558075597073e+00

lamda i 27 : 3.080240509307e+00

lamda i 28 : 3.613620867692e+00

lamda i 29 : 4.091378510451e+00

lamda i 30 : 4.589143359739e+00

lamda i 31 : 5.132924283898e+00

lamda i 32 : 5.594906348083e+00

lamda i 33 : 6.080933857027e+00

lamda i 34 : 6.680354092112e+00

lamda i 35 : 7.293877448127e+00

lamda i 36 : 7.717111714236e+00

lamda i 37 : 8.225220014050e+00

lamda i 38 : 8.648666032539e+00

lamda i 39 : 9.254200327472e+00

Q3

det(A): 2.772786141752e+118

cond2(A): 1.925204273921e+03

结果与教材后给出的参考答案进行对比，大部结果前8位有效数字一致，说明计算结果是正确的

# 讨论分析

## 初始向量的选取

调试程序过程中，我发现幂法和反幂法并非对初始迭代向量没有条件。教材上写的是“任取非零实向量**u0** ”，实际上对u0的选取应当是有限制的。起初我在程序中设定的初始向量为单位向量e1，即 ( 1, 0, 0, … , 0 )。该初始条件下，幂法和反幂法求得的解都不是正确答案。当我将初始向量置为 ( 1, 1, 1, … , 1 ) 时，得到的结果就是正确的。结果正确说明算法应该是正确的，那么先前得到错误的收敛结果，应当是由于算法本身或其代码实现对初始向量的选取有一定条件。具体原因仍有待探究。

## 针对问题类型选取合适的算法可以节省时间

该次大作业让我切实认识到，解决问题的方法应当依据具体的问题来设计。之前我一直觉得高斯消元法原理简单，代码实现也相对容易，比LU三角分解更容易使用。所以在该实验中，我首先采用了高斯消元法求解反幂法中的线性方程组。大概也有代码错误的原因，计算过程比较慢，迭代次数很大，大量占用计算资源。这时我才开始仔细分析，高斯算法虽然简单，但没有灵活性，计算次数难以减少。而该程序中，只有方程组右端的向量发生改变，系数矩阵是不变的，使用LU解法，可以“分解一次，求解n次”。相比之下，高斯算法相当于“分解n次，求解n次”，所以计算效率不如LU算法。这让我想到，数值分析的各种方法没有绝对的优劣好坏，对于不同的问题，有不同的适用方法。这大概也是如此纷繁复杂的数值方法得以存在的原因吧。

## 使用反幂法结合平移法求出矩阵所有特征值

在求解第一题时，我想到，是否可以使用反幂法求出矩阵的所有特征值。对于N阶矩阵，假设有N个特征值，在求出λmin和λmax的情况下，将两特征值最值之间平均插入N-2个点，使用带平移的反幂法求出N-2个点附近的N-2个特征值。如果有重根，则N个解出的特征值会出现相同值。我对第二小题的求解代码进行了简单修改，结果和我预想的一样，而且运算速度较快，说明这种方法是可行的。

## 避免粗心

编码/调试的过程就是一个不断犯错、不断改错的过程。平移矩阵求解特征值要记得将特征值平移回来，反幂法求得的β要取倒数才是求得的特征值，看清三角分解中行指标和列指标的含义，……在这个过程中，粗心大意会导致为一个微小的细节浪费好几个小时。