**数值分析第一次大作业**

BY1702103 斯志仲

具体思路：（我用的是第三版书，算法以这个版本为主）

1. 幂法直接采用的是书上P49页算法。
2. 反幂法直接采用的是带位移量的算法。在书上P52页。求绝对值最小的特征值时，直接令位移量为0.
3. Lambda1和lambda501，直接通过判断幂法算出的Lambda\_max的正负，如果为负数，则是Lambda1，否则则是Lambda501，另一个则利用带位移的反幂法求出。
4. 反幂法需要求解方程组，我采用的是普通高斯消去法。
5. 由于算法要求不能保持A的非零元素。我是采用两种方式保存A矩阵的

第一种高斯消去的时候

c b a1 b c

c b a2 b c

…………

c b a501 b c

第二种其他运算。 a1 a2 a3……a501 为一个数据，还有b c 为两个double 量。

编程里面的一些特点

1. 由于C语言返回指针数组的时候。容易导致内存管理混乱，得出错误结论。我所有需要返回数组的函数，均是采用在输入参数的最后一个加入一个数值指针，通过这个参数，返回结果数值，这样同时可以返回多个参数。
2. 防止在底层函数里面修改输入矩阵的数值，所有底层函数的矩阵运算，都对输入的矩阵做了个深拷贝，然后利用这个矩阵进行运算。
3. 在具体算法实现前，都写了很多矩阵和向量的基本运算函数，直接调用。虽然这个代码很长，但是非常易读。

#include <stdio.h>

#include <stdlib.h>

#include<math.h>

#define SIZE 501

//注明：由于题目中给出的sin(0.2i)中，没有注明是弧度还是角度，本算法全部按照弧度计算。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

幂法部分

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 向量除以相邻的模长，对应书中的第一个和第二个公式,将向量归一化

//v 被归一化的向量 nomalized\_v 归一化之后的结果

void vector\_normalize(double v[SIZE],double nomalized\_v[SIZE])

{

double length=0;

int i=0;

for(i=0;i<SIZE;i++)

{

length=length+v[i]\*v[i];

}

for(i=0;i<SIZE;i++)

nomalized\_v[i]=v[i]/sqrt(length);

}

//A矩阵乘以向量v,由于要求A的非0数据都不保存，故单独定义这种乘法，其中非0元素均不保存

//v A\*v中的v， result 计算后的结果

void A\_plus\_vector(double v[SIZE],double result[SIZE])

{

double a[SIZE];

int i=0;

for(i=0;i<SIZE;i++)

{

//a 的公式中，i从1开始，c语言是从0开始，故而公式上加了1. 另外sin(x),x是弧度

a[i]=(1.64-0.024\*(i+1))\*sin(0.2\*(i+1))-0.64\*exp(0.1/(i+1));

}

double b=0.16;

double c=-0.064;

//A的第一行乘以v

result[0]=a[0]\*v[0]+b\*v[1]+c\*v[2];

//A的第二行乘以v

result[1]=b\*v[0]+a[1]\*v[1]+b\*v[2]+c\*v[3];

//A的第三行到A的第499行乘v

for(i=2;i<499;i++)

{

result[i]=v[i-2]\*c+v[i-1]\*b+v[i]\*a[i]+v[i+1]\*b+v[i+2]\*c;

}

//A的第500行乘v

result[499]=c\*v[497]+b\*v[498]+a[499]\*v[499]+b\*v[500];

//A的第501行乘以v

result[500]=c\*v[498]+b\*v[499]+a[500]\*v[500];

}

//两个向量相乘，对应公式3

double dot\_multiply(double v1[SIZE],double v2[SIZE])

{

double result=0;

int i=0;

for(i=0;i<SIZE;i++)

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

return result;

}

//幂法

double power\_method()

{

double u0[SIZE];

double u1[SIZE];

double y0[SIZE];

double y1[SIZE];

int i=0;

//初始化u0

for(i=0;i<SIZE;i++)

u0[i]=0.1;

//计算y0和u1

vector\_normalize(u0,y0);

A\_plus\_vector(y0,u1);

double beta1=dot\_multiply(y0,u1);

//计算y1和u2

vector\_normalize(u1,y1);

A\_plus\_vector(y1,u0);

double beta2=dot\_multiply(y1,u0);

//循环计算，直到达到误差精度。

while((fabs(beta2-beta1)/fabs(beta2))>pow(10,-12))

{

vector\_normalize(u0,y0);

A\_plus\_vector(y0,u1);

beta1=dot\_multiply(y0,u1);

vector\_normalize(u1,y1);

A\_plus\_vector(y1,u0);

beta2=dot\_multiply(y1,u0);

};

return beta2;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

带原点位移反幂法部分

由于要求A\*u=y,故使用Gauss消去法。同时可以求Det(A)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//对A进行p的位移，即对(A-pI)u=y\*x进行高斯消去，返回u和Det(A)。

//p 位移量,y是公式中的y， result为计算后的结果，返回值为Det(A)

double Gauss\_Elimination(double p,double y[SIZE],double result[SIZE])

{

double a[501];

int i=0;

for(i=0;i<501;i++)

{

//a 的公式中，i从1开始，c语言是从0开始，故而公式上加了1. 另外sin(x),x是弧度

a[i]=(1.64-0.024\*(i+1))\*sin(0.2\*(i+1))-0.64\*exp(0.1/(i+1))-p;//p是位移量

}

double b=0.16;

double c=-0.064;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

由于矩阵的特殊性，每次只需要对下面三行做Gauss消去

为了不存储0结果。定义一个501\*5的矩阵。五列为

c b ai b c或者他们消去后的结果。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int j=0;

double temp\_matrix[SIZE][5];

//对temp\_matrix进行赋值

for(i=0;i<SIZE;i++)

{

temp\_matrix[i][0]=c;

temp\_matrix[i][1]=b;

temp\_matrix[i][2]=a[i];

temp\_matrix[i][3]=b;

temp\_matrix[i][4]=c;

result[i]=y[i];

}

double m=0;

//消去过程，只需要消去i下面两行

for(i=0;i<499;i++)

{

//消去i下面第一行

m=temp\_matrix[i+1][1]/temp\_matrix[i][2];

for(j=1;j<4;j++)

{

temp\_matrix[i+1][j]=temp\_matrix[i+1][j]-m\*temp\_matrix[i][j+1];

}

result[i+1]=result[i+1]-m\*result[i];

//消去i下面第二行

m=temp\_matrix[i+2][0]/temp\_matrix[i][2];

for(j=0;j<3;j++)

{

temp\_matrix[i+2][j]=temp\_matrix[i+2][j]-m\*temp\_matrix[i][j+2];

}

result[i+2]=result[i+2]-m\*result[i];

}

//消去最后一行

m=temp\_matrix[500][1]/temp\_matrix[499][2];

for(j=1;j<3;j++)

{

temp\_matrix[500][j]=temp\_matrix[500][j]-m\*temp\_matrix[499][j+1];

}

result[500]=result[500]-m\*result[499];

//回代过程，同样只需要回代前面两个

for(i=500;i>1;i--)

{

m=temp\_matrix[i-1][3]/temp\_matrix[i][2];

temp\_matrix[i-1][3]=temp\_matrix[i-1][3]-m\*temp\_matrix[i][2];

result[i-1]=result[i-1]-m\*result[i];

m=temp\_matrix[i-2][4]/temp\_matrix[i][2];

temp\_matrix[i-2][4]=temp\_matrix[i-2][4]-m\*temp\_matrix[i][2];

result[i-2]=result[i-2]-m\*result[i];

}

m=temp\_matrix[0][3]/temp\_matrix[1][2];

temp\_matrix[0][3]=temp\_matrix[0][3]-m\*temp\_matrix[1][2];

result[0]=result[0]-m\*result[1];

double Det=1;

for(i=0;i<SIZE;i++)

{

Det=Det\*temp\_matrix[i][2];

result[i]=result[i]/temp\_matrix[i][2];

}

return Det;

}

//带位移的反幂法

double anti\_power\_method(double p)

{

double u0[SIZE];

double u1[SIZE];

double y0[SIZE];

double y1[SIZE];

int i=0;

//初始化u0

for(i=0;i<SIZE;i++)

u0[i]=1;

//计算y0和u1

vector\_normalize(u0,y0);

Gauss\_Elimination(p,y0,u1);

double beta1=dot\_multiply(y0,u1);

//计算y1和u2

vector\_normalize(u1,y1);

Gauss\_Elimination(p,y1,u0);

double beta2=dot\_multiply(y1,u0);

//循环计算，直到达到误差精度。

while((fabs(beta2-beta1)/fabs(beta2))>pow(10,-12))

{

vector\_normalize(u0,y0);

Gauss\_Elimination(p,y0,u1);

beta1=dot\_multiply(y0,u1);

vector\_normalize(u1,y1);

Gauss\_Elimination(p,y1,u0);

beta2=dot\_multiply(y1,u0);

};

return 1/beta2+p;

}

int main()

{

double lambda\_max=power\_method();

double lambda1,lambda501;

//如果lambda>0,则lambda501=lambda,且为按模最大。则lambda1为最接近-lambda501的

printf("\n\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nanswer of question 1 is:\n");

if(lambda\_max>0)

{

lambda501=lambda\_max;

printf("lambda501=%.14e\n",lambda501);

lambda1=anti\_power\_method(-lambda501);

printf("lambda1=%.14e\n",lambda1);

}

//如果lambda<0,则lambda1=lambda，且为按摩最大。则lambda501为最接近-lambda1的

if(lambda\_max<0)

{

lambda1=lambda\_max;

printf("lambda1=%.14e\n",lambda1);

lambda501=anti\_power\_method(-lambda1);

printf("lambda501=%.14e\n",lambda501);

}

double lambdas=anti\_power\_method(0);

printf("lambdas=%.14e\n",lambdas);

//求于uk=lambda1+k(lambda501-lambda1)/40 最接近的lambda

printf("\n\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nanswer of quetion 2 is:\n");

int k=0;

double uk=0;

for(k=1;k<40;k++)

{

uk=lambda1+k\*(lambda501-lambda1)/40;

double lambda=anti\_power\_method(uk);

printf("lambda\_i%d=%.14e\n",k,lambda);

}

printf("\n\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nanswer of quetion 3 is:\n");

//求条件数：对于非奇异实对称矩阵，cond为按模最大特征值/按模最小特征值的绝对值

printf("cond(A)2=%.14e\n",fabs(lambda\_max/lambdas));

//求Det ，直接调用Gauss消去法

double y[SIZE];

double result[SIZE];

double Det=Gauss\_Elimination(0,y,result);

printf("Det(A)=%.14e\n",Det);

return 1;

}

结果：

