

数值分析第一次大作业实验报告

16091038 叶宇轩

题目描述

给定一个501阶矩阵 A ,

$$\begin{bmatrix} a_1 & b & c & & \\ b & \ddots & \ddots & \ddots & \\ c & \ddots & \ddots & \ddots & c \\ \ddots & \ddots & \ddots & & b \\ c & b & a_{501} & & \end{bmatrix}$$

其中, $a_i = (1.64 - 0.024i) \sin(0.2i) - 0.64e^{\frac{0.1}{i}}$ ($i = 1, 2, \dots, 501$), $b = 0.16$, $c = -0.064$ 。记矩阵 A 的特征值为 λ_i ($i = 1, 2, \dots, 501$), 并且有

$$\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_{501}, |\lambda_s| = \min\{|\lambda_i| : 1 \leq i \leq 501\}$$

1. 求 $\lambda_1, \lambda_{501}, \lambda_s$

2. 求 A 与 $\mu_k = \lambda_1 + k \frac{\lambda_{501} - \lambda_1}{40}$ 最接近的特征值

3. 求 A 的谱范数条件数 $\text{cond}(A)_2$ 和行列式 $\det A$

要求

1. 算法涉及的精度取 $\epsilon = 10^{-12}$

2. A 的零元素不储存

3. 显示至少12位有效数字

算法设计

0. 带状矩阵的储存方法

由于 A 为 $r = s = 2$ 的带状矩阵，因此我们可以设置 $m (= r + s + 1)$ 行 n 列的矩阵矩阵 C ，令

$$c_{i-j+s+1,j} = a_{ij}$$

即可将 A 中不为0的元素储存下来。

1. 需要使用的算法

1.1 幂法

幂法主要用于计算矩阵按模最大的特征向量。

设 $n \times n$ 实矩阵 A 有 n 个线性无关的特征向量 x_1, x_2, \dots, x_n ， x_i 对应的特征值 λ_i 满足

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|, Ax_i = \lambda_i x_i \quad (i = 1, 2, \dots, n)$$

任取 n 维非零向量 u_0 ，有

$$u_0 = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n$$

从 u_0 出发，令 $u_k = Au_{k-1}$ ($k = 1, 2, \dots$)，可以推出

$$\begin{aligned} u_k &= Au_{k-1} = \cdots = A^k u_0 = \\ \alpha_1 A^k x_1 + \alpha_2 A^k x_2 + \cdots + \alpha_n A^k x_n &= \\ \alpha_1 \lambda_1^k x_1 + \alpha_2 \lambda_2^k x_2 + \cdots + \alpha_n \lambda_n^k x_n &= \\ \lambda_1^k [\alpha_1 x_1 + \alpha_2 (\frac{\lambda_2}{\lambda_1})^k x_2 + \cdots + \alpha_n (\frac{\lambda_n}{\lambda_1})^k x_n] & \end{aligned}$$

设 $\alpha_1 \neq 0$ ，则 k 充分大时，有

$$u_k \approx \lambda_1^k \alpha_1 x_1$$

实际计算中为了避免 u_k 的模过大，每次迭代后对 u_k 单位化，因此迭代公式为

$$\begin{cases} y_{k-1} = \frac{u_{k-1}}{\|u_{k-1}\|} & (k=1, 2, \dots) \\ u_k = Ay_{k-1} \end{cases}$$

当使用二范数 $\|\cdot\|_2$ 时，令 $\beta_k = y_{k-1}^T u_k$ ，可以得到

$$\lim_{k \rightarrow \infty} \beta_k = \lambda_1$$

当使用无穷范数 $\|\cdot\|_\infty$ 时，令 $\beta_k = \frac{e_r^T u_k}{e_r^T y_{k-1}}$ ，设 u_{k-1} 模最大的分量为第 r 个分量， e_r 是 n 维基本单位向量，则有

$$\lim_{k \rightarrow \infty} \beta_k = \frac{e_r^T Ax_1}{e_r^T x_1} = \lambda_1$$

当 $\frac{|\beta_k - \beta_{k-1}|}{|\beta_k|} \leq \epsilon$ 时，认为当前 β_k 与 λ_1 足够接近，算法停止。

1.2 反幂法

反幂法主要用于计算矩阵按模最大的特征向量。

设 $n \times n$ 实矩阵 A 有 n 个线性无关的特征向量 x_1, x_2, \dots, x_n ， x_i 对应的特征值 λ_i 满足

$$|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_{n-1}| > |\lambda_n|, Ax_i = \lambda_i x_i \quad (i = 1, 2, \dots, n)$$

由于 A 非奇异，故 $\lambda_i \neq 0$ ($i = 1, 2, \dots, n$)。由 $Ax_i = \lambda_i x_i$ 得

$$A^{-1}x_i = \frac{1}{\lambda_i}x_i$$

此时 $\frac{1}{\lambda_n}$ 是 A^{-1} 的按模最大的特征值， x_n 是对应的特征向量，对 A^{-1} 使用幂法即可求出 $\frac{1}{\lambda_n}$ ，为了减少计算 A^{-1} 的计算量，将迭代公式变形为

$$\begin{cases} y_{k-1} = \frac{u_{k-1}}{\|u_{k-1}\|} & (k=1, 2, \dots) \\ Au_k = y_{k-1} \end{cases}$$

每次迭代需要对线性方程组 $Au_k = y_{k-1}$ 进行求解，为了节省计算量，我们求解方程组时不使用高斯消去法，而是选择使用下面的Doolittle分解法。

1.3 Doolittle分解法

*Doolittle*分解法主要用于求解线性方程组 $Ax = b$ 的解。通过将方程系数矩阵 A 分解为 $A = LU$, 其中 L 是下三角矩阵, U 是上三角矩阵。此时方程组可化简成为两个容易求解的三角形方程组

$$Ly = b, Ux = y$$

先由 $Ly = b$ 解出向量 y , 再由 $Ux = y$ 解出向量 x 即为原方程组解向量。

Doolittle 分解计算公式:

$$\begin{cases} \text{for } i := 1 \rightarrow n \\ u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj} \quad (j = i, i+1, \dots, n) \\ l_{ji} = (a_{ji} - \sum_{k=1}^{i-1} l_{jk}u_{ki})/u_{ii} \quad (j = i+1, i+2, \dots, n) \end{cases}$$

解三角方程组 $Ly = b, Ux = y$ 的计算公式:

$$\begin{cases} y_1 = b_1 \\ y_i = b_i - \sum_{j=1}^{i-1} l_{ij}y_j \quad (i = 2, 3, \dots, n) \\ x_n = y_n/u_{nn} \\ x_i = (y_i - \sum_{j=i+1}^n u_{ij}x_j)/u_{ii} \quad (i = n-1, n-2, \dots, 1) \end{cases}$$

1.4 带原点位移的幂法和反幂法

若 λ 是 A 的特征值, 则 $\lambda - p$ 是矩阵 $A - pI$ 的特征值, 其中 I 是单位矩阵; 反之也成立, 即

$$Ax = \lambda x, (A - pI)x = (\lambda - p)x$$

可互为因果。其中 p 称为原点位移。

2. 总体算法

2.1 求 $\lambda_1, \lambda_{501}, \lambda_s$

1. 先使用反幂法求出按模最小的特征值 λ_s
2. 使用幂法求出 A 的按模最大的特征值, 记为 λ
3. 对于 λ , 有

$$\lambda_1 - \lambda \leq \lambda_2 - \lambda \leq \dots \leq \lambda_{501} - \lambda \leq 0$$

此时按模最大的特征值为 $\lambda_1 - \lambda$, 使用位移为 λ 的幂法求出这个值并求出 λ_1

4. 若 $\lambda \neq \lambda_1$, 则 $\lambda_{501} = \lambda$; 若 $\lambda = \lambda_1$, 有

$$0 \leq \lambda_1 + \lambda \leq \lambda_2 + \lambda \leq \dots \leq \lambda_{501} + \lambda$$

此时按模最大的特征值为 $\lambda_{501} + \lambda$, 使用位移为 $-\lambda$ 的幂法求出这个值并求出 λ_{501}

2.2 求与 $\mu_k = \lambda_1 + k \frac{\lambda_{501} - \lambda_1}{40}$ 最接近的特征值

利用带原点平移的反幂法, 设位移为 μ_k , 则与其最接近的特征值 λ_{i_k} 满足

$$|\lambda_{i_k}| = \min_{1 \leq i \leq 501} |\lambda_i - \mu_k|$$

使用反幂法求出 $\lambda_{i_k} - \mu_k$, 并计算得到 λ_{i_k}

2.3 求 A 的谱范数条件数 $cond(A)_2$ 和行列式 $det A$

2.3.1 $cond(A)_2$

对于非奇异的实对称矩阵 A , 有

$$cond(A)_2 = \left| \frac{\lambda_1}{\lambda_n} \right|$$

其中 λ_1 和 λ_n 分别是 A 的按模最大最小的特征值

2.3.2 $det A$

由 Doolittle 分解法

$$A = LU \implies det A = det L \ det U$$

而 L 的对角线上都为 1, 所以有

$$det A = \prod_{i=1}^{501} u_{ii}$$

Source Code

```
#include<cmath>
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<iomanip>
#include<iostream>
#include<algorithm>
using namespace std;
#define NORM_2
//#define NORM_INF
/*如果define NORM_2则使用2范数，如果define NORM_INF则使用无穷范数*/
const int N=501,N_MAX=501+10,inf=~0u>>1,r=2,s=2;
const double eps=1e-15,b=0.16,c=-0.064;
double lambda[N_MAX],C[r+s+1+10][N_MAX],Doo[r+s+1+10][N_MAX],lambda1,lambda501,lambda_s,de
inline int sgn(double x){
    if (x>eps) return 1;
    if (x<-eps) return -1;
    return 0;
}
inline double sqr(double x){ return x*x; }
inline double Matrix(int i,int j,double p=0){
    if (i>j+2 || i<j-2) return 0;
    if (i==j) return C[i-j+s+1][j]-p;
    return C[i-j+s+1][j];
}//计算矩阵元素的值
void Init(){
    memset(C,0,sizeof(C));
    for (int i=1; i<=N; ++i)
        C[s+1][i]=(1.64-0.024*i)*sin(0.2*i)-0.64*exp(0.1/i);
    for (int i=2; i<=N; ++i)
        C[s+2][i-1]=b;
    for (int i=2; i<=N; ++i)
        C[s+3][i-2]=c;
    for (int i=1; i<=N-1; ++i)
        C[s][i+1]=b;
    for (int i=1; i<=N-1; ++i)
        C[s-1][i+2]=c;
}//将501阶的带状矩阵转化为r+s+1行501列的矩阵

#ifdef NORM_2
double PowerMethod(double p=0){
    double u[N_MAX],y[N_MAX],eta,beta=0;
```

```

for (int i=1; i<=N; ++i)
    u[i]=1;
for ( ; ; ){
    eta=0;
    for (int i=1; i<=N; ++i)
        eta+=sqrt(u[i]);
    eta=sqrt(eta);
    for (int i=1; i<=N; ++i)
        y[i]=u[i]/eta;
    for (int i=1; i<=N; ++i){
        u[i]=0;
        for (int j=1; j<=N; ++j){
            if (j>i+2) break;
            u[i]+=Matrix(i,j,p)*y[j];
        }
    }
    double cur=0;
    for (int i=1; i<=N; ++i)
        cur+=y[i]*u[i];
    if (fabs(cur-beta)/fabs(cur)<=eps) return cur;
    beta=cur;
}
}//位移为p, 使用2范数的幂法 (默认位移为0)
#endif
#ifdef NORM_INF
double PowerMethod(double p=0){
    double u[N_MAX],y[N_MAX],beta=0;
    for (int i=1; i<=N; ++i)
        u[i]=1;
    for ( ; ; ){
        double hr=0;
        for (int i=1; i<=N; ++i)
            if (fabs(u[i])-fabs(hr)>eps)
                hr=u[i];
        for (int i=1; i<=N; ++i)
            y[i]=u[i]/fabs(hr);
        for (int i=1; i<=N; ++i){
            u[i]=0;
            for (int j=1; j<=N; ++j){
                if (j>i+2)) break;
                u[i]+=Matrix(i,j,p)*y[j];
            }
        }
        double cur=0;
        for (int i=1; i<=N; ++i)
            if (fabs(u[i])-fabs(cur)>eps)
                cur=u[i];
    }
}

```

```

    cur=cur*sgn(hr);
    if (fabs(cur-beta)/fabs(cur)<=eps) return cur;
    beta=cur;
}
}//位移为p, 使用无穷范数的幂法 (默认位移为0)
#endif
inline int convert(int i,int j){
    return ((i-j+s+1>0)&&(i-j+s+1<6))?i-j+s+1:0;
}
//坐标转换
void DoolittleInit(double p=0){
    memcpy(Doo,C,sizeof(C));
    for (int i=1; i<=N; ++i)
        Doo[convert(i,i)][i]-=p;
    for (int i=1; i<=N; ++i){
        for (int j=i; j<=min(N,i+2); ++j){
            double tmp=Doo[convert(i,j)][j];
            for (int k=1; k<=i-1; ++k)
                tmp-=Doo[convert(i,k)][k]*Doo[convert(k,j)][j];
            Doo[convert(i,j)][j]=tmp;
        }
        for (int j=i+1; j<=N; ++j){
            double tmp=Doo[convert(j,i)][i];
            for (int k=max(1,i-2); k<=i-1; ++k)
                tmp-=Doo[convert(j,k)][k]*Doo[convert(k,i)][i];
            Doo[convert(j,i)][i]=tmp/Doo[convert(i,i)][i];
        }
    }
}
//Doolittle分解
void Doolittle(double u[],double y[],double p=0){
    double Y[N_MAX];
    memcpy(Y,y,sizeof(Y));
    for (int i=2; i<=N; ++i){
        double tmp=y[i];
        for (int j=max(1,i-2); j<=i-1; ++j)
            tmp-=Doo[convert(i,j)][j]*Y[j];
        Y[i]=tmp;
    }
    u[N]=Y[N]/Doo[convert(N,N)][N];
    for (int i=N-1; i>=1; --i){
        double tmp=Y[i];
        for (int j=i+1; j<=min(N,i+2); ++j)
            tmp-=(Doo[convert(i,j)][j])*u[j];
        u[i]=tmp/Doo[convert(i,i)][i];
    }
}

double InvPowerMethod(double p=0){

```

```

double u[N_MAX],y[N_MAX],beta=inf;
for (int i=1; i<=N; ++i)
    u[i]=1;
for (; ; ){
    double eta=0;
    for (int i=1; i<=N; ++i)
        eta+=sqr(u[i]);
    eta=sqrt(eta);
    for (int i=1; i<=N; ++i)
        y[i]=u[i]/eta;
    Doolittle(u,y,p);
    double cur=0;
    for (int i=1; i<=N; ++i)
        cur+=y[i]*u[i];
    if (fabs(1/cur-1/beta)/fabs(1/cur)<=eps) return 1/cur;
    beta=cur;
}
}//位移为p的反幂法 (默认位移为0)
double Det(){
    double ret=1;
    for (int i=1; i<=N; ++i)
        ret*=Doo[convert(i,i)][i];
    return ret;
}//计算行列式
int main(){
    Init();
    DoolittleInit();
    det=Det();
    lambda1=PowerMethod();
    lambda_s=InvPowerMethod();
    lambda501=PowerMethod(lambda1)+lambda1;
    cond=fabs(lambda1/lambda_s);
    if (lambda1>lambda501) swap(lambda1,lambda501);
    //用%.12E控制输出格式为科学计数法, 且保留小数点后12位 (此时有效数字至少为13位)
    printf("lambda_1=%.12E lambda_501=%.12E lambda_s=%.12E\n"
    , lambda1, lambda501, lambda_s);
    for (int i=1; i<=39; ++i){
        double miu=lambda1+i*(lambda501-lambda1)/40;
        DoolittleInit(miu);
        printf("lambda_i%d=%.12E ", i, InvPowerMethod(miu)+miu);
    }
    printf("\ncond(A)=%.12E detA=%.12E\n", cond, det);

    return 0;
}

```

结果

```
1 lambda_1=-1.070011361502E+01 lambda_501=9.724634098777E+00 lambda_s=-5.557910794230E-03
2 lambda_i1=-1.018293403315E+01 lambda_i2=-9.585707425068E+00 lambda_i3=-9.172672423928E+00 lambda_i4=-8.652284007898E+00
lambda_i5=-8.093483808675E+00 lambda_i6=-7.659405407692E+00 lambda_i7=-7.119684648691E+00 lambda_i8=-6.611764339397E+00
lambda_i9=-6.066103226595E+00 lambda_i10=-5.585101052628E+00 lambda_i11=-5.114083529812E+00 lambda_i12=-4.578872176865E+00
lambda_i13=-4.096470926260E+00 lambda_i14=-3.554211215751E+00 lambda_i15=-3.041090018133E+00 lambda_i16=-2.533970311130E+00
lambda_i17=-2.003230769563E+00 lambda_i18=-1.503557611227E+00 lambda_i19=-9.935586060075E-01 lambda_i20=-4.870426738850E-01
lambda_i21=2.231736249575E-02 lambda_i22=5.324174742069E-01 lambda_i23=1.052898962693E+00 lambda_i24=1.589445881881E+00
lambda_i25=2.060330460274E+00 lambda_i26=2.558075597073E+00 lambda_i27=3.080240509307E+00 lambda_i28=3.613620867692E+00
lambda_i29=4.091378510451E+00 lambda_i30=4.603035378279E+00 lambda_i31=5.132924283898E+00 lambda_i32=5.594966348083E+00
lambda_i33=6.080933857027E+00 lambda_i34=6.680354092112E+00 lambda_i35=7.293877448127E+00 lambda_i36=7.717111714236E+00
lambda_i37=8.225220014050E+00 lambda_i38=8.648666065193E+00 lambda_i39=9.254200344575E+00
3 cond(A)=1.925204273902E+03 deta=2.772786141752E+118
```

算法分析

空间复杂度分析

根据本题带状矩阵的特点，我们可以使用特殊的方式储存矩阵，从而减少所占用的空间。储存矩阵所需的空间从 $O(n^2)$ 级别，降到了 $O(n)$ 级别。幂法、反幂法、Doolittle分解法等算法中需要用到的辅助数组所占空间也是 $O(n)$ 级别的，因此我们可以认为算法整体的空间复杂度是 $O(n)$

幂法、反幂法

- **优点：**思路清晰，编程复杂度低，查错方便
- **缺点：**算法的收敛性受初始向量 u_0 的影响，对某些特殊的初始迭代向量，若只使用精度控制结果，则算法无法计算出正确结果，此时需要使用精度和迭代次数同时控制结果；同时，存在复特征值的情况难以使用幂法及反幂法解决

Doolittle分解法

根据本题需要多次计算系数矩阵相同，常数向量不同的线性方程组的特点，因此我们选用Doolittle分解法时，系数矩阵 A 的三角分解形式不变，这样能够节省很大一部分的计算量，从而提高程序运行效率；而高斯消去法每次需要重新进行计算，冗余计算过多，因此在此处不使用这个算法

剪枝

考慮到 A 是 $r = s = 2$ 的帶狀矩陣，當 $i > j + 2$ 或 $i < j - 2$ 時有 $a_{ij} = 0$ ，此時跳出循環可節省一部分循環所需的时间。經過測試，加上剪枝後算法運行時間從 $14.335s$ 降至 $0.0806s$

編譯優化

1. 部分類似於 \min, \max, \abs 一類的小函數在程序中被多次用到，在函數前增加關鍵詞 $inline$ 可以部分提高程序運行效率
2. 編譯時打開編譯開關 $-O2$ ，可以提高 $double$ 類型的計算效率

經過測試，使用編譯優化後算法運行時間可以降至 $0.464s$

測試平台

CPU:i7-6700HQ 2.60GHz×8 RAM:7.7GB OS:Ubuntu 17.10