北京航空航天大學
BEIHANG UNIVERSITY

数值分析大作业

计算实习题（1）

院（系）名称　宇航学院

学　　号　ZY2115107

学生姓名　段诗阳

2021年10月26日

题目要求：

1. 求出矩阵A所有特征值中最小的特征值λ_1和最大的特征值λ_{501}
2. 求出所有特征值中模最小的特征值λ_s
3. 求出靠近$\{\mu_k\}$的一组数$\{\lambda_{i_k}\}$
4. 求出矩阵A的条件数$cond(A)_2$
5. 求解$\det A$

解题思路：

1. 利用幂法求出矩阵A的模最大的特征值λ_a，其一定是λ_1或λ_{501}之一。然后再利用带原点平移的幂法，将矩阵变成$|A-\lambda_a*I|$，求出的特征值λ_b离λ_a最远，故也是λ_1或λ_{501}之一。将两个特征值比较大小；大的是λ_{501}，小的是λ_1。

2. 利用反幂法求出矩阵A的模最小的特征值λ_s。

3. $\{\lambda_{i_k}\}$是最接近$\{\mu_k\}$的特征值，因此可以利用带原点平移的反幂法，将矩阵变成$|A-\mu_k*I|$，求出模最小的特征值，即是$\{\lambda_{i_k}\}$。

4. 谱范数条件数$cond(A)_2$是模最小的特征值和模最大的特征值的绝对值比值，由（1）问结果可求得。

5. 将矩阵A进行 LU 分解，$A=L*U$，得到$detA=detU$。U为上三角矩阵，$detA$为对角线元素的乘积。

算法流程：

1. 幂法求解模最大的特征值

a. 初始化迭代向量$u_0=[1,1,\cdots,1]$。

b. 开始迭代

c. 计算上一次迭代得到的u_{k-1}的 2 范数$\eta_{k-1}=\sqrt{u_{k-1}^T*u_{k-1}}$，并将$u_{k-1}$归一化得到$y_{k-1}$。计算矩阵$A$与归一化迭代向量$y_{k-1}$的乘积，即为$u_k$。计算$u_k$和$y_{k-1}$的内积$\beta_k=u_k^T*y_{k-1}$。

d. 终止迭代：如果$\frac{|\beta_k-\beta_{k-1}|}{|\beta_k|}\le\varepsilon$，则跳出循环；$\beta_k$作为所求的特征值$\lambda$，$y_{k-1}$作为对应的特征向量。如果不满足终止条件，则返回上一步，重新迭代。

2. 反幂法求解模最小的特征值

a. 初始化迭代向量$u_0 = [1,1,\cdots,1]$。

b. 对矩阵A进行 LU 分解，得到上三角矩阵U和下三角矩阵L。

c. 开始迭代

d. 计算上一次迭代得到的u_{k-1}的 2 范数$\eta_{k-1} = \sqrt{u_{k-1}^T * u_{k-1}}$，并将$u_{k-1}$归一化得到$y_{k-1}$。

e. 求解$A * u_k = y_{k-1}$：利用 LU 分解的结果，首先求解$Ly = b$, 然后求解$Ux = y$，得到的x即为u_k，计算u_k和y_{k-1}的内积$\beta_k = u_k^T * y_{k-1}$。

f. 终止迭代：如果$\frac{|\beta_k - \beta_{k-1}|}{|\beta_k|} \le \varepsilon$，则跳出循环；$1/\beta_k$作为所求的特征值$\lambda_s$，$y_{k-1}$作为对应的特征向量。如果不满足终止条件，则返回上一步，重新迭代。

计算程序

```
// Question1_new.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
// 编程环境：VS2019

#include <iostream>
#include <math.h>
#include <iomanip>

using namespace std;

constexpr auto N = 501;   // 矩阵阶数;;

double epsilon = 1e-12;   // 精度水平=1e-12

double A[5][N] = { 0 }; // 定义对称矩阵A
double Temp[5][N] = { 0 };

// 函数声明
double PowerMethod(double T[][N]);
double AntiPowerMethod(double T[][N]);
void LUresolve(double T[][N]);
int GetMin(int i, int j);
int GetMax(int i, int j);

int GetMin(int i, int j) {
    if (i <= j) return i;
    else return j;
}
```

```
int GetMax(int i, int j) {
    if (i > j) return i;
    else return j;
}

// 幂法的实现
double PowerMethod(double T[][N]) {
    int i, j, m, n;
    double u[N] = { 0 };    // 迭代向量
    double y[N] = { 0 };    // 归一化迭代向量
    double beta_last = 0;   // k-1时的β
    double beta_now = 0;    // k时的β
    for (i = 0; i < N; i++)
        u[i] = 1;   // 初始化迭代向量
    while (1) {// 开始迭代
        double Sum = 0;
        for (i = 0; i < N; i++)
            Sum += u[i] * u[i];
        double Mo = sqrt(Sum);  // 得到u(k-1)的模
        // 归一化迭代向量，得到y(k-1)
        for (i = 0; i < N; i++)
            y[i] = u[i] / Mo;
        for (i = 0; i < N; i++)
            u[i] = 0;

        // 计算u(k)：A与y(k-1)的乘积
        for (i = 2; i < N + 2; i++) {
            m = GetMin(i, 4);
            n = GetMin(i, N);
            for (j = i - m; j < n; j++)
                u[i - 2] = u[i - 2] + T[i - j][j] * y[j];
        }
        // 计算k-1时的β：yT(k-1)和u(k)的乘积
        for (i = 0; i < N; i++)
            beta_now = beta_now + y[i] * u[i];
        if (fabs(beta_now - beta_last) / fabs(beta_now) <= epsilon)
            return beta_now;    // 迭代结束，得到主特征值
        else {
            beta_last = beta_now;
            beta_now = 0;
            // cout << "beta_last = " << beta_last << endl;
        }
    }
}
```

```
// 反幂法的实现
double AntiPowerMethod(double T[][N]) {
    int i, j, k, m, n;
    double u[N] = { 0 };    // 迭代向量
    double y[N] = { 0 };    // 归一化的迭代向量
    double Temp_1[N] = { 0 };
    double beta_last = 0;
    double beta_now = 0;

    for (i = 0; i < N; i++)
        u[i] = 1;   // 初始化迭代向量
    // 对T作LU分解
    LUresolve(T);
    while (1) {
        double Sum = 0;
        for (i = 0; i < N; i++)
            Sum += u[i] * u[i];
        double Mo = sqrt(Sum);  // 得到u(k-1)的模
        // cout << "Mo = " << Mo << endl;
        // 归一化迭代向量，得到y(k-1)
        for (i = 0; i < N; i++)
            y[i] = u[i] / Mo;
        for (i = 0; i < N; i++)
            u[i] = 0;
        // 求解Ly=b
        Temp_1[0] = y[0];
        for (i = 1; i < N; i++) {
            double sum = 0;
            for (m = GetMax(1, i - 1); m <= i; m++)
                sum += T[i - m + 3][m - 1] * Temp_1[m - 1];
            Temp_1[i] = y[i] - sum;
        }
        // 求解Ux=y
        u[N - 1] = Temp_1[N - 1] / T[2][N - 1];
        for (i = N - 2; i >= 0; i--) {
            if (i + 2 >= N) n = N - 1;
            else n = i + 2;
            double sum = 0;
            for (j = i + 1; j <= n; j++)
                sum += T[i - j + 2][j] * u[j];
            u[i] = (Temp_1[i] - sum) / T[2][i];
        }
        // 计算k-1时的β：yT(k-1)和u(k)的乘积
```

```
        for (i = 0; i < N; i++)
            beta_now = beta_now + y[i] * u[i];
        if (fabs(beta_now - beta_last) / fabs(beta_now) <= epsilon)
            return 1 / beta_now;    // 迭代结束，得到主特征值
        else {
            beta_last = beta_now;
            beta_now = 0;
            // cout << "beta_last = " << beta_last << endl;
        }

    }
}

// LU分解的实现
void LUresolve(double T[][N]) {
    int i, j, p, k, t;
    for (i = 3; i < 5; i++)
        T[i][0] = T[i][0] / T[2][0];
    for (k = 2; k <= N; k++) {
        p = GetMin(k + 2, N);
        for (j = k; j <= p; j++) {// 得到U
            double sum = 0;
            for (t = GetMax(1, GetMax(k - 2, j - 2)); t < k; t++)
                sum += T[k - t + 2][t - 1] * T[t - j + 2][j - 1];
            T[k - j + 2][j - 1] = T[k - j + 2][j - 1] - sum;
        }
        if (k < N) {
            p = GetMin(k + 2, N);
            for (i = k + 1; i <= p; i++) {
                double sum = 0;
                for (t = GetMax(1, GetMax(i - 2, k - 2)); t < k; t++)
                    sum += T[i - t + 2][t - 1] * T[t - k + 2][k - 1];
                T[i - k + 2][k - 1] = T[i - k + 2][k - 1] - sum;
                T[i - k + 2][k - 1] = T[i - k + 2][k - 1] / T[2][k - 1];
            }
        }
    }
}

int main()
{
    cout << "Hello World!\n";
    int i, j, k;
    for (i = 0; i < N; i++) {
```

```
        // 构造对称矩阵A
        k = i + 1;
        A[0][i] = -0.064;
        A[1][i] = 0.16;
        A[2][i] = (1.64 - 0.024 * k) * sin(0.2 * k) - 0.64 * exp(0.1 /
k);
        A[3][i] = 0.16;
        A[4][i] = -0.064;
    }
    A[0][0] = A[0][1] = A[1][0] = 0;
    A[3][N - 1] = A[4][N - 1] = 0;
    A[4][N - 2] = 0;

    // 复制矩阵A，用于求解
    for (i = 0; i < 5; i++) {
        for (j = 0; j < N; j++)
            Temp[i][j] = A[i][j];
    }

    // 幂法求解主特征值
    double Lamd1 = PowerMethod(Temp);

    // 构造带原点平移的矩阵Temp
    for (i = 0; i < 5; i++) {
        for (j = 0; j < N; j++) {
            if (i == 2) Temp[i][j] = A[i][j] - Lamd1;
            else Temp[i][j] = A[i][j];
        }
    }
    // Lamd1模值最大，为最大值最小值之一
    // Lamd2离Lamd1最远，故也为最大值最小值之一
    double Lamd2 = PowerMethod(Temp) + Lamd1;

    cout << setiosflags(ios::scientific);   // 输出E型数
    // cout << setprecision(12) << "Lamd1 = " << Lamd1 << endl;
    // cout << setprecision(12) << "Lamd2 = " << Lamd2 << endl;

    double Lamd_1, Lamd_501;
    if (Lamd1 >= Lamd2) {// 比较特征值大小
        Lamd_1 = Lamd2;
        Lamd_501 = Lamd1;
    }
    else {
        Lamd_1 = Lamd1;
```

```
        Lamd_501 = Lamd2;
    }
    cout << setprecision(12) << "Lamd_1 = " << Lamd_1 << endl;
    cout << setprecision(12) << "Lamd_501 = " << Lamd_501 << endl;

    // 复制矩阵A，用于求解
    for (i = 0; i < 5; i++) {
        for (j = 0; j < N; j++)
            Temp[i][j] = A[i][j];
    }

    // 反幂法求解模最小的特征值
    double Lamd_s = AntiPowerMethod(Temp);
    cout << setprecision(12) << "Lamd_s = " << Lamd_s << endl;

    // 第二问
    for (k = 1; k < 40; k++) {
        double U_k = Lamd_1 + k * (Lamd_501 - Lamd_1) / 40;
        // 构造带位移的矩阵Tem
        for (i = 0; i < 5; i++) {
            for (j = 0; j < N; j++) {
                if (i == 2)
                    Temp[i][j] = A[i][j] - U_k;
                else
                    Temp[i][j] = A[i][j];
            }
        }
        double Lam_ik = AntiPowerMethod(Temp) + U_k;
        cout << "when k = " << k << "  " << setprecision(12) << "Lam_ik="
<< Lam_ik << endl;
    }

    // 第三问
    cout << "cond(A)2 = " << setprecision(12) << fabs(Lamd1 / Lamd_s) <<
endl;

    for (i = 0; i < 5; i++) {
        for (j = 0; j < N; j++)
            Temp[i][j] = A[i][j];
    }
    LUresolve(Temp);
    double det = 1;
    for (i = 0; i < N; i++) det = det * Temp[2][i];
    cout << "detA = " << setprecision(12) << det << endl;
```

```
    return 0;
}
```

运行结果

Microsoft Visual Studio 调试控制台

```
Hello World!
Lamd_1 = -1.067258214243e+01
Lamd_501 = 9.730828409770e+00
Lamd_s = -5.557910794230e-03
when k = 1  Lam_ik=-1.018293403315e+01
when k = 2  Lam_ik=-9.585707425068e+00
when k = 3  Lam_ik=-9.172672423928e+00
when k = 4  Lam_ik=-8.652284007898e+00
when k = 5  Lam_ik=-8.093483808675e+00
when k = 6  Lam_ik=-7.566937453586e+00
when k = 7  Lam_ik=-7.119684648691e+00
when k = 8  Lam_ik=-6.611764339397e+00
when k = 9  Lam_ik=-6.066103226595e+00
when k = 10  Lam_ik=-5.575417562838e+00
when k = 11  Lam_ik=-5.114083529812e+00
when k = 12  Lam_ik=-4.548997078815e+00
when k = 13  Lam_ik=-4.015219580729e+00
when k = 14  Lam_ik=-3.543244165725e+00
when k = 15  Lam_ik=-3.041090018133e+00
when k = 16  Lam_ik=-2.505720773926e+00
when k = 17  Lam_ik=-2.003230769564e+00
when k = 18  Lam_ik=-1.487964308096e+00
when k = 19  Lam_ik=-9.819964436995e-01
when k = 20  Lam_ik=-4.691213015575e-01
when k = 21  Lam_ik=4.726132411530e-02
when k = 22  Lam_ik=5.465796646731e-01
when k = 23  Lam_ik=1.052898962693e+00
when k = 24  Lam_ik=1.589445881881e+00
when k = 25  Lam_ik=2.081232695829e+00
```

Microsoft Visual Studio 调试控制台

```
when k = 17   Lam_ik=-2.003230769564e+00
when k = 18   Lam_ik=-1.487964308096e+00
when k = 19   Lam_ik=-9.819964436995e-01
when k = 20   Lam_ik=-4.691213015575e-01
when k = 21   Lam_ik=4.726132411530e-02
when k = 22   Lam_ik=5.465796646731e-01
when k = 23   Lam_ik=1.052898962693e+00
when k = 24   Lam_ik=1.589445881881e+00
when k = 25   Lam_ik=2.081232695829e+00
when k = 26   Lam_ik=2.603457745271e+00
when k = 27   Lam_ik=3.109795990741e+00
when k = 28   Lam_ik=3.613620867692e+00
when k = 29   Lam_ik=4.091378510451e+00
when k = 30   Lam_ik=4.603035378279e+00
when k = 31   Lam_ik=5.144584028615e+00
when k = 32   Lam_ik=5.594906348083e+00
when k = 33   Lam_ik=6.080933857027e+00
when k = 34   Lam_ik=6.680354092112e+00
when k = 35   Lam_ik=7.293877448127e+00
when k = 36   Lam_ik=7.717111714236e+00
when k = 37   Lam_ik=8.225220014050e+00
when k = 38   Lam_ik=8.648666065193e+00
when k = 39   Lam_ik=9.254200344575e+00
cond(A)2 = 1.920250708865e+03
detA = 2.772786141752e+118

C:\Users\dell\Desktop\数值分析\计算实习大作
.exe (进程 18224)已退出，代码为 0。
按任意键关闭此窗口. . .
```

结果分析

1. 在编程时发现λ_1与参考答案偏差较大，经过测试其与迭代向量的初始化有关。而λ_{501}与迭代向量的初始化取值没有明显关系，总能迭代到准确的值。

2. 通过将迭代向量取不同的初始值发现对迭代次数（速度）没有明显影响。

3. 为了避免初始向量对程序的影响，可以先对 A 做平移变换再求λ_1。