北京航空航天大學

BEIHANG UNIVERSITY

数值分析大作业

计算实习题（2）

院（系）名称	宇航学院
学　　　　号	ZY2115107
学 生 姓 名	段诗阳

2021 年 11 月 17 日

一、题目要求

1. 求出矩阵A经过拟上三角化后的矩阵$A^{(n-1)}$

2. 求出对$A^{(n-1)}$进行带双步位移的 QR 分解后的矩阵

3. 求出A的全部特征值（包含实数特征值和复数特征值）

4. 求出A相应于实特征值的特征向量

二、算法流程

1. 拟上三角化

定义并构造函数 A[N][N];

记 $A^{(1)}$=A，对于 r=1,2,…,N-2 作如下循环：

（1） 判断 A[i][r]是否全为零（i=r+2,r+3,…,N）,若全为零，则记 $A^{(r+1)}$=$A^{(r)}$,进入下一次循环，否则转入（2）

（2） 计算

$$d_r = \sqrt{\sum_{i=r+1}^{N} (a_{ir}^r)^2}$$

$$c_r = -sgn(a_{r+1}^r) * d_r$$

$$h_r = c_r^2 - c_r a_{r+1,r}^r$$

$$u_r = \left(0, \dots, 0, a_{r+1,r}^r - c_r, a_{r+2,r}^r, \dots, a_{nr}^r\right)^T \in R^n$$

$$p_r = A^{r^T} * \frac{u_r}{h_r}$$

$$q_r = A^r * \frac{\mathrm{u_r}}{h_r}$$

$$\mathrm{t_r} = p_r^T * \frac{u_r}{h_r}$$

$$\omega_r = q_r - t_r * u_r$$

$$A^{r+1} = A^r - \omega_r * u_r^T - u_r * p_r^T$$

进入下一次循环，当算法执行完毕，得到了和原矩阵A相似的拟上三角化矩阵A^{n-1}

2. 对矩阵A^{n-1}作带双步位移的 QR 分解

记 A_1=A^{n-1},令 k=1,m=N，k 为迭代次数，给定最大迭代次数 L 和精度水平ϵ，当

k<L 时，作如下循环：

（1） 如果 m=0，迭代完成，退出；

（2） 如果 m=1，矩阵为 1 阶矩阵，得到 A 的一个特征值 A[m][m]，迭代完成，退出；

（3） 如果 m=2，矩阵为 2 阶矩阵，得到 A 的两个特征值 s_1 和 s_2，迭代完成，退出；

（4） 如果$|A[m][m-1]| \le \epsilon$，得到 A 的一个特征值 A[m][m]，令 m=m-1，降阶并继续，否则转（5）；

（5） 如果$|A[m-1][m-2]| \le \epsilon$，得到 A 的两个特征值 s_1 和 s_2，令 m=m-2，降阶并继续，否则转（6）；

二阶子阵$D_k = \begin{matrix} A[m-1][m-1] & A[m-1][m] \\ A[m][m-1] & A[m][m] \end{matrix}$

s_1 和 s_2 即是上述子阵的特征值

（6） 计算M_k矩阵和A_{k+1}矩阵

$$s = A[m-1][m-1] + A[m][m]$$

$$t = A[m-1][m-1] * A[m][m] - A[m][m-1] * A[m-1][m]$$

$$M_k = A_k^2 - sA_k + tI$$

$$M_k = Q_k * R_k$$

$$A_{k+1} = Q^T A_k Q^k$$

结束循环后即得到矩阵A的全部特征值（包含实特征值和复数特征值）。

3. 求M_k和A_{k+1}的方法和拟上三角化类似

定义$B_1 = M_k$，$C_1 = A_k$，对于 r=1,2,⋯,m-1，执行如下循环：

（1） 判断 B[i][r]是否全为零（i=r+1,r+2,⋯,m），若全为零，则记$B_{r+1} = B_r$，$C_{r+1} = C_r$，进入下一次循环，否则转入（2）；

（2） 计算

$$d_r = \sqrt{\sum_{i=r}^{m} \left(b_{ir}^{(r)}\right)^2}$$

$$c_r = -sgn\left(b_{rr}^{(r)}\right) * d_r$$

$$h_r = c_r^2 - c_r b_{rr}^{(r)}$$

$$u_r=\left(0,\dots,0,b_{rr}^{(r)}-c_r,b_{r+1,r}^{(r)},\dots,b_{mr}^{(r)}\right)^T\in R^m$$

$$v_r=\frac{B_r^T u_r}{h_r}$$

$$B_{r+1}=B_r-u_r v_r^T$$

$$p_r=\frac{C_r^T u_r}{h_r}$$

$$q_r=\frac{C_r u_r}{h_r}$$

$$t_r=\frac{p_r^T u_r}{h_r}$$

$$\omega_r=q_r-t_r u_r$$

$$C_{r+1}=C_r-\omega_r u_t^T-u_r p_r^T$$

进入下一次循环，当算法执行完毕，得到$A_{k+1}=C_m$

4. 列主元 Gauss 消元法求解特征向量

在得到特征值λ后，求解特征向量即为求解$Ax=\lambda x$的解，进一步转化为求解$(A-\lambda I)x=0$的解，构造矩阵$B=A-\lambda I$，利用列主元 Gauss 消元法可以求得解向量即为矩阵A对应于特征值λ的特征向量。列主元 Gauss 的算法流程如下：

记$A^{(1)}=A=a_{i,j}^{(1)},(i,j=1,2,\dots,N)$

消元过程：

对于 k=1,2,⋯,N-1，执行

（1） 选行号i_k，使得$\left|a_{i_k k}^{(k)}\right|=\max\limits_{k\le i\le N}\left|a_{ik}^{(k)}\right|$

（2） 交换$a_{kj}^{(k)}$和$a_{i_k j}^{(k)}$所在行的全部数值

（3） 对于 i=k+1,k+2,⋯,N，计算

$$m_{ik}=\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$$

$$a_{ij}^{(k+1)}=a_{ij}^{(k)}-m_{ik}a_{kj}^{(k)},\quad (j=k+1,k+2,\dots,N)$$

回代过程：

（1） 如果$A[N-1][N-1]\le\epsilon,A[N-1][n-2]\le\epsilon$，则转入（2），否则转入（3）

（2） 令

$$x[N]=1$$

$$x[N-1]=1$$

对于$k=N-2,N-3,\dots,0$，执行

a. 置 Sum=0;

b. 对于$j=k+1,\dots,N$，记$Sum+=A[k][j]*x[j]$;

c. $x[k]=-Sum/A[k][k]$

（3） 令

$$x[N]=1$$

对于$k=N-1,N-2,\dots,0$，执行

a. 置 Sum=0;

b. 对于$j=k+1,\dots,N$，记$Sum+=A[k][j]*x[j]$;

c. $x[k]=-Sum/A[k][k]$

消元和回代过程全部完成后，对所得的解向量作归一化即为所求特征向量。

三、计算程序

```
// Quesiton2_new.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//
#include <iostream>
#include <math.h>
#include <iomanip>

using namespace std;

constexpr auto N = 10;      // 矩阵维数
constexpr auto L = 100000;  // 迭代最大次数
constexpr auto epsilon = 1e-12; // 精度水平

void UpHessenberg(double T[N][N]);    // 拟上三角化
void CreatA(double T[N][N]);     // 构造矩阵A
int Sgn(double d);     // 符号函数
void PrintAij(double T[N][N]);      // 在终端打印矩阵
void PrintAi(double x[N]);      // 在终端打印向量
void QRsolve(double A[N][N], double Mk[N][N], int m);       // QR方法
int DoubleStepQRsolve(double T[N][N], double Eigenvalue[N][2]);  // 双步位移QR方法
```

```
void SolveRoots(double a, double b, double c, double lamda[2][2]); //
求解一元二次方程的根
void GetMk(double T[N][N], double s, double t, double M_k[N][N], int m);
// 获取矩阵的M_k矩阵

// 构造矩阵A
void CreatA(double A[N][N]) {
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            if (i != j) A[i - 1][j - 1] = sin(0.5 * i + 0.2 * j);
            else A[i - 1][j - 1] = 1.52 * cos(i + 1.2 * j);
        }
    }
}

// 拟上三角化
void UpHessenberg(double T[N][N]) {
    int i, j, r;
    double d_r, c_r, h_r, t_r, sum;
    double u_r[N], p_r[N], q_r[N], w_r[N];
    for (r = 1; r <= N - 2; r++) {
        int flTg = 0;   // 判断全为零的标志
        for (i = r + 2; i <= N; i++) {
            if (T[i - 1][r - 1] != 0) {
                flTg = 1;
                break;
            }
        }
        if (flTg == 0) continue;

        d_r = 0;    // d_r置零
        for (i = r + 1; i <= N; i++) d_r += T[i - 1][r - 1] * T[i - 1][r
- 1];
        d_r = sqrt(d_r);
        if (T[r][r - 1] == 0) c_r = d_r;
        else c_r = -Sgn(T[r][r - 1]) * d_r;
        h_r = c_r * c_r - c_r * T[r][r - 1];
        for (i = 1; i <= N; i++) {
            if (i <= r) u_r[i - 1] = 0;
            else if (i == r + 1) u_r[i - 1] = T[r][r - 1] - c_r;
            else u_r[i - 1] = T[i - 1][r - 1];
        }
        for (i = 1; i <= N; i++) {
            sum = 0;
```

```
            for (j = 1; j <= N; j++) sum += T[j - 1][i - 1] * u_r[j - 1];
            p_r[i - 1] = sum / h_r;
        }
        for (i = 1; i <= N; i++) {
            sum = 0;
            for (j = 1; j <= N; j++) sum += T[i - 1][j - 1] * u_r[j - 1];
            q_r[i - 1] = sum / h_r;
        }
        sum = 0;
        for (i = 1; i <= N; i++) {
            sum += p_r[i - 1] * u_r[i - 1];
        }
        t_r = sum / h_r;
        for (i = 1; i <= N; i++) {
            w_r[i - 1] = q_r[i - 1] - t_r * u_r[i - 1];
        }
        // 求T(r+1)
        for (i = 1; i <= N; i++) {
            for (j = 1; j <= N; j++) {
                T[i - 1][j - 1] = T[i - 1][j - 1] - w_r[i - 1] * u_r[j - 1] - u_r[i - 1] * p_r[j - 1];
            }
        }
    }
}

// 符号函数
int Sgn(double d) {
    if (d < 0) return -1;
    else return 1;
}

// 在终端打印矩阵
void PrintAij(double T[N][N]) {
    cout << setiosflags(ios::scientific);   // 输出E型数
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            cout << setprecision(12) << T[i - 1][j - 1] << " ";
        }
        cout << endl;
    }
}

// 在终端打印向量
```

```cpp
void PrintAi(double x[N]) {
    cout << "[";
    for (int i = 0; i < N; i++) {
        cout << x[i];
        if (i < N - 1) cout << ",";
    }
    cout << "]" << endl;
}

// 求解一元二次方程的根
void SolveRoots(double a, double b, double c, double lamda[2][2]) {
    // 一元二次方程的系数a,b,c
    // lamda[2][2]保存的方程的实根/复根
    double delta = b * b - 4 * a * c;
    if (delta >= 0) {// 方程有两个实根
        lamda[0][0] = (-b - sqrt(delta)) / (2 * a);
        lamda[0][1] = 0;
        lamda[1][0] = (-b + sqrt(delta)) / (2 * a);
        lamda[1][1] = 0;
    }
    else {
        lamda[0][0] = -b / (2 * a);
        lamda[0][1] = -sqrt(-delta) / (2 * a);
        lamda[1][0] = -b / (2 * a);
        lamda[1][1] = sqrt(-delta) / (2 * a);
    }
}

// 获取矩阵的M_k矩阵
void GetMk(double T[N][N], double s, double t, double M_k[N][N], int m)
{
    int i, j;
    int k;

    for (i = 0; i < m; i++)
        for (j = 0; j < m; j++)
            M_k[i][j] = 0;

    for (i = 0; i < m; i++) {
        for (j = 0; j < m; j++) {
            double sum = 0;
            for (k = 0; k < m; k++)
                sum += T[i][k] * T[k][j];
            M_k[i][j] = sum - s * T[i][j];
```

```
            if (i == j) M_k[i][j] += t;
        }
    }
}

// QR方法
void QRsolve(double A[N][N], double Mk[N][N], int m) {
    int i, j, r;
    double dr, cr, hr, tr;
    double sum;
    double B[N][N], C[N][N];
    double ur[N], vr[N], pr[N], qr[N], wr[N];

    for (i = 1; i <= m; i++) {
        for (j = 1; j <= m; j++) {
            B[i - 1][j - 1] = Mk[i - 1][j - 1];
            C[i - 1][j - 1] = A[i - 1][j - 1];
        }
    }

    for (r = 1; r < m; r++) {
        int flag = 0;
        for (i = r + 1; i <= m; i++) {
            if (B[i - 1][r - 1] != 0) {
                flag = 1;
                break;
            }
            else flag = 0;
        }
        if (flag != 0) {
            dr = 0;
            for (i = r; i <= m; i++) dr += B[i - 1][r - 1] * B[i - 1][r
- 1];
            dr = sqrt(dr);
            if (B[r - 1][r - 1] == 0) cr = dr;
            else cr = -Sgn(B[r - 1][r - 1]) * dr;
            hr = cr * cr - cr * B[r - 1][r - 1];
            for (i = 1; i <= m; i++) {
                if (i < r) ur[i - 1] = 0;
                else if (i == r) ur[i - 1] = B[r - 1][r - 1] - cr;
                else ur[i - 1] = B[i - 1][r - 1];
            }
            for (i = 1; i <= m; i++) {
                sum = 0;
```

```
                for (j = 1; j <= m; j++) sum += B[j - 1][i - 1] * ur[j - 1];
                vr[i - 1] = sum / hr;
            }
            for (i = 1; i <= m; i++)
                for (j = 1; j <= m; j++)
                    B[i - 1][j - 1] = B[i - 1][j - 1] - ur[i - 1] * vr[j - 1];
            for (i = 1; i <= m; i++) {
                sum = 0;
                for (j = 1; j <= m; j++) sum += C[j - 1][i - 1] * ur[j - 1];
                pr[i - 1] = sum / hr;
            }
            for (i = 1; i <= m; i++) {
                sum = 0;
                for (j = 1; j <= m; j++) sum += C[i - 1][j - 1] * ur[j - 1];
                qr[i - 1] = sum / hr;
            }
            sum = 0;
            for (i = 1; i <= m; i++) sum += pr[i - 1] * ur[i - 1];
            tr = sum / hr;
            for (i = 1; i <= m; i++) wr[i - 1] = qr[i - 1] - tr * ur[i - 1];
            for (i = 1; i <= m; i++)
                for (j = 1; j <= m; j++)
                    C[i - 1][j - 1] = C[i - 1][j - 1] - wr[i - 1] * ur[j - 1] - ur[i - 1] * pr[j - 1];
        }
    }

    for (i = 1; i <= m; i++)
        for (j = 1; j <= m; j++)
            A[i - 1][j - 1] = C[i - 1][j - 1];
}

// 双步位移QR方法
int DoubleStepQRsolve(double T[N][N], double Eigenvalue[N][2]) {
    int k = 1, m = N;
    double Mk[N][N];

    while (k < L) {
        k = k + 1;
```

```
        if (m == 0) return 0;
        else if (m == 1) {// 矩阵有一个特征值
            Eigenvalue[m - 1][0] = T[0][0];
            m = 0;
            continue;
        }
        else if (m == 2) {// 矩阵有两个特征值
            double s = -(T[m - 2][m - 2] + T[m - 1][m - 1]);
            double t = T[m - 2][m - 2] * T[m - 1][m - 1] - T[m - 1][m -
2] * T[m - 2][m - 1];
            double lamda[2][2];
            SolveRoots(1, s, t, lamda);
            // 得到A的两个特征值
            Eigenvalue[m - 2][0] = lamda[0][0]; Eigenvalue[m - 2][1] =
lamda[0][1];
            Eigenvalue[m - 1][0] = lamda[1][0]; Eigenvalue[m - 1][1] =
lamda[1][1];
            m = 0;
            continue;
        }
        else if (fabs(T[m - 1][m - 2]) <= epsilon) {
            Eigenvalue[m - 1][0] = T[m - 1][m - 1]; // 得到A的一个特征值
            m = m - 1;
            continue;
        }
        else {
            double s = -(T[m - 2][m - 2] + T[m - 1][m - 1]);
            double t = T[m - 2][m - 2] * T[m - 1][m - 1] - T[m - 1][m -
2] * T[m - 2][m - 1];

            if (fabs(T[m - 2][m - 3]) <= epsilon) {
                double lamda[2][2];
                SolveRoots(1, s, t, lamda);
                // 得到A的两个特征值
                Eigenvalue[m - 2][0] = lamda[0][0]; Eigenvalue[m - 2][1]
= lamda[0][1];
                Eigenvalue[m - 1][0] = lamda[1][0]; Eigenvalue[m - 1][1]
= lamda[1][1];
                m = m - 2;
                continue;
            }
            else {
                // 获取A_k矩阵的M_k矩阵
                GetMk(T, s, t, Mk, m);
```

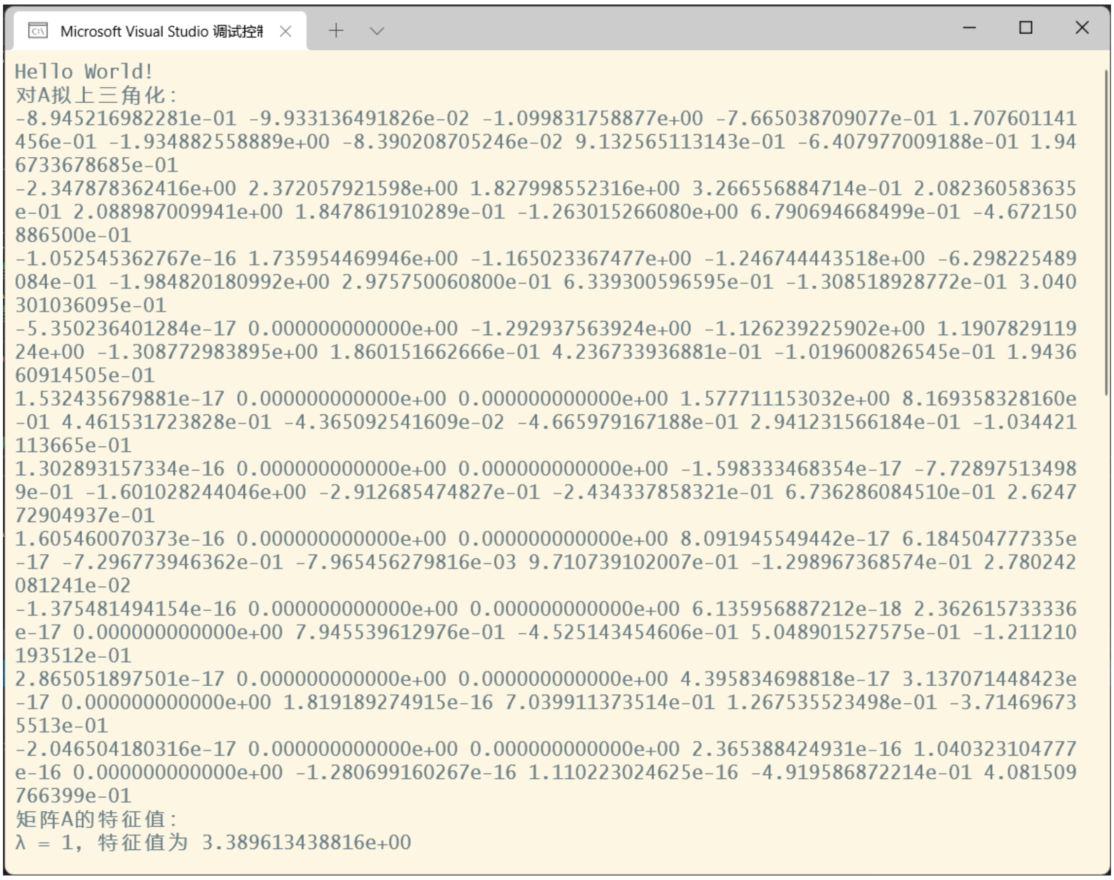
```cpp
                // 对M_k作QR分解
                QRsolve(T, Mk, m);
            }
        }
        // cout << "k=" << k << endl;
    }
    cout << "Error! 达到迭代最大次数，k=" << k << endl;
    return 0;
}

// Gauss消去法
void Gauss(double T[N][N], double ans[N]) {
    int k, i, j;
    double temp;

    // 消元过程
    for (k = 0; k < N - 1; k++) {
        int flag = k;
        temp = fabs(T[k][k]);
        for (j = k + 1; j < N; j++)
            if (fabs(T[j][k]) > temp) {
                flag = j;
                temp = fabs(T[j][k]);   // 找出主元
            }
        if (flag != k)
            for (j = k; j < N; j++)
                swap(T[k][j], T[flag][j]);  // 交换主元所在行全部元素
        for (i = k + 1; i < N; i++) {// 消元
            temp = T[i][k] / T[k][k];
            for (j = k + 1; j < N; j++)
                T[i][j] -= temp * T[k][j];
        }
    }
    //PrintAij(T);
    // 回代过程
    for (k = N - 1, ans[N - 1] = 1; k >= 1; k--) {
        for (j = k + 1, temp = 0; j <= N; j++)
            temp += T[k - 1][j - 1] * ans[j - 1];
        ans[k - 1] = -temp / T[k - 1][k - 1];
    }
    // 特征向量归一化
    temp = 0;
    for (i = 0; i < N; i++)
        temp += pow(ans[i], 2);
```

```
    temp = sqrt(temp);
    for (i = 0; i < N; i++)
        ans[i] = ans[i] / temp;
}


int main() {
    std::cout << "Hello World!\n";

    int i, j;

    double A[N][N] = { 0 };        // 定义实矩阵A
    CreatA(A);
    //cout << "实矩阵A为: " << endl;
    //PrintAij(A);

    UpHessenberg(A);
    cout << "对A拟上三角化:  " << endl;
    PrintAij(A);

    // 双步位移QR方法
    double Eigenvalue[N][2];
    for (i = 1; i <= N; i++)
        for (j = 1; j <= 2; j++)
            Eigenvalue[i - 1][j - 1] = 0;
    DoubleStepQRsolve(A, Eigenvalue);

    // 在终端打印特征值
    cout << "矩阵A的特征值:  " << endl;
    for (i = 1; i <= N; i++) {
        cout << setiosflags(ios::scientific);   // 输出E型数
        if (Eigenvalue[i - 1][1] != 0)
            cout << "λ = " << i << ", 特征值为 (" << setprecision(12) <<
Eigenvalue[i - 1][0] << ", "
            << Eigenvalue[i - 1][1] << ")" << endl;
        else
            cout << "λ = " << i << ", 特征值为 " << setprecision(12) <<
Eigenvalue[i - 1][0] << endl;
    }

    // 求解实特征值对应的特征向量
    double Temp[N][N];  // 系数矩阵
    double x[N];        // 特征向量
    for (int k = 1; k <= N; k++) {
```

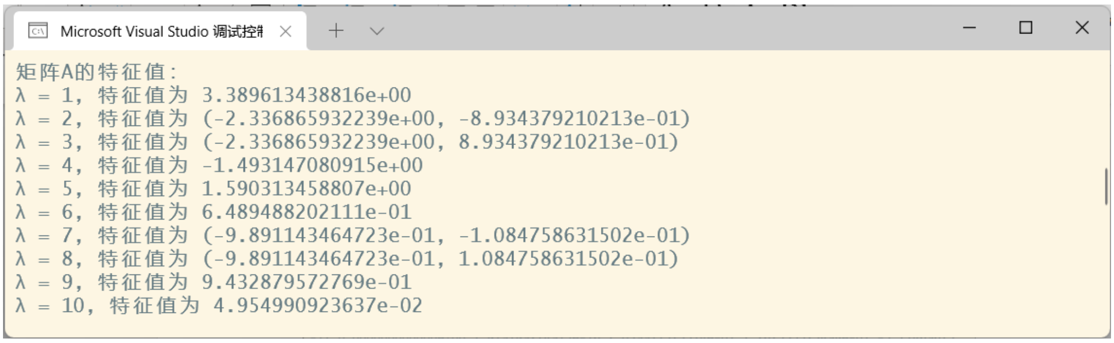
```
        if (Eigenvalue[k - 1][1] == 0) {// 实特征值
            // 构造矩阵A
            CreatA(A);
            // 构造矩阵Temp=A-λI
            for (i = 0; i < N; i++)
                for (j = 0; j < N; j++) {
                    if (i == j) Temp[i][j] = A[i][j] - Eigenvalue[k -
1][0];
                    else Temp[i][j] = A[i][j];
                }
            // 高斯消去法求解特征向量
            cout << "λ = " << setprecision(4) << Eigenvalue[k - 1][0] <<
endl;
            cout << "对应的特征向量为: " << endl;
            Gauss(Temp, x);     // 列主元Guass消元法
            PrintAi(x);         // 打印特征向量
        }
    }
}
```

四、运行结果

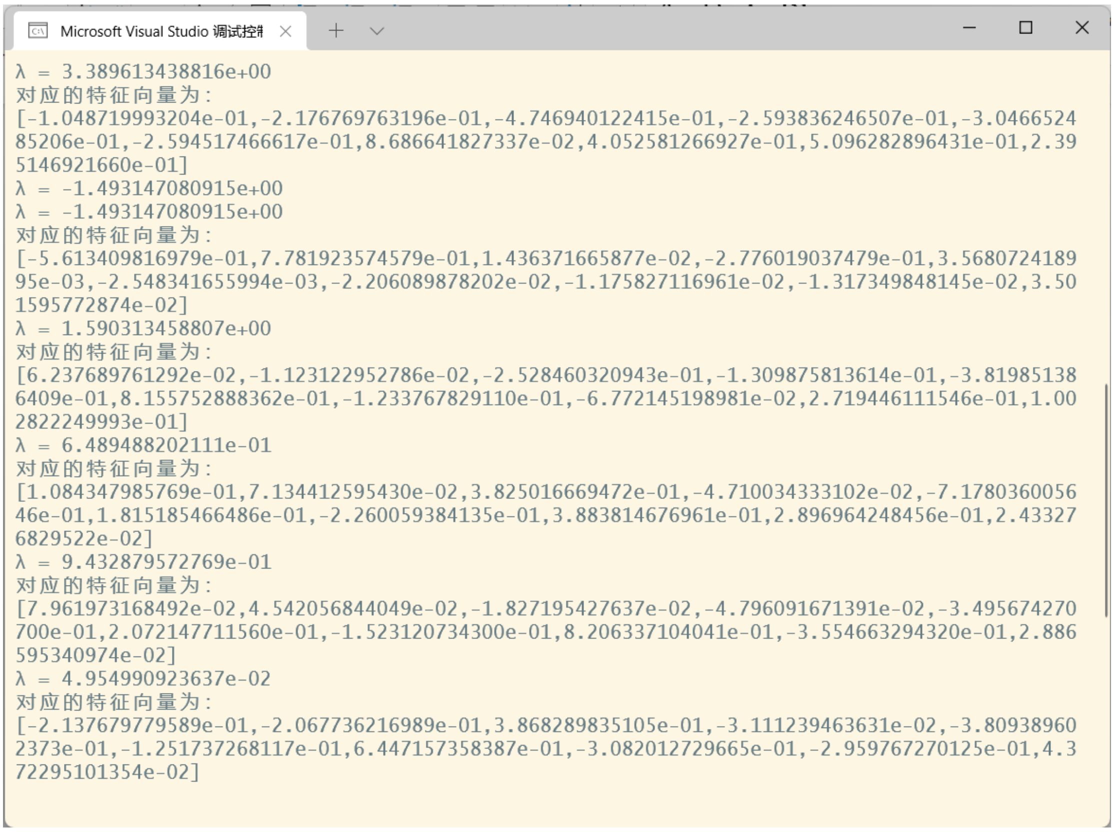
1、对矩阵A进行拟上三角化的结果：



2、矩阵A的全部特征值：



3、矩阵A对应于实特征值的特征向量：



五、结果总结

在对一般的方针进行一次 QR 迭代需要的运算复杂度是$O(n^3)$，但是在进行 QR 迭代前，先对 A 进行一定的变换，得到一个具有较多 0 元素的相似矩阵，则可以减少迭代的运算量。通过对拟上三角化的编程实现，我了解到拟上三角化通过得到一个 A 的一个上 Hessenberg 矩阵$A^{n-1} = H_{n-2} \dots H_2 H_1 A H_1 H_2 \dots H_{n-1}$，使得对 A 的 QR 分解变成了对 H 的 QR 分解，迭代过程为找到 N-1 个 Givens 矩阵，使得

$$H = G_{n-1} \dots G_2 G_1 G_1^T G_2^T \dots G_{n-1}^T$$

使得运算复杂度减少到$O(n^2)$，从而减少了运算量。

简单的 QR 分解中，A 的特征值是

$$|\lambda_1| \geq \cdots \geq |\lambda_{n-1}| > |\lambda_n|$$

随着迭代的增加，A^k的右下角的对角线元素a_{nn}^k逐渐收敛到特征值λ_n，收敛速度是$|\frac{\lambda_n}{\lambda_{n-1}}|$，由此可见如果引入一个常数，使得$A \rightarrow A - \mu I$，矩阵$A - \mu I$的收敛速度将变成$|\frac{\lambda_n - \mu}{\lambda_{n-1} - \mu}|$。如果$\mu$取值和 A 的特征值很接近则可以提高收敛速度。

在利用列主元的 Gauss 消元法求解方程组时，由于给定的特征值均为实特征值，其没有重根，每一特征值均对应一个特征向量。首先用高斯消元法将矩阵$A - \lambda I$化为上三角矩阵，其最后一行全为零，在反代时需要令解向量的最后一个元素为 1，即得到方程组的一个基础解系。值得注意的是最后的解向量需要进行归一化处理。