数值分析第二次大作业

院（系）名称：计算机学院
学 生 学 号：ZY2106339
学 生 姓 名：陈 铭 煊
授 课 教 师：谢 家 新

2021 年 11 月

1. 设计方案

题目描述

本次作业出自《数值分析（第四版）》（颜庆津 编著）计算实习说明书第二题：

试求矩阵$A=[a_{i,j}]_{10\times10}$的全部特征值，并对其中每一个实特征值求相应的特征向量，已知

$$a_{ij}=\begin{cases}\sin(0.5i+0.2j) & i\neq j\\ 1.52\cos(i+1.2j) & i=j\end{cases}\quad(i,j=1,2,\cdots,10)$$

说明：

1. 用幂法，反幂法和QR方法求矩阵特征值时，要求迭代的精度水平为$\varepsilon=10^{-12}$。
2. 打印以下内容：
    1. 全部源程序；
    2. 矩阵A经过拟上三角化后所得到的矩阵$A^{(n-1)}$；
    3. 对矩阵$A^{(n-1)}$实行QR方法迭代后所得的矩阵；
    4. 矩阵A的全部特征值$\lambda_i=(R_i,I_i)\ (i=1,2,\ldots,10)$，其中$R_i=\mathrm{Re}(\lambda_i)$，$I_i=\mathrm{Im}(\lambda_i)$，若$\lambda_i$是实数就令$I_i=0$；
    5. A的相应于实特征值的特征向量。
3. 采用e型输出实型数，并且至少显示12位有效数字。

算法分析

求特征值

观察到矩阵A的维度较小，且是要求出所有特征值，使用QR方法比幂法/反幂法更为合适。另外，A存在复数特征值，使用QR方法，能很大程度地避免复数运算。

这里我们采用了带双部位移的QR方法。其基本思路以及详细推导如书本P63-P64所示，其中引入了矩阵M来避免复数运算。这里简单整理成方便编程的步骤如下：

1. 使用矩阵的拟上三角化的算法，把矩阵$A\in R^{n\times n}$ $(n=10)$化为拟上三角矩阵；给定精度水平$\varepsilon(10^{-12})$。
2. 记$A_1=A^{(n-1)}=[a_{ij}^{(1)}]_{n\times n}$，令$m=n$。
3. 判断是否$m\leq 1$，如果是，则A的所有特征值计算完毕（$m=1$时，最后一个特征值$\lambda_1=a_{1,1}^k$），**结束计算**。如果否，转（3）。
4. 如果$|a_{m,m-1}^{(k)}|\leq\varepsilon$，那么我们得到了特征值$\lambda_m=a_{m,m}^k$，置$m:=m-1$，转（3）。否则转（5）。
5. 求二阶子阵

    $$\boldsymbol{D}_k=\begin{bmatrix}a_{m-1,m-1}^{(k)} & a_{m-1,m}^{(k)}\\ a_{m,m-1}^{(k)} & a_{mm}^{(k)}\end{bmatrix}$$

    的两个特征值s_1,s_2，也就是计算

    $$s=\operatorname{tr}(D_k)=a_{m-1,m-1}^{(k)}+a_{m,m}^{(k)}$$
    $$t=\det(D_k)=a_{m-1,m-1}^{(k)}a_{m,m}^{(k)}-a_{m,m-1}^{(k)}a_{m-1,m}^{(k)}$$

    再求二次方程：

$$\lambda^2 - s\lambda + t = 0$$

的两个根s_1, s_2。

6. 如果$m=2$或者$a_{m-1,m-2}^{(k)} \le \varepsilon$，则得到了两个特征值$s_1, s_2$，置$m := m-2$，转（3），否则转（7）。
7. 记$A_k = [a_{ij^{(k)}}]_{m\times m}(1 \le i, j \le m)$，计算:

$$M_k = A_k^2 - sA_k + tI \quad (I\text{是}m\text{阶单位矩阵})$$
$$M_k = Q_kR_k \quad (\text{对}M_k\text{作QR分解})$$
$$A_{k+1} = Q_k^T A_k Q_k$$

转（3）

在第一步中，我们需要求出矩阵A的拟上三角矩阵（Hessenberg矩阵），这是因为得到的矩阵和A有相同的特征值，同时也能让迭代收敛的速度更快。另外在判断元素下方是否全0时，只需看下方第一个元素即可。Hessenberg矩阵的求法与书本P61-P62基本相同，限于篇幅不再赘述。

在第七步中，可以稍稍进行化简，不必求出矩阵Q，再和矩阵A做乘法，这是因为

$$A_{k+1} = Q_k^T A_k Q^T = H_{n-1} \dots H_1 A_k H_1 \dots H_{n-1}$$

通过对Hessenberg矩阵$H = I - 2\frac{u \cdot u^T}{||u_k||}$进行代入，可以得到乘法运算次数只有$m^2$级别的算法，细节和书本P64-P65相同，不再赘述。

求实特征值的特征向量

求实特征值的特征向量有两种主要思路:

1. 将相似变换过程中所有变换矩阵相乘（即拟上三角化和QR分解中的所有Q矩阵），得到的矩阵的对应列向量就是特征向量;
2. 解齐次线性方程组$(A-\lambda I)x=0$，得到解空间，对解空间求出一组正交基;

第一种方法好处是不用解方程组，且得到的所有特征向量是正交的（因为每一个Q都是正交阵），缺点是需要做较多的矩阵乘法。此外在实践中发现，带双步位移的QR方法中进行了维度的缩减，导致得出的Q无法直接相乘；另外双步位移的优化中，二阶矩阵的解可能是两个实根，和原本的QR方法已经不同，这两个实特征值无法求出对应的特征向量。因此，只能放弃这种思路。

第二种方法缺陷主要在于需要对空间整理出特征向量，再进行正交化，单位化，比较麻烦。然而实践中发现所有的实特征值是唯一的，因此，$(A-\lambda I)x=0$的解空间维度不会超过1；很容易找到特征向量，而且所有得到的特征向量将会是正交的。

因此我们采用第二种方法。具体解齐次线性方程组$Ax=0$的方法类似于列主元高斯消去法，只是额外考虑对角元素为0的情况。这里整理如下:

消元过程:

设置$k:=1$，对于$s=1,2,\dots,n$（k枚举行号，s枚举列号）执行

1. 选择行号i_s，使得$|a_{i_s s}^{(s)}| = \max_{k\le i\le n}|a_{is}^{(s)}|$。
2. 如果$|a_{i_s s}^{(s)}| \le \varepsilon$，那么跳过；否则继续执行。
3. 交换$a_{kj}^{(s)}$和$a_{i_s j}^{(s)}(j=s,s+1,\dots,n)$的数值
4. 对于$i=k+1,k+2,\dots,n$计算:

   $m_{is} = a_{is}^{(s)}/a_{ks}^{(s)}$
   $a_{ij}^{(s+1)} = a_{ij}^{(s)} - m_{is}a_{kj}^{(s)} \ (j=k+1,k+2,\dots,n)$
5. $k := k+1$

注意，循环结束后$k-1$即为矩阵行秩，在我们的问题中，应该严格等于$n-1$。消元后的矩阵最后一行应全为0。

接着是回代过程（假设上一个条件满足），回代过程需要事先指定一个自由变量的值f（一般取1）。设置$s=n, \operatorname{rank} n = n-1$，对于$k=\operatorname{rank} n, \dots, 1$执行：

1. 如果$a_{ks}^{(n)}=0$，则$x_s=f$。
2. 否则$x_s=(-\sum_{i=s+1}^{n} a_{ki}^{(n)} x_i)/a_{ks}^{(n)}$。

至此我们解出x，接着进一步单位化即可得到需要的特征向量。

2. 源程序代码

本程序采用C++17标准编写，模块清晰，通用性强，效率优秀，使用CMake（version ≥ 3.16）组织代码，使用MSVC 8.1(amd64)或GCC 9.3.0均可编译通过。文件结构如下：

```
.
|  CMakeLists.txt
|  EigenUtils.cpp
|  EigenUtils.h
|  LinearEqUtil.cpp
|  LinearEqUtil.h
|  main.cpp
|  Matrix.cpp
|  Matrix.h
|  Vector.cpp
|  Vector.h
|  ZeroRangeGuard.h
```

对于向量和矩阵的实现，分别在`Vector`和`Matrix`类中；对于齐次线性方程组的求解，在类`LinearEqUtil`中；对特征值的求解放在了类`EigenUtil`中；最后我们实现了一个`ZeroRangeGuard`类，该类定义的对象可以直接对作用域中的零值判断进行控制。

代码内容见下页

3. 上机计算结果

以某次运行为例，得到结果：

```
Hessenberg Matrix:
-0.895  -0.099  -1.100  -0.767  0.171   -1.935  -0.084  0.913   -0.641  0.195
-2.348  2.372   1.828   0.327   0.208   2.089   0.185   -1.263  0.679   -0.467
-0.000  1.736   -1.165  -1.247  -0.630  -1.985  0.298   0.634   -0.131  0.304
-0.000  -0.000  -1.293  -1.126  1.191   -1.309  0.186   0.424   -0.102  0.194
0.000   0.000   0.000   1.578   0.817   0.446   -0.044  -0.467  0.294   -0.103
0.000   -0.000  -0.000  0.000   -0.773  -1.601  -0.291  -0.243  0.674   0.262
0.000   -0.000  0.000   0.000   0.000   -0.730  -0.008  0.971   -0.130  0.028
-0.000  -0.000  -0.000  0.000   0.000   0.000   0.795   -0.453  0.505   -0.121
0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.704   0.127   -0.371
-0.000  -0.000  0.000   0.000   0.000   0.000   -0.000  0.000   -0.492  0.408

k = 15
Matrix after QR Method:
3.390   -0.615  -1.097  -0.088  0.266   -0.641  -1.805  -0.076  -0.751  -0.743
-0.000  -2.686  2.311   0.045   -0.043  -1.306  -1.326  0.161   -0.563  -0.153
-0.000  -0.398  -1.987  0.638   0.022   -1.551  -0.983  0.306   1.465   1.694
-0.000  0.000   0.000   1.590   0.015   0.459   0.561   -0.007  -0.034  -0.135
0.000   -0.000  -0.000  -0.000  -1.493  0.056   0.096   -0.034  0.151   -0.054
-0.000  -0.000  -0.000  -0.000  -0.000  -1.090  0.684   -0.254  -0.359  -0.690
-0.000  0.000   0.000   0.000   -0.000  -0.032  -0.889  -0.114  0.279   0.065
-0.000  0.000   -0.000  -0.000  -0.000  -0.000  0.000   0.943   -0.121  -0.202
0.000   0.000   0.000   -0.000  0.000   0.000   -0.000  0.000   0.462   0.449
-0.000  0.000   -0.000  -0.000  0.000   0.000   0.000   0.000   0.172   0.236

eigenValue: .338961343882E+01   eigenVector: [-.104871999320E+00,
-.217676976320E+00, -.474694012241E+00, -.259383624651E+00, -.304665248521E+00,
-.259451746662E+00, .868664182734E-01, .405258126693E+00, .509628289643E+00,
.239514692166E+00]
||A*x-lambda*x||=.888178419700E-14

eigenValue: (-.233686593224E+01, -.893437921021E+00)

eigenValue: (-.233686593224E+01, .893437921021E+00)

eigenValue: -.149314708091E+01  eigenVector: [-.561340981698E+00,
.778192357458E+00, .143637166588E-01, -.277601903748E+00, .356807241900E-02,
-.254834165599E-02, -.220608987820E-01, -.117582711696E-01, -.131734984814E-01,
.350159577287E-01]
||A*x-lambda*x||=.666133814775E-15

eigenValue: .159031345881E+01   eigenVector: [.623768976129E-01, -.112312295279E-
01, -.252846032094E+00, -.130987581361E+00, -.381985138641E+00,
.815575288836E+00, -.123376782911E+00, -.677214519898E-01, .271944611155E+00,
.100282224999E+00]
||A*x-lambda*x||=.888178419700E-15

eigenValue: (-.989114346472E+00, -.108475863150E+00)

eigenValue: (-.989114346472E+00, .108475863150E+00)
```

```
eigenValue: .943287957277E+00   eigenVector: [.796197316849E-01, .454205684405E-01, -.182719542764E-01, -.479609167139E-01, -.349567427070E+00, .207214771156E+00, -.152312073430E+00, .820633710404E+00, -.355466329432E+00, .288659534097E-01]

||A*x-lambda*x||=.455191440096E-14

eigenValue: .495499092363E-01   eigenVector: [-.213767977959E+00, -.206773621699E+00, .386828983510E+00, -.311123946363E-01, -.380938960237E+00, -.125173726812E+00, .644715735839E+00, -.308201272967E+00, -.295976727012E+00, .437229510136E-01]
||A*x-lambda*x||=.360822483003E-15

eigenValue: .648948820211E+00   eigenVector: [.108434798577E+00, .713441259543E-01, .382501666947E+00, -.471003433310E-01, -.717803600565E+00, .181518546649E+00, -.226005938413E+00, .388381467696E+00, .289696424846E+00, .243327682952E-01]
||A*x-lambda*x||=.444089209850E-15
```

这里我们看到进行了15次迭代即得到了答案，而且$||Ax-\lambda x||$值也非常小（10^{-14}级别），意味着答案正确且精确度较好。

此外，在Release模式下，无输出时进行时间测量，基本上在400-500微秒之间。

4. 讨论分析

在本次程序设计中，主要面临的困难和思考有如下方面：

1. 对于特征值求法的问题，一开始选用的是方法1，实现并不困难，但是由于双步位移优化，使得原有QR方法的某些性质遭到破坏。后来采用了方法2。是否能实现方法1来求特征值，可以进行进一步思考。
2. 数值计算的精度把握比较困难。虽然统一最后要求$\varepsilon=10^{-12}$，但对中间结果的要求可能要高于这个精度，例如在QR分解和拟上三角化时对数值的判0；以及高斯消元时，对空行的判断。为了灵活对ε值进行细粒度控制，我们模仿了Python中的with语句，定义了`ZeroRangeGuard`类，该类定义的对象可以直接对作用域中的零值判断进行控制。这么做也提高了代码的可读性。
3. 在拟上三角化和QR分解中，出现的h变量，实际上有$h_r=\frac{1}{2}||u_r||^2$，由于h需要参与接下来的运算，值如果太小的化会破坏算法的数值稳定性。如果依赖对u向量中每个元素的判0，其实仍不够，因为这里存在一个小量的平方。这里需要直接对h进行零值判断。