数值分析上机实验

代数特征值问题

1 问题描述

本实验的目的是对如下的三对角矩阵求解特征值问题：

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \tag{1}$$

该矩阵是一个实对称矩阵，对这类矩阵我们有一些特殊的方式能够求出其所有特征值（而不只是少数几个最小或最大的特征值）。在本次实验中，我们分别使用 Jacobi 方法和 QR 方法求解特征值。

在 Jacobi 方法中，我们考虑如下的“旋转矩阵”（称为 Givens Rotation）：

$$J(j,k;\theta) = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & c & & & & s & & \\ & & & 1 & & & & & \\ & & & & \ddots & & & & \\ & & & & & 1 & & & \\ & & -s & & & & c & & \\ & & & & & & & 1 & \\ & & & & & & & & \ddots \\ & & & & & & & & & 1 \end{pmatrix} \tag{2}$$

在迭代过程中，我们按照如下规则确定矩阵 $J(k,l;\theta)$ 中角度 θ：

$$\theta = \begin{cases} \frac{1}{2}\arctan\frac{2a_{kl}}{a_{kk}-a_{ll}}, & a_{kk} \neq a_{ll} \\ \frac{\pi}{4}, & a_{kk} = a_{ll} \end{cases} \tag{3}$$

在每一步中，我们找出模最大的非对角元，记为 (k,l) 元，然后确定 $J(k,l;\theta)$，并对矩阵 A 做 Givens 变换：

$$A \to JAJ^{\dagger}, \tag{4}$$

对更新后的 A 重复以上过程，可以证明，当迭代步数足够大时，A 的非对角元的最大值会变得足够小，也就是，A 会收敛到一个对角矩阵。而酉（正交）相似变换不会改变特征值。因此，在实际操作时，我们可以设定一个阈值（在这次实验中默认采用 10^{-10}。当非对角元素的最大值小于这个阈值时，并输出此时 A 的对角元素。根据前面的讨论可知，在一定误差允许范围内此时 A 的对角元就是特征值。

QR 方法也是类似的迭代——收敛到对角矩阵的算法。其实现非常简单，直需先对 A 做 QR 分解：

$$A = QR, \tag{5}$$

然后，用交换位置相乘所得到的结果来更新 A，也就是：

$$A \to RQ, \tag{6}$$

事实上，这个过程相当于：

$$A \to RQ = Q^{\dagger}AQ, \tag{7}$$

可见，每一步相当于对 A 进行酉（相似）变换，此时特征值不变。而且，可以证明，如果 A 的特征值都按模分离，那么，由 QR 方法产生的矩阵序列最终将收敛到上三角矩阵。此时，其对角线上的元素就是 A 的特征值。在这个例子中，矩阵实对称的，所以实际上它收敛到对角矩阵。

在下一个部分，我们将讨论这两个算法的代码实现。

2 实验内容

这两个算法的代码实现都非常简单。

```
def jacobi_eigensolver(A, tol=1e-10):
"""
Jacobi method for computing eigenvalues and eigenvectors of a real symmetric matrix
    A.
"""
n = A.shape[0]
Ak = copy.copy(A)
while True:
    # Find maximum off-diagonal element
    B_temp = copy.copy(Ak)
    np.fill_diagonal(B_temp, 0)
    p, q = np.unravel_index(np.abs(B_temp[:n, :n]).argmax(), (n, n))
```

```
    if np.abs(B_temp[p, q]) < tol:
        break

    # Compute Jacobi rotation matrix
    if np.abs(Ak[q, q]-Ak[p, p])> 1e-10:
        theta = 0.5 * np.arctan(2 * Ak[p, q]/(Ak[q, q] - Ak[p, p]))
    else:
        theta = math.pi/4
    c, s = np.cos(theta), np.sin(theta)
    J = np.eye(n)
    J[p, p], J[q, q],J[p, q], J[q, p] = c, c, -s, s

    # Update Ak
    Ak = J @ Ak @ J.T

# Extract eigenvalues
evals = np.sort(np.diag(Ak))
return evals, None
```

由于本算法是迭代算法，因此在实现过程中，我们最好创建 A 的一个拷贝，以免破坏原来矩阵 A 的信息。在寻找其模最大的非对角元时，我们首先将矩阵 A 的所有对角元素都设定为 0，然后运用 `np.unravel_index` 和`.argmax` 方法找到最大元对应的行列指标。运用式 (3) 得到 θ 时，由于浮点计算的误差，当 $|a_{kk} - a_{ll}| < 10^{-10}$ 时。我们就取 $\theta = \frac{\pi}{4}$。

QR 算法的代码实现就更加简单：

```
def qr_eigensolver(A, tol=1e-10):
"""
QR method for computing eigenvalues and eigenvectors of a real symmetric matrix A.
"""
n = A.shape[0]
Ak = copy.copy(A)
while True:
    # Find maximum off-diagonal element
    B_temp = copy.copy(Ak)
    np.fill_diagonal(B_temp, 0)
    p, q = np.unravel_index(np.abs(B_temp[:n, :n]).argmax(), (n, n))
    if np.abs(B_temp[p, q]) < tol:
        break

    # Perform QR decomposition on Ak
    Q, R = np.linalg.qr(Ak)

    # Compute new matrix Ak
```

```
    Ak = np.dot(R, Q)

# Extract eigenvalues
evals = np.sort(np.diag(Ak))
return evals, None
```

在每次循环中，我们调用 `np.linalg.qr` 方法对矩阵进行 QR 分解，再反向相乘即可。

3 实验结果与讨论

我们选择 $n = 10, 35, 50$ 几个比较极端的阶数。与此同时，我们也使用 `numpy` 内置 `np.linalg.eigh` 求解特征值，作为以上编写两个函数的比较。代码如下:

```
import time
for n in (10, 35, 50):
    print(("*"*20 + " n = {} output " + "*"*20).format(n))
    # create the tridiagonal matrix
    A = np.zeros([n,n])+0.0
    for i in range(n):
        A[i,i] = 2
    for j in range(n-1):
        A[j+1,j] = -1
        A[j,j+1] = -1
    for func in [jacobi_eigensolver, qr_eigensolver, np.linalg.eigh]:
        print("="*20 + " " + func.__name__ + " " + "="*20)
        start_time = time.time()

        evals, evecs = func(A)

        end_time = time.time()
        total_time = end_time - start_time
        print("Total time: {} seconds".format(total_time))
        print("Eigenvalues: {}".format(evals))
```

在以上代码中，我们使用 `time.time` 来记录特征值求解的运行时长。结果如下：

```
******************** n = 10 output ********************
==================== jacobi_eigensolver ====================
Total time: 0.0029714107513427734 seconds
Eigenvalues: [0.08101405 0.31749293 0.69027853 1.16916997 1.71537032 2.28462968
 2.83083003 3.30972147 3.68250707 3.91898595]
==================== qr_eigensolver ====================
Total time: 0.0187685489654541 seconds
Eigenvalues: [0.08101405 0.31749293 0.69027853 1.16916997 1.71537032 2.28462968
```

```
 2.83083003 3.30972147 3.68250707 3.91898595]
==================== eigh ====================
Total time: 0.00038433074951171875 seconds
Eigenvalues: [0.08101405 0.31749293 0.69027853 1.16916997 1.71537032 2.28462968
 2.83083003 3.30972147 3.68250707 3.91898595]
******************** n = 35 output ********************
==================== jacobi_eigensolver ====================
Total time: 0.04143095016479492 seconds
Eigenvalues: [0.0076106  0.03038449 0.06814835 0.12061476 0.18738443 0.26794919
 0.36169591 0.46791111 0.58578644 0.71442478 0.85284713 1.
 1.15476348 1.31595971 1.48236191 1.65270364 1.82568851 2.
 2.17431149 2.34729636 2.51763809 2.68404029 2.84523652 3.
 3.14715287 3.28557522 3.41421356 3.53208889 3.63830409 3.73205081
 3.81261557 3.87938524 3.93185165 3.96961551 3.9923894 ]
==================== qr_eigensolver ====================
Total time: 0.573065996170044 seconds
Eigenvalues: [0.0076106  0.03038449 0.06814835 0.12061476 0.18738443 0.26794919
 0.36169591 0.46791111 0.58578644 0.71442478 0.85284713 1.
 1.15476348 1.31595971 1.48236191 1.65270364 1.82568851 2.
 2.17431149 2.34729636 2.51763809 2.68404029 2.84523652 3.
 3.14715287 3.28557522 3.41421356 3.53208889 3.63830409 3.73205081
 3.81261557 3.87938524 3.93185165 3.96961551 3.9923894 ]
==================== eigh ====================
Total time: 0.0015192031860351562 seconds
Eigenvalues: [0.0076106  0.03038449 0.06814835 0.12061476 0.18738443 0.26794919
 0.36169591 0.46791111 0.58578644 0.71442478 0.85284713 1.
 1.15476348 1.31595971 1.48236191 1.65270364 1.82568851 2.
 2.17431149 2.34729636 2.51763809 2.68404029 2.84523652 3.
 3.14715287 3.28557522 3.41421356 3.53208889 3.63830409 3.73205081
 3.81261557 3.87938524 3.93185165 3.96961551 3.9923894 ]
******************** n = 50 output ********************
==================== jacobi_eigensolver ====================
Total time: 0.4494516849517822 seconds
Eigenvalues: [3.79334253e-03 1.51589807e-02 3.40538006e-02 6.04061279e-02
 9.41159991e-02 1.35055541e-01 1.83069456e-01 2.37975611e-01
 2.99565729e-01 3.67606175e-01 4.41838851e-01 5.21982166e-01
 6.07732108e-01 6.98763400e-01 7.94730727e-01 8.95270054e-01
 1.00000000e+00 1.10852329e+00 1.22042825e+00 1.33529040e+00
 1.45267402e+00 1.57213383e+00 1.69321669e+00 1.81546328e+00
 1.93840988e+00 2.06159012e+00 2.18453672e+00 2.30678331e+00
 2.42786617e+00 2.54732598e+00 2.66470960e+00 2.77957175e+00
 2.89147671e+00 3.00000000e+00 3.10472995e+00 3.20526927e+00
```

```
 3.30123660e+00 3.39226789e+00 3.47801783e+00 3.55816115e+00
 3.63239382e+00 3.70043427e+00 3.76202439e+00 3.81693054e+00
 3.86494446e+00 3.90588400e+00 3.93959387e+00 3.96594620e+00
 3.98484102e+00 3.99620666e+00]
==================== qr_eigensolver ====================
Total time: 3.665611743927002 seconds
Eigenvalues: [3.79334253e-03 1.51589807e-02 3.40538006e-02 6.04061279e-02
 9.41159991e-02 1.35055541e-01 1.83069456e-01 2.37975611e-01
 2.99565729e-01 3.67606175e-01 4.41838851e-01 5.21982166e-01
 6.07732108e-01 6.98763400e-01 7.94730727e-01 8.95270054e-01
 1.00000000e+00 1.10852329e+00 1.22042825e+00 1.33529040e+00
 1.45267402e+00 1.57213383e+00 1.69321669e+00 1.81546328e+00
 1.93840988e+00 2.06159012e+00 2.18453672e+00 2.30678331e+00
 2.42786617e+00 2.54732598e+00 2.66470960e+00 2.77957175e+00
 2.89147671e+00 3.00000000e+00 3.10472995e+00 3.20526927e+00
 3.30123660e+00 3.39226789e+00 3.47801783e+00 3.55816115e+00
 3.63239382e+00 3.70043427e+00 3.76202439e+00 3.81693054e+00
 3.86494446e+00 3.90588400e+00 3.93959387e+00 3.96594620e+00
 3.98484102e+00 3.99620666e+00]
==================== eigh ====================
Total time: 0.0 seconds
Eigenvalues: [3.79334253e-03 1.51589807e-02 3.40538006e-02 6.04061279e-02
 9.41159991e-02 1.35055541e-01 1.83069456e-01 2.37975611e-01
 2.99565729e-01 3.67606175e-01 4.41838851e-01 5.21982166e-01
 6.07732108e-01 6.98763400e-01 7.94730727e-01 8.95270054e-01
 1.00000000e+00 1.10852329e+00 1.22042825e+00 1.33529040e+00
 1.45267402e+00 1.57213383e+00 1.69321669e+00 1.81546328e+00
 1.93840988e+00 2.06159012e+00 2.18453672e+00 2.30678331e+00
 2.42786617e+00 2.54732598e+00 2.66470960e+00 2.77957175e+00
 2.89147671e+00 3.00000000e+00 3.10472995e+00 3.20526927e+00
 3.30123660e+00 3.39226789e+00 3.47801783e+00 3.55816115e+00
 3.63239382e+00 3.70043427e+00 3.76202439e+00 3.81693054e+00
 3.86494446e+00 3.90588400e+00 3.93959387e+00 3.96594620e+00
 3.98484102e+00 3.99620666e+00]
```

由此可知，使用 Jacobi、QR 算法和 `NumPy` 内置的对角化方法得到的结果在 10^{-8} 都是一致的，因此，我们自己用代码实现的 Jacobi 和 QR 迭代算法都能以很好的精度求解代数特征值问题。

在耗时方面，我们将结果列于下表：

从表中看出，Jacobi 方法的耗时相比 QR 方法明显更少，在本次实验的三组数据中，Jacobi 方法比 QR 方法快 6 至 14 倍。

表 1: 耗时比较

	Jacobi 方法	QR 方法	`NumPy`
$n = 10$	0.0029714 s	0.0187685 s	0.0003843 s
$n = 35$	0.0414310 s	0.5730660 s	0.0015192 s
$n = 50$	0.4494517 s	3.6656117 s	0.0 s

另外也可以看出，我们这里仅仅是实现了具有功能的求解代数特征值的迭代算法，而并没有对一些算法细节进行优化，也没有采用任何加速手段。所以，我们的方法比 `Numpy` 慢几十倍到几百倍。因而可以想象，用这样简单的、没有经过优化的算法来真正求解实际遇到的那些大规模特征值问题，可能效果就会非常差，甚至无法求解。