数值分析实验五

计 63 陈晟祺 2016010981

2019 年 5 月 12 日

0.1 上机题 1

0.1.1 实验概述

本实验要求用幂法求矩阵模最大的特征值 λ_1 和其对应的特征向量 $\mathbf{x}_1$，并控制迭代前后误差小于 10^{-5}。

0.1.2 实验过程

幂法的实现比较简单，只需按照算法 5.1 描述的规则进行迭代即可。

```
In [1]: import numpy as np


        def power_method(A):
            n = A.shape[0]
            u = np.random.normal(0.0, 1.0, (n,1))
            l = 0
            # iteration for lambda
            while True:
                v = np.dot(A, u)
                new_l = v[np.argmax(np.abs(v))] # approximation of lambda_1
                u = v / new_l
                if np.abs(new_l - l) < 1e-5:
                    return new_l[0], u
                l = new_l
```

使用幂法分别对所给的两个矩阵进行迭代：

```
In [2]: A = np.array([[5, -4, 1], [-4, 6, -4], [1, -4, 7]])
        B = np.array([[25, -41, 10, -6], [-41, 68, -17, 10], [10, -17, 5, -3], [-6, 10, -3, 2]
        l_a, x_a = power_method(A)
        l_b, x_b = power_method(B)

        print('A has main eigenvalue {:.5f} with eigenvector {}'.format(l_a, list(map('{:.5f}'
        print('B has main eigenvalue {:.5f} with eigenvector {}'.format(l_b, list(map('{:.5f}'
```

```
A has main eigenvalue 12.25432 with eigenvector ['-0.67402', '1.00000', '-0.88956']
B has main eigenvalue 98.52170 with eigenvector ['-0.60397', '1.00000', '-0.25114', '0.14895']
```

使用 numpy 内置函数求值进行比较：

```
In [3]: def np_method(A, ref_eig_vec):
            w_a, v_a = np.linalg.eig(A)
            a_main_pos = np.argmax(np.abs(w_a)) # find main eigenvalue
            l_a, x_a = w_a[a_main_pos], v_a[:,a_main_pos]
            return l_a, x_a / x_a[np.where(ref_eig_vec == 1.)[0]] # do the same normalize as p

        l_a_n, x_a_n = np_method(A, x_a)
        l_b_n, x_b_n = np_method(B, x_b)

        print('A has main eigenvalue {:.5f} with eigenvector {}'.format(l_a_n, list(map('{:.5f
        print('B has main eigenvalue {:.5f} with eigenvector {}'.format(l_b_n, list(map('{:.5f
```

```
A has main eigenvalue 12.25432 with eigenvector ['-0.67402', '1.00000', '-0.88956']
B has main eigenvalue 98.52170 with eigenvector ['-0.60397', '1.00000', '-0.25114', '0.14895']
```

求得的主特征值小数点后五位都是一致的，并且当采用同样的归一化系数时，特征向量也是相同的。可见幂法的实现是正确的。

0.2 上机题 3

0.2.1 实验概述

本实验要求实现矩阵的 QR 分解，并使用基本的 QR 算法尝试计算给定矩阵的所有特征值，观察算法的收敛过程并给出解释。

0.2.2 实验过程

首先使用 Householder 旋转实现矩阵的 QR 分解：

```
In [4]: def householder(x):
            if (x[0] >= 0):
                sign = 1
            else:
                sign = -1
            sigma = sign * np.linalg.norm(x, ord=2)
            if np.abs(sigma - x[0]) < 1e-10:
                return None
            h = x.copy()
            h[0] += sigma
            return h


        def QR(A):
            n = A.shape[0]
            R = A.copy()
            Q = np.identity(n)
            for i in range(n - 1):
                # get sub-matrix
                R_1 = R[i:,i:]
                # householder vector v and w
                v = householder(R_1[:,0])
                if v is None: # go to next submatrix
                    continue
                w = v / np.linalg.norm(v, ord=2)
                # caculate H and transform Q
                H = np.identity(n)
                H[i:,i:] = np.identity(n - i) - 2 * np.dot(w, w.transpose())
                Q = np.matmul(Q, H)
                # use v to calculate transformed R
                beta = np.dot(v.transpose(), v)[0,0]
                for j in range(n - i):
                    gamma = np.dot(v.transpose(), R_1[:,j])[0,0]
                    R_1[:,j] -= 2 * gamma / beta * v
            return Q, R
```

使用 numpy 内置的 QR 分解可以测试算法的正确性：

```
In [5]: A = np.matrix([[1., 2], [3, 4]])
        Q, R = QR(A)
        Q_n, R_n = np.linalg.qr(A)
        print('Total Error: {:.3e}, Q Error: {:.3e}, R Error: {:.3e}'.format(np.max(np.abs(Q *
```

```
Total Error: 9.992e-16, Q Error: 2.220e-16, R Error: 8.882e-16
```

接下来可以实现基本的 QR 算法求特征值。在判定拟对角阵和求解拟对角阵的特征值时，都需要对对角块为 2 * 2 矩阵的情况进行特殊处理。

```
In [6]: eps = 1e-5

        # check if the matrix is quasi-diagonal
        def check_quasi_diag(A):
            n = A.shape[0]
            cond = A < eps
            i = 0
            while i < n:
                cond[i, i] = True
                if i < n - 1 and cond[i + 1,i] == False:
                    # 2d-matrix
                    cond[i + 1, i] = True
                    i += 2
                else:
                    i += 1
            for i in range(n):
                for j in range(i):
                    if not cond[i, j]:
                        return False
            return True

        # find eigenvalues by each block
        def derive_eigen(A):
            n = A.shape[0]
            eigen = np.zeros(n,dtype=np.complex128)
            i = 0
            while i < n:
```

```
            if i < n - 1 and A[i + 1, i] > eps:
                # 2d-matrix
                eigen[i : i + 2] = np.linalg.eig(A[i:i+2,i:i+2])[0]
                i += 2
            else:
                # 1d-matrix
                eigen[i] = A[i, i]
                i += 1
        return np.round(eigen, decimals=4)

    # basic QR algorithm
    def QR_eigen(A):
        n = A.shape[0]
        step = 0
        while not check_quasi_diag(A):
            # iterate
            Q, R = np.linalg.qr(A)
            A_new = R * Q
            step += 1
            # iteration converged
            if np.max(np.abs(A_new - A)) < 1e-8:
                print('QR algorithm converged to non-quasi-diagonal matrix after {} steps,
                return None
            A = A_new
        print('QR algorithm found eigenvalues of A after {} steps'.format(step))
        return derive_eigen(A)
```

对题中给出的矩阵使用 QR 算法：

```
In [7]: A = np.matrix([[0.5, 0.5, 0.5, 0.5], [0.5, 0.5, -0.5 , -0.5], [0.5, -0.5, 0.5, -0.5],
        l_A = QR_eigen(A)
```

```
QR algorithm converged to non-quasi-diagonal matrix after 1 steps, failed to find eigenvalues
```

算法在进行了一步迭代后就失败了，结合代码中的判定条件，可知一步迭代后 A 没有发生变化，如下所示：

```
In [8]: Q, R = QR(A)
        R * Q
```

```
Out[8]: matrix([[ 0.5,  0.5,  0.5,  0.5],
                [ 0.5,  0.5, -0.5, -0.5],
                [ 0.5, -0.5,  0.5, -0.5],
                [ 0.5, -0.5, -0.5,  0.5]])
```

这是由于 **A** 事实上本身是一个正交矩阵，因此 QR 分解得到的 **R** 是恒等的（或者只差一个符号），故无法使用基本的 QR 算法进行迭代寻找特征值。

0.3 上机题 4

0.3.1 实验概述

本题要求用带原点位移的 QR 算法计算第三题中矩阵的特征值，并观察收敛结果，与第三题进行比较。

0.3.2 实验过程

实现带原点位移的 QR 算法，并打印每次迭代过程。当每次迭代出一个特征值后，都检查矩阵的（拟）对角性；如果成立则立刻停止迭代。

```
In [9]: def print_matrix(A):
            n = A.shape[0]
            for i in range(n):
                print('\t'.join(map(lambda x: '{: .4f}'.format(x), A[i,:].tolist()[0])))


        def QR_shift_eigen(A, n=None):
            if n is None: # initial calling
                n = A.shape[0]
                A = A.copy()
                print('Original matrix:')
                print_matrix(A)
            if n <= 1 or check_quasi_diag(A):
                print('Matrix is already quasi-diagonal, end iteration')
                return
            # find the last diagonal element of size n
            count = 0
            while np.abs(A[n - 1,n - 2]) > eps or np.abs(A[n - 1,n - 1]) < eps:
                old_A = A.copy()
                s = A[n - 1, n - 1]
                Q, R = QR(A[:n,:n] - s * np.identity(n))
                A[:n,:n] = R * Q + s * np.identity(n)
                count += 1
                print('After iteration {}:'.format(count))
                print_matrix(A)
                if np.max(np.abs(A - old_A)) < eps:
                    raise Exception('Iteration converged but no more eigenvalue is found')
            print('Shifted QR took {} steps to find eigenvalue {:.4f} of A'.format(count, A[n
            QR_shift_eigen(A, n - 1)
            return derive_eigen(A)
```

```
In [10]: a_l_shift = QR_shift_eigen(A)
```

```
Original matrix:
 0.5000          0.5000          0.5000          0.5000
 0.5000          0.5000         -0.5000         -0.5000
 0.5000         -0.5000          0.5000         -0.5000
 0.5000         -0.5000         -0.5000          0.5000
After iteration 1:
-0.5000          0.6708         -0.4392         -0.3273
 0.6708          0.7000          0.1964          0.1464
-0.4392          0.1964          0.8714         -0.0958
-0.3273          0.1464         -0.0958          0.9286
After iteration 2:
-0.9991         -0.0349          0.0202         -0.0143
-0.0349          0.9994          0.0004         -0.0002
 0.0202          0.0004          0.9998          0.0001
-0.0143         -0.0002          0.0001          0.9999
After iteration 3:
-1.0000         -0.0000          0.0000         -0.0000
-0.0000          1.0000          0.0000         -0.0000
 0.0000          0.0000          1.0000          0.0000
-0.0000         -0.0000          0.0000          1.0000
Shifted QR took 3 steps to find eigenvalue 1.0000 of A
Matrix is already quasi-diagonal, end iteration
```

可以看到，带原点位移的 QR 算法解决了简单 QR 算法处理正交矩阵时的问题（因为位移破坏

了正交性），仅在三个迭代后就得到了第一个特征值。并且此时矩阵刚好已成为对角矩阵，故所有特征值都已经找到：

```
In [11]: a_l_shift
```

```
Out[11]: array([-1.+0.j,  1.+0.j,  1.+0.j,  1.+0.j])
```

我们还可以使用更多的正交矩阵进行测试，比如下列矩阵有一对共轭复特征值：

```
In [12]: A = np.matrix([[0, -0.8, -0.6], [0.8, -0.36, 0.48], [0.6, 0.48, -0.64]])
         QR_shift_eigen(A)
```

```
Original matrix:
 0.0000         -0.8000         -0.6000
 0.8000         -0.3600          0.4800
 0.6000          0.4800         -0.6400
After iteration 1:
 0.0000         -0.9751         -0.2218
 0.9751         -0.0492          0.2162
 0.2218          0.2162         -0.9508
After iteration 2:
 0.0000         -1.0000         -0.0081
 1.0000         -0.0001          0.0081
 0.0081          0.0081         -0.9999
After iteration 3:
-0.0000         -1.0000         -0.0000
 1.0000         -0.0000          0.0000
 0.0000          0.0000         -1.0000
Shifted QR took 3 steps to find eigenvalue -1.0000 of A
Matrix is already quasi-diagonal, end iteration
```

```
Out[12]: array([-0.+1.j, -0.-1.j, -1.+0.j])
```

可以看到带原点位移的 QR 算法也顺利地将其迭代成为拟对角矩阵，并且找到了所有的特征值。

但是书中给出的单位移策略也并非通用的，例如对于下列矩阵，这一策略就无法求得特征值：

```
In [13]: A = np.matrix([[0., 0, 0, 1], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0]])
         try:
             QR_shift_eigen(A)
         except Exception as e:
             print(e)
```

```
Original matrix:
 0.0000         0.0000         0.0000         1.0000
 0.0000         0.0000         1.0000         0.0000
 0.0000         1.0000         0.0000         0.0000
 1.0000         0.0000         0.0000         0.0000
After iteration 1:
-0.0000         0.0000         0.0000         1.0000
 0.0000        -0.0000         1.0000         0.0000
 0.0000         1.0000        -0.0000         0.0000
 1.0000         0.0000         0.0000        -0.0000
Iteration converged but no more eigenvalue is found
```

0.4 实验结论

本实验中，我实现了求矩阵特征值的三种方法。幂法较为简单，可以快速求出绝对值最大的特征值。简单 QR 算法和带简单原点位移策略的 QR 算法都能求所有特征值，且后者的适用范围更广。事实上，如果使用更佳的策略（如双位移），带原点位移的 QR 算法总是能够收敛到拟三角阵，从而能方便地求出特征值。