Zhejiang University

Numerical Analysis

Numerical Analysis Project Report

信息与电子工程学院

3220103612 章杨

2023 年 11 月 14 日

目录

1 Chapter1:Introduction

1.1 背景介绍

在实际工程和科学应用中，我们经常会遇到大规模的数据集，其中数据往往以矩阵的形式表示。同时，这些矩阵往往是“稀疏”的，这意味着它们具有大量的零元素，而非零元素只占据了整个矩阵的一小部分。在不同的领域我们都会遇到稀疏矩阵，比如社交网络分析、图像处理、自然语言处理、有限元分析、物理学模拟等。这些领域中的数据通常具有高维度，但由于各种原因，矩阵中的数据元素只有极小一部分是非零的。因此，稀疏矩阵的特性和处理方法变得至关重要。

在这样的情况下，如何高效地计算和理解稀疏矩阵的特征值成为一个具有挑战性的问题。

在本次报告中，我将使用特定的数值计算方法来解决该问题，包括 Power Method、QR 算法和 Arnoldi 迭代算法以及 ERAM 算法。我将比较不同算法在普通矩阵和高维稀疏矩阵上的性能，这将有助于深入理解稀疏矩阵的重要性以及特征值计算方法的应用。

1.2 任务介绍

- 1. 生成随机矩阵
  - （1）生成一个 10x10 维度的随机矩阵。
  - （2）生成一个 10000x10000 维度且密度为 0.001 的随机稀疏矩阵，并统计矩阵中非零元素数量。
  - （3）利用库函数计算（1）和（2）中矩阵的特征值。
- 2. 给出 Power Method 的伪代码并用代码实现，能够输出绝对值最大的特征值。
  - （1）利用 Power Method 计算题目 1（1）中矩阵的绝对值最大的特征值。
  - （2）利用 Power Method 计算题目 1（2）中稀疏矩阵的绝对值最大的特征值。
- 3. 给出 QR 算法的伪代码并用代码实现，并能够实现输出前 k 个绝对值最大的特征值，其中 k 为自定义参数。
  - （1）利用 QR 算法计算题目 1（1）中矩阵的前 4 个绝对值最大的特征值。
  - （2）利用 QR 算法计算题目 1（2）中稀疏矩阵的前 5 个绝对值最大的特征值。
- 4. 用代码实现 Arnoldi 迭代算法，并能够实现输出前 k 个绝对值最大的特征值，其中 k 为自定义参数。
  - （1）利用 Arnoldi 迭代算法计算题目 1（1）中矩阵的前 6 个绝对值最大的特征值。
  - （2）利用 Arnoldi 迭代算法计算题目 1（2）中稀疏矩阵的前 7 个绝对值最大的特征值。
- 5.（bonus，非必做）给出 ERAM 或 IRAM 算法的伪代码并用代码实现，并能够实现输出前 k 个绝对值最大的特征值，其中 k 为自定义参数。
  - （1）利用 ERAM 或 IRAM 算法计算题目 1（2）中稀疏矩阵的前 8 个绝对值最大的特征值。
  - （2）利用 ERAM 或 IRAM 算法计算题目 1（2）中稀疏矩阵的前 20 个绝对值最大的特征值。

2 Chapter2:Algorithm Specification

2.1 Power Method

2.1.1 幂法介绍

幂法是计算矩阵的按模最大的特征值和相应特征向量的一种向量迭代法。

基本思想如下:

设 n 阶矩阵 A 的特征值和特征向量为

$$\lambda_1, \lambda_2, ...\lambda_n$$

$$v_1, v_2, ...v_n$$

$v_1, v_2, ...v_n$ 线性无关，且

$$|\lambda_1|, |\lambda_2|, ...|\lambda_n|$$

称 λ_1 为 A 的主特征值。任取非零向量 x_0, 构造序列x_k=$A^k x0$，由

$$x_0 = a_1v_1 + a_2v_2 + ... + a_nv_n$$

得到

$$x_k = a_1\lambda_1^k v_1 + a_2\lambda_2^k v_2 + ... + a_n\lambda_n^k v_n = \lambda_1^k(a_1v_1 + \epsilon_k)$$

$$\epsilon_k = a_2(\frac{\lambda_2}{\lambda_1})^k + ... + a_n(\frac{\lambda_n}{\lambda_1})^k$$

$$\lim_{k\to+\infty} \frac{1}{\lambda_1^k}x^k = a_1v_1$$

就是 λ_1 对应的特征向量。

2.1.2 伪代码

实际计算的时候，为了避免计算过程中出现绝对值过大或者绝对值过小的数参加运算，在每一步迭代的时候，用模最大分量进行归一化。

$$\begin{cases} y_k = Ax_{k-1} \\ m_k = max(y_k) \\ x_k = y_k/m_k \end{cases}$$

当 k 充分大时，有

$$\begin{cases} \lambda_1 = m_k \\ v_1 = x_k \end{cases}$$

由此可以用程序进行幂法的操作。

下面给出伪代码:

Algorithm 1 幂法

function POWERMETHOD(A, ϵ)
  $m, n \leftarrow$ shape of A
  $uk \leftarrow \text{ones}(n, 1)$ ▷ 初始向量
  $flag \leftarrow 1$ ▷ 判断条件
  $val \leftarrow 0$ ▷ 特征值
  $pre_val \leftarrow 0$ ▷ 上一次的特征值
  $n \leftarrow 0$
  while flag **do**
    $n \leftarrow n + 1$
    $vk \leftarrow A \cdot uk$ ▷ 乘上 A 矩阵
    $val \leftarrow \max(|x| \text{ for } x \text{ in } vk)$ ▷ 最终收敛的时候，最大特征值将会是 vk 中的最大元素
    $uk \leftarrow \frac{vk}{val}$ ▷ 除掉最大值用于下一次计算
    if $|val - pre\_val| < \epsilon$ **then** ▷ 判断是否符合条件
      flag $\leftarrow 0$
    end if
    $pre\_val \leftarrow val$ ▷ 把当前计算的特征值赋值给 pre_val 用于下一次计算
  end while
  return val
end function

2.1.3 代码实现

```python
def powermethod(A,eps):#幂法
    m,n=A.shape
    uk=np.ones((n,1))#初始向量
    flag=1#判断条件
    val=0#特征值
    pre_val=0#上一次的特征值
    n=0
    while flag:
        n=n+1
        vk =np.dot(A,uk)#乘上A矩阵
        val = max(abs(x) for x in vk)#最终收敛的时候，最大特征值将会是vk中的最大元素
        uk = vk / val#除掉最大值用于下一次计算
        if (np.abs(val - pre_val) < eps):#判断是否符合条件
            flag = 0
        pre_val = val#把当前计算的特征值赋值给pre_val用于下一次计算
    return val
```

2.2 QR 算法

2.2.1 Householder

1.Householder 实现 QR 分解

Householder 变换可以将一个向量的某些函数置 0，同时保持范数不变。例如，可以将

$$x = [x_1, x_2, ...x_n]^T$$

转化为

$$e = [1, 0, ..., 0]^T$$

采取的变化矩阵为

$$H = I - 2\frac{vv^T}{||v||_2}$$

其中，$v = x + (sgn)||x||_2 e_1$

由此，我们首先将第一列的元素集中到第一行，然后关注右下角的 (n-1)*(n-1) 的矩阵，再将它的第一列的元素集中到它的第一行，最终获得上三角矩阵，即 R 矩阵。

2. 代码实现

```
def Householder(A):
    (m, n) = A.shape
    R = np.copy(A)#R为和A同样的矩阵
    for line in range(n - 1):#line表示现在处理第i列
        l = n - line#(n-1)-line+1,表示要构造的规模是l*l的
        a = np.array(R[line:, line]).reshape(l, 1)#取R矩阵的第line列，其中的数据从第
            line行开始
        a_norm = np.linalg.norm(a)#计算范数
        e = np.zeros((l, 1))#构造一个l维的向量
        e[0] = 1#把它的第一个元素赋值为1
        w = a - np.dot(a_norm, e)#进行householder变换的工作
        if(np.linalg.norm(w)==0):#检查范数是否为0（为了应对稀疏矩阵的全0列）
            continue
        w = w / np.linalg.norm(w)#正交化
        H_sizeofl = np.identity(l) - 2.0 * np.dot(w, w.T)#计算l*l的矩阵H
        if (line == 0):#如果是第一次
            H = H_sizeofl#直接赋值给H
        else:#如果不是
            H = np.block([#进行一个分块矩阵的操作，左上角为单位阵，其他为0
                [np.identity(line), np.zeros((line, l))],
                [np.zeros((l, line)), H_sizeofl]
            ])#分块矩阵
        # 最后计算Q和R
        if (line == 0):
            Q = H
        else:
            Q = np.dot(Q, H)
        R = np.dot(H, R)
    return Q,R
```

2.2.2 QR 算法

1. 基本思想

QR 算法的核心思路是利用矩阵的 QR 分解，迭代构造一系列矩阵并最终收敛为上三角矩阵，从而计算特征值。具体而言，矩阵 A 可以通过 QR 分解得到 A=QR ，其中 Q 为正交矩阵，R 为上三角阵。

通过构建矩阵序列 $A_{k+1} = R_k Q_k = Q_k^T A_k Q_k$，可以得到矩阵序列 $\{A_k\}$ ，其中每一个矩阵 A_k 都与矩阵 A 相似，即具有相同的特征值。假设特征值各不相同且降序排列。因此，当 k 趋于无穷时，A_k 收敛为上三角矩阵，此时的对角线元素即为特征值。

2. 伪代码

Algorithm 2 QR 方法

function QRmethod(A, ϵ)
  $n \leftarrow 0$
  flag $\leftarrow 1$
  $A_{\text{pre}} \leftarrow A$
  while flag **do**
    $Q, R \leftarrow$ Householder(A_{pre}) ▷ 对 A_{pre} 进行 QR 分解
    $A_{\text{next}} \leftarrow R \cdot Q$ ▷ 得到的 A 矩阵为 $R \cdot Q$
    if $\max(|\text{diag}(A_{\text{next}} - A_{\text{pre}})|) < \epsilon$ **then** ▷ 判断误差
      flag $\leftarrow 0$
    end if
    $A_{\text{pre}} \leftarrow A_{\text{next}}$ ▷ 用于迭代
    $n \leftarrow n + 1$
  end while
  return diag(A_{next}) ▷ 返回 A_{next} 的对角线元素作为特征值
end function

2. 代码实现

```python
def QRmethod(A,eps):#QR方法的思路
    n=0
    flag=1
    A_pre=A
    while flag:
        Q,R=Householder(A_pre)#对A_pre进行QR分解
        A_next=np.dot(R,Q)#得到的A矩阵为R*Q
        # print(max(x for x in np.diag(np.abs(A_next - A_pre))))
        if (max(x for x in np.diag(np.abs(A_next-A_pre)))<eps):#判断误差
            flag=0
        A_pre=A_next#用于迭代
        n+=1
    return np.diag(A_next)
```

2.3 Arnoldi 迭代算法

2.3.1 算法介绍

Arnoldi 迭代算法的核心思路是将矩阵投影到低维 Krylov 子空间，从而将矩阵约化为上 Hessenberg 矩阵，从而计算其特征值。通过低维 Krylov 子空间，Arnoldi 迭代并不直接利用矩阵本身，而是利用矩阵向量乘积，从而减小处理高维矩阵时的复杂度。

具体而言，对于给定的矩阵 A 和向量，Krylov 序列为向量集合

$$\{b, Ab, A^2b, ...\}$$

M 维 Krylov 子空间定义为长度为 m 的 Krylov 序列张成的空间（可以把序列中的向量理解为基向量）

Arnoldi 迭代期望将矩阵分解为 $A = Q^T HQ$ ，其中 H 为上 Hessenberg 矩阵，Q 为正交矩阵, 然后用对 H 进行特征值求解。

值得注意的是，对于高维矩阵我们往往只关注其一部分的特征值，比如只考虑前 k«n 列。此时，可以将矩阵分解简化为矩阵分解为 $AQ_k = Q_{k+1}H_k$

2.3.2 伪代码

Algorithm 3 Arnoldi 算法

function ARNOLDI(A, k)
- $m, n \leftarrow$ shape of A ▷ 获取规模
- $Q \leftarrow \text{zeros}(m, k+1)$ ▷ 构造一个 m 行，k+1 列的矩阵，用于存放 Krylov 空间的向量
- $H \leftarrow \text{zeros}(k+1, k)$ ▷ 构造一个 k+1 行，k 列的矩阵，因为我们只关注前 k 行 k 列
- $q \leftarrow \text{random.rand}(m)$ ▷ 随机初始化一个向量
- $q \leftarrow \frac{q}{\|q\|}$ ▷ 正交化
- $Q[:, 0] \leftarrow q$ ▷ Q 的第一列放置初始向量
- **for** $j \leftarrow 0$ to $k-1$ **do** ▷ 进行 k 次，填充完整个 Krylov 空间
  - $v \leftarrow A \cdot Q[:, j]$ ▷ 计算 $v = A \cdot q_m$，得到 v 为一个 m*1 的列向量
  - **for** $i \leftarrow 0$ to j **do**
    - $H[i, j] \leftarrow Q[:, i] \cdot v$ ▷ 计算 H 的第 i 行第 j 列元素
    - $v \leftarrow v - H[i, j] \cdot Q[:, i]$ ▷ 根据公式更新 v
  - **end for**
  - $H[j+1, j] \leftarrow \|v\|$ ▷ 计算 H 的第 j+1 行第 j 列元素
  - **if** $H[j+1, j] = 0$ **then**
    - **break**
  - **end if**
  - $Q[:, j+1] \leftarrow \frac{v}{H[j+1, j]}$ ▷ 更新 Q 的第 j+1 列
- **end for**
- **return** Q, H ▷ 返回计算得到的 Q 和 H 矩阵

end function

2.3.3 代码实现

```python
def arnoldi(A, k):
    m, n = A.shape#获取规模
    Q = np.zeros((m, k+1))#构造一个m行，k+1列的矩阵，用于存放krylov空间的向量
    H = np.zeros((k+1, k))#构造一个k+1行，k列的矩阵，因为我们只关注前k行k列
    q = np.random.rand(m)#随机初始化一个向量
    q = q / np.linalg.norm(q)#正交化
    Q[:, 0] = q#Q的第一列放置初始向量

    for j in range(k):#进行k次，填充完整个krylov空间
        v = A @ Q[:, j]#v=Aq_m ,得到v为一个m*1的列向量
        #计算q_{m+1}
        for i in range(j+1):
            H[i, j] = np.dot(Q[:, i], v)
            v = v - H[i, j] * Q[:, i]#根据公式
        H[j+1, j] = np.linalg.norm(v)
        if H[j+1, j] == 0:
            break
        Q[:, j+1] = v / H[j+1, j]

    return Q[:, :k], H[:k, :k]#最后取前k行k列
```

2.4 ERAM 算法优化

2.4.1 算法介绍

Arnoldi 算法能够高效地从一个 m×m 的低维子空间估计出矩阵的一部分特征值，但是当矩阵的特征值存在分簇现象，子空间的维度 m 会不可避免地变大。此时，Arnoldi 迭代算法会占用较大的存储空间。为解决该问题，一种新的方法被用于改进 Arnoldi 迭代算法，被称为 explicitly restarted Arnoldi method（ERAM）。

该方法依旧使用较小维度的 Krylov 子空间，当近似的特征值不满足要求时，算法会在一个新的 Krylov 子空间上重启。新的 Krylov 子空间初始向量 b ，由上一轮近似的特征值线性组合形成。

在本次算法设计中，重点在于重启 Krylov 子空间并确定初始向量 b，b 为一个 10000*1 的向量，但是我们只有前 k 个特征值，为了作线性组合，我取任意两个特征值进行相加，这样就有 C_k^2 个值，然后归一化作为初始向量。可以在后面的测试结果中看到这个线性组合方式确实优化了 Arnoldi 算法。

2.4.2 伪代码

Algorithm 4 ERAM 算法

function ERAM$_s parse$(A, K)　M, N ← SHAPE OF A　▷ 获取规模

FLAG ← 1

$Q, H \leftarrow$ ARNOLDI(A, k)　▷ 进行一次 ARNOLDI 迭代后得到 H

$evals_p re, \leftarrow$ EIG(H)　▷ 得到一个差不多的特征值

while FLAG $= 1$ **do**

$q \leftarrow$ ONES(m)

for $i \leftarrow 0$ TO $m - 1$ **do**

$t \leftarrow$ RANDOM.RANDINT$(0, k - 1)$　▷ 随机化两个坐标

$s \leftarrow$ RANDOM.RANDINT$(0, k - 1)$

$q[i] \leftarrow evals_p re[t] + evals_p re[s]$　▷ 让 Q 向量等于随机两个特征值的和

end for

$Q \leftarrow$ ZEROS$(m, k + 1)$

$H \leftarrow$ ZEROS$(k + 1, k)$

$q \leftarrow \frac{q}{\|q\|}$

$Q[:, 0] \leftarrow q$

for $j \leftarrow 0$ TO $k - 1$ **do**

$v \leftarrow A \cdot Q[:, j]$

for $i \leftarrow 0$ TO j **do**

$H[i, j] \leftarrow Q[:, i] \cdot v$

$v \leftarrow v - H[i, j] \cdot Q[:, i]$

end for

$H[j + 1, j] \leftarrow \|v\|$

if $H[j + 1, j] = 0$ **then**

break

end if

$Q[:, j + 1] \leftarrow \frac{v}{H[j+1,j]}$

end for

$evals \leftarrow eig(H)$　▷ 计算新的特征值

if MAX$(|evals - evals_p re|) < \epsilon$ **then**　▷ 判断是否收敛

FLAG ← 0

end if

$evals_p re \leftarrow evals$　▷ 更新特征值

end while

return $evals$　▷ 返回计算得到的特征值

$=0$

2.4.3 代码实现

```
def ERAM_sparse(A,k):#ERAM算法，因为这里只处理稀疏矩阵
    m, n = A.shape#获取规模
    flag=1
    Q,H=arnoldi(A,k)#进行一次aroldi迭代后得到H
    evals_pre,_=scipy.linalg.eig(H)#得到一个差不多的特征值
    while(flag==1):
        # print(evals_pre)
        # print(evals)
        q = np.ones(m)
        for i in range(m):
            t=random.randint(0,k-1)#随机化两个坐标
            s=random.randint(0,k-1)
            q[i] =evals_pre[t]+evals_pre[s]#然后让这个q向量等于随机两个特征值的和
        #接下来的操作同arnoldi算法
        Q = np.zeros((m, k + 1))
        H = np.zeros((k + 1, k))
        q = q / np.linalg.norm(q)
        Q[:, 0] = q
        for j in range(k):
            v = A @ Q[:, j]
            for i in range(j + 1):
                H[i, j] = np.dot(Q[:, i], v)
                v = v - H[i, j] * Q[:, i]
            H[j + 1, j] = np.linalg.norm(v)
            if H[j + 1, j] == 0:
                break
            Q[:, j + 1] = v / H[j + 1, j]
        H=H[:k, :k]#更新H

        evals, evecs = scipy.linalg.eig(H)#更新新的特征值
        maxd = max(x for x in abs(evals - evals_pre))#计算向量差的无穷范数
        # print(maxd)
        if (maxd < 1):#误差给定1
            flag = 0
    return evals
```

3 Chapter3:Testing Results

3.1 测试流程

在整个程序中，将先由程序生成一个 10*10 的对称阵，然后生成一个密度为 0.001 的 10000*10000 的稀疏矩阵，它的非零个数显然为 100000，为了后续的计算，我们把这个稀疏矩阵进行对称变化（采取加上转置除以二的方法）。

在这个方法中，稀疏矩阵的密度将发生变化，但这对后续的矩阵处理没有过大的影响，对于测试算法性能也没有过大的影响，为了避免复特征值的出现采取了这种方法。在测试过程中，我也试着将矩阵的密度调整为 0.0005 然后做对称变换之后的密度接近 0.001，但是还是有一定的误差，所以就不过多追究。

测试程序中，输入 1-5 的值将进行不同的操作，返回对应题号的结果。值得一提的是，上交的版本中矩阵的规模为 10000*10000，输入 3（即 QR 分解）将返回时间过长不进行 QR 分解的提示。若要验证 QR 分解的准确性需要将规模与密度的参数改变后进行。在下面的测试结果里，我也放置了我的测试结果。

3.2 测试交互程序

输入 1-5 查看不同题目的结果，输入 q 退出程序。

```
def main():
    while(1):
        user_input = input("Which problem do you want to check:")
        if user_input == '1':
            problem1()
        elif user_input == '2':
            problem2()
        elif user_input == '3':
            problem3()
        elif user_input == '4':
            problem4()
        elif user_input == '5':
            problem5()
        elif user_input == 'q':
            print("Thanks.")
            break
        else:
            print("Invalid input, please try again.")
```

3.3 测试结果

一、先进行完整的一套流程，即 T1(2) 矩阵的规模为 10000*1000 的。

1.T1: 第一题中，我们用库函数求解两个矩阵的特征值，由于第二个矩阵采取稀疏矩阵的 sparse.linalg.eigs 函数，一般情况下返回 6 个特征值，若扩大获取的特征值数量，获得的序列也只有前 6 个是有序的，所以在此只取 6 个特征值。

```
(3):1矩阵的特征值为
[ 5.40752489+0.j -1.06108709+0.j -0.85768572+0.j  0.96310202+0.j
 -0.4956872 +0.j  0.72674889+0.j  0.41782177+0.j  0.23523182+0.j
 -0.24545517+0.j -0.03023356+0.j]
----------------------------------------------------------------------
    2矩阵的特征值为
[ 5.34890854+0.j  2.71782662+0.j  2.70630998+0.j  2.70015053+0.j
 -2.70937231+0.j -2.70343878+0.j]
```

图 1: T1

2.T2: 第二题是采用幂法求矩阵的绝对值最大的特征值。在此，我使用第一题得到的特征值序列来给出幂法的一个检验。

```
Which problem do you want to check :2
problem2:
(1):1矩阵中绝对值最大的特征值的绝对值：
[5.40752614]
----------------------------------------------------------------------
(2):2矩阵中绝对值最大的特征值的绝对值：
[[5.34894872]]
----------------------------------------------------------------------
使用problem1中得到的特征值序列得到的最大值(作为检验):
检验1: 5.40752489301102
检验2: 5.3489085392229745
```

图 2: T2

3.T3: 第三题是用 QR 分解求解矩阵前 k 个特征值。可以跟第一题获得的所有特征值序列进行比较，会发现准确度很高。

```
Which problem do you want to check :3
Problem3
(1)第一个矩阵的前4个绝对值最大的特征值为：
[5.40752489301102, 1.0610870897209403, 0.9631020220970071, 0.8576857208474987]
----------------------------------------------------------------------
规模过大，使用QR分解将持续很长时间，故不进行
```

图 3: T3

4.T4: 第四题是采取 Arnoldi 迭代算法获取前 k 个特征值，此处我拿库函数序列进行检验，可以看到对小规模的矩阵 arnoldi 算法的精度较高，但对大规模的稀疏矩阵，随着获取的特征值个数的增大，误差变大。

```
Which problem do you want to check :4
Problem4
(1)第一个矩阵的前6个绝对值最大的特征值为
[5.407524893011029, 1.061087089720941, 0.9631020220970146, 0.8576857208474966, 0.7267488944203093, 0.4956871985563918]
用库函数得到的前6个绝对值最大的特征值为
[5.40752489301102, 1.061087089720942, 0.9631020220970147, 0.8576857208474943, 0.7267488944203077, 0.49568719855639204]
----------------------------------------------------------------------
(2)第二个矩阵的前7个绝对值最大的特征值为
[5.348908307347877, 2.4510532977454593, 2.3793492911526295, 1.7524986172730093, 1.534964729631428, 0.7256941448965143, 0.4299464869381696]
用库函数得到的前7个绝对值最大的特征值为
[5.348908539222979, 2.717826620220503, 2.709372314169863, 2.7063099750444723, 2.703438783163568, 2.7001505316057575]
```

图 4: T4

5.T5: 第五题是用 ERAM 算法对 Arnoldi 算法进行一个改进。可以看到得到的特征值误差被减小了，并且随着 k 值的增大，距离真正的特征值序列的误差减小。(警告原因为强制转化特征值时有可能丢失虚部，但是本次操作使用的矩阵是实对称阵，没有复特征值，所以没有影响)

```
Which problem do you want to check :5
Problem5
(1)第二个矩阵的前8个绝对值最大的特征值为
D:\Study\大二秋冬\数值分析方法\HW\3220103612_章杨_project\3220103612_章杨_project_code\project.py:128: ComplexWarning: Casting complex values to real di
  q[i] =evals_pre[t]+evals_pre[s]#然后让这个q向量等于随机两个特征值的和
[5.348908196729003, 2.53022940185606, 2.4277409746789322, 1.962395914604727, 1.77056726029793, 1.1688359225843814, 0.8348443354802051, 0.17459667611
----------------------------------------------------------------------
(2)第二个矩阵的前20个绝对值最大的特征值为
[5.348908539222976, 2.688189890985703, 2.6876249189906516, 2.5841228892927277, 2.5811861573992236, 2.4154895154009717, 2.404519977045627, 2.20067888
```

图 5: T5

二、为了验证 QR 分解对于稀疏矩阵的正确性，采用 100*100 的矩阵以减小计算时间，修改密度为 0.01，精度 eps 为 1e-3。

在此，我拿第四题（Arnoldi 算法）的结果以及库函数的结果作为检验，可以发现对稀疏矩阵的的精确度也比较高。误差差距在于，在 10*10 的矩阵中，我的误差为 1e-10，这里为了减少计算时间改成了 1e-3.

```
Which problem do you want to check :3
Problem3
(1)第一个矩阵的前4个绝对值最大的特征值为:
[4.57647448050302, 1.3202697441965845, 0.8857571711426848, 0.7973480987255123]
----------------------------------------------------------------------
(2)第二个矩阵的前5个绝对值最大的特征值为:
[1.0109482320899585, 1.0106316997526072, 0.8844505302292256, 0.8676019290352256, 0.865669971564896]
Which problem do you want to check :4
Problem4
(1)第一个矩阵的前6个绝对值最大的特征值为
[4.576474480503022, 1.3202697441966318, 0.8857571711426796, 0.7973480995681396, 0.7543293374210164, 0.4712943922715281]
用库函数得到的前6个绝对值最大的特征值为
[4.576474480503025, 1.3202697441966267, 0.88575717114268, 0.7973480995681409, 0.7543293374210166, 0.4712943922715285]
----------------------------------------------------------------------
(2)第二个矩阵的前7个绝对值最大的特征值为
[1.025561961375873, 0.9553210539842384, 0.8117371686504938, 0.6470690578366828, 0.5182252651785875, 0.23249126237431197, 0.043684569280471784]
用库函数得到的前7个绝对值最大的特征值为
[1.0291686391771183, 1.0288521068398366, 0.8844525360215525, 0.8726564956834556, 0.8707265440054575, 0.8281841374231851]
```

图 6: T6

4 Chapter4:Analysis and Comments

4.1 幂法

1. 时间复杂度：幂法的时间复杂度为 $O(mn^2)$, 其中 m 是迭代的次数，n 是矩阵的维度。在每次迭代中，需要进行一次矩阵向量乘法和一次向量归一化操作，时间复杂度为 $O(n^2)$。迭代次数 m 通常较小，因此可以将其视为常数。因此，幂法的总时间复杂度为 $O(n^2)$。

2. 空间复杂度：幂法的空间复杂度是 O(n)，其中 n 是矩阵的维度。在每次迭代中，需要存储当前向量和上一次迭代的向量，它们的维度都是 n。因此，幂法的总空间复杂度为 O(n)。

3. 性能：对比而言，幂法求主特征值的速度较快，但仅限于最大的特征值，有所局限。老师上课提到过对矩阵进行变换并由新的矩阵寻找第二大特征值，原本想按照这种方式做一个测试函数的，但最后时间有限就没有实践。

4.2 QR 分解

1. 时间复杂度：对于一个 n×n 的矩阵，Householder 进行 QR 分解的时间复杂度为 $O(n^3)$。这是因为 QR 分解涉及到多次的 Householder 变换，每次变换的时间复杂度为 $O(n^2)$，而进行 n 次变换。对于用 QR 分解寻找特征值，在迭代时还有小计算量的迭代次数 m，以及 n 维序列的加减，复杂度都没有到达 $O(n^3)$, 所以总时间复杂度为 $O(n^3)$。

2. 空间复杂度：Householder 分解的空间复杂度为 $O(n^2)$。这是因为在 Householder 变换的过程中，需要存储一些临时向量和矩阵（如矩阵 H 与矩阵 R）。

3. 性能：对比而言，用 QR 分解可以求得所有特征值，而且根据上述的测试结果，精确度较高。然而，我的 householder 分解很难支持稀疏矩阵（因为存在全 0 列向量的情况），以及对于高维的稀疏矩阵，运算时间会过长。

4.3 Arnoldi 算法

1. 时间复杂度：Arnoldi 算法的时间复杂度为 $O(n^2)$。这是因为在每次迭代中，需要进行一次矩阵向量乘法和一些向量运算，它们的时间复杂度都与矩阵的维度 n 相关。而进行 k 次迭代，因此总的时间复杂度为 $O(kn^2)$, 由于 k 远小于 n，所以获得最终的时间复杂度为 $O(n^2)$。

2. 空间复杂度：Arnoldi 算法的空间复杂度为 $O(n)$。这是因为在每一次迭代中创造了一个 n*(k+1) 维的矩阵与一个 (k+1)*k 维度的矩阵。因为 k 远小于 n 所以空间复杂度为 $O(n)$。

3. 性能：Arnoldi 算法的时间复杂度小了很多，对小规模的矩阵 arnoldi 算法的精度较高，但对大规模的稀疏矩阵，随着获取的特征值个数的增大，误差变大。这可能是因为在大规模的矩阵中只关注前 k*k，舍弃了部分特征值，并且初始向量的选取也会影响整个算法的精度。

4.4 ERAM 算法

1. 时间复杂度：ERAM 算法的时间复杂度为 $O(n^2)$。ERAM 相当于把 arnoldi 算法的空间重复了很多遍，但是迭代次数显然不会多于 n，所以时间复杂度仍为 $O(n^2)$

2. 空间复杂度：ERaM 算法继承了 Arnoldi 算法的空间复杂度，为 $O(n)$。因为重启的时候并没有创造新的空间，只是把原来的空间置零了。

3. 性能：相比而言，我认为改进是成功的，在面对大规模稀疏矩阵的时候，Arnoldi 算法的误差被大大降低了，通过不断地重启 Krylov 空间以及改进初始向量，最终在测试结果中可以看到显著的进步。而且在处理更多特征值的情况更为出色。

5 Declaration

I hereby declare that all the work done in this project titled ”Numerical Analysis Project Report” is of my independent effort.