《数值分析方法》课程

稀疏矩阵特征值

项目报告

Xm H

2023.11.19

目录

一、项目简介

本项目使用matlab代码，实现了Power Method算法，QR 算法，Arnoldi 迭代算法，IRAM 算法，计算矩阵的特征值。

项目具体内容：

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。

二、项目简介

2.1 矩阵生成模块

Ⅰ算法描述：

1. 生成一个 10x10 的复数随机矩阵。实部和虚部都是随机生成的。
2. 生成一个 100x100 维度且密度为 0.1 的随机稀疏矩阵。首先生成实数部分的稀疏矩阵，然后找到矩阵中所有非零元素的位置，为这些非零元素加上一个随机的纯虚数，生成虚部，最后重新构造矩阵生成复数稀疏矩阵。
3. 统计稀疏矩阵中非零元素的数量。
4. 计算 10x10 随机矩阵的特征值。
5. 计算 100x100 稀疏矩阵的特征值，这里计算的是最大的 100 个特征值。

Ⅱ代码分析：

```
%1. 生成随机矩阵
% (1)生成一个 10x10 的随机矩阵
matrix_10x10 = rand(10)+1i*rand(10);

% (2)生成一个 100x100 维度且密度为 0.1 的随机稀疏矩阵
%生成实数部分
sparse_matrix = sprand(100, 100, 0.1);

% 找到矩阵所有非零元素的位置
[row, col, val] = find(sparse_matrix);

% 为所有非零元素加上一个随机的纯虚数，生成虚部
val = val + 1i * rand(size(val));
% 重新构造矩阵生成复数矩阵
sparse_matrix = sparse(row, col, val, size(sparse_matrix, 1), size(sparse_matrix, 2));

% 统计稀疏矩阵中非零元素的数量
non_zero_elements = nnz(sparse_matrix);

% (3) 计算 10x10 随机矩阵的特征值
eigenvalues_10x10 = eig(matrix_10x10);
% 计算 100x100 稀疏矩阵的特征值
eigenvalues_sparse = eigs(sparse_matrix,100);
```

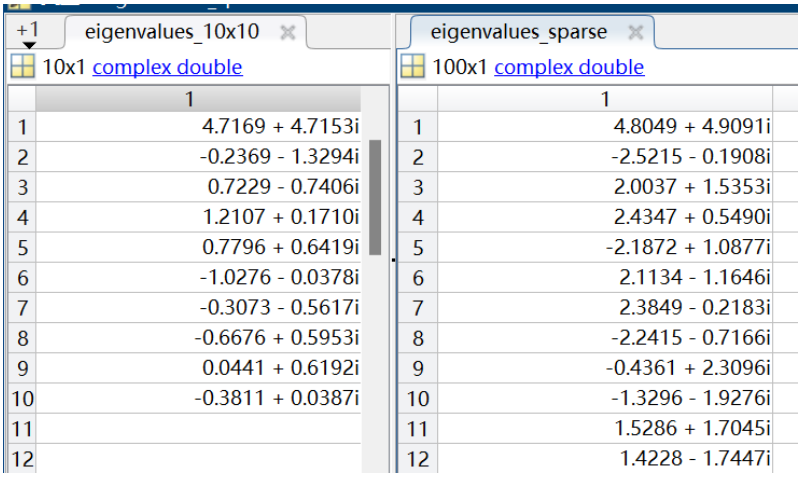
图1 矩阵生成部分源码

由于矩阵的生成和统计非零元素较为简单，未涉及复杂算法，直接使用matlab库函数实现；

1. 使用matlab的rand函数生成复矩阵；
2. 使用sparse函数随生成一个实稀疏矩阵，然后找到非零元素，加上虚部生成稀疏复矩阵；
3. 用nnz函数直接统计非零元素；
4. 用eig和eigs函数求随机生成的复矩阵的特征值，作为标准值，用于检验之后算法结果的正确性。

III计算结果分析：

```
matrix_10x10 =

  列 1 至 6

   0.3108 + 0.6343i   0.6479 + 0.4072i   0.4294 + 0.9323i   0.6514 + 0.9834i   0.5695 + 0.5261i   0.1855 + 0.0423i
   0.6346 + 0.6784i   0.7737 + 0.5807i   0.8306 + 0.9244i   0.5497 + 0.3368i   0.0041 + 0.0394i   0.6919 + 0.8130i
   0.2798 + 0.2496i   0.9214 + 0.4591i   0.3709 + 0.5776i   0.0658 + 0.5794i   0.8456 + 0.4797i   0.2417 + 0.5483i
   0.3500 + 0.0950i   0.2787 + 0.3604i   0.9383 + 0.4941i   0.4701 + 0.4344i   0.9384 + 0.9060i   0.3355 + 0.7076i
   0.9236 + 0.5933i   0.5081 + 0.9774i   0.0604 + 0.8489i   0.2447 + 0.9215i   0.1328 + 0.5829i   0.3526 + 0.5767i
   0.8403 + 0.4016i   0.4516 + 0.2415i   0.5061 + 0.7249i   0.3064 + 0.5882i   0.5491 + 0.4000i   0.5483 + 0.5150i
   0.5639 + 0.2699i   0.0468 + 0.0924i   0.5138 + 0.4663i   0.4033 + 0.0134i   0.3319 + 0.0357i   0.2678 + 0.1491i
   0.3163 + 0.6621i   0.0164 + 0.4169i   0.1579 + 0.6849i   0.2375 + 0.7584i   0.1583 + 0.6579i   0.4213 + 0.5408i
   0.4015 + 0.4658i   0.2672 + 0.6772i   0.4645 + 0.3395i   0.1469 + 0.0040i   0.8447 + 0.4146i   0.4916 + 0.5356i
   0.9454 + 0.1245i   0.9977 + 0.4247i   0.0484 + 0.3653i   0.3832 + 0.6050i   0.0089 + 0.6483i   0.4251 + 0.8441i

  列 7 至 10

   0.1500 + 0.3421i   0.4331 + 0.2193i   0.6648 + 0.2715i   0.4028 + 0.9517i
   0.5235 + 0.3389i   0.1534 + 0.4847i   0.3687 + 0.9980i   0.3120 + 0.7572i
   0.2382 + 0.6027i   0.0299 + 0.3835i   0.4390 + 0.2264i   0.9160 + 0.8104i
   0.3646 + 0.8500i   0.8303 + 0.3745i   0.0774 + 0.5897i   0.7865 + 0.3744i
   0.9223 + 0.1981i   0.4600 + 0.7909i   0.2298 + 0.6903i   0.6672 + 0.0557i
   0.3950 + 0.8913i   0.1318 + 0.3614i   0.5146 + 0.7753i   0.0173 + 0.1274i
   0.8818 + 0.3417i   0.9625 + 0.1497i   0.5295 + 0.7984i   0.5704 + 0.0123i
   0.9578 + 0.6654i   0.1664 + 0.0971i   0.8932 + 0.5894i   0.7629 + 0.4993i
   0.9516 + 0.7964i   0.9057 + 0.0217i   0.3769 + 0.0529i   0.4609 + 0.2063i
   0.0676 + 0.1096i   0.3815 + 0.0384i   0.9403 + 0.1007i   0.8222 + 0.2951i

>> non_zero_elements

non_zero_elements =

   954
```



eigenvalues_10x10

10x1 complex double

	1
1	4.7169 + 4.7153i
2	-0.2369 - 1.3294i
3	0.7229 - 0.7406i
4	1.2107 + 0.1710i
5	0.7796 + 0.6419i
6	-1.0276 - 0.0378i
7	-0.3073 - 0.5617i
8	-0.6676 + 0.5953i
9	0.0441 + 0.6192i
10	-0.3811 + 0.0387i
11	
12	

eigenvalues_sparse

100x1 complex double

	1
1	4.8049 + 4.9091i
2	-2.5215 - 0.1908i
3	2.0037 + 1.5353i
4	2.4347 + 0.5490i
5	-2.1872 + 1.0877i
6	2.1134 - 1.1646i
7	2.3849 - 0.2183i
8	-2.2415 - 0.7166i
9	-0.4361 + 2.3096i
10	-1.3296 - 1.9276i
11	1.5286 + 1.7045i
12	1.4228 - 1.7447i

图 2 矩阵生成和特征值计算结果

如图为随机矩阵和特征值结果，稀疏矩阵生成结果过大，图略。

IV计算性能分析：

1. 生成随机矩阵：这部分的时间复杂度和空间复杂度都是 O(n^2)，其中 n 是矩阵的维度。因为生成一个 n 维的随机矩阵需要 n^2 次操作，同时也需要 n^2 的空间来存储这个矩阵。

2. 生成稀疏矩阵：这部分的时间复杂度和空间复杂度都是 O(m)，其中 m 是矩阵中非零元素的数量。因为生成一个稀疏矩阵只需要对非零元素进行操作，同时也只需要存储非零元素。

3. 计算特征值：对于一般的 n 维矩阵，eig 函数计算特征值的时间复杂度是 O(n^3)，空间复杂度是 O(n^2)。但是对于稀疏矩阵，由于大部分元素都是零，所以可以使用特殊的算法来提高效率。在 MATLAB 中，eigs 函数就是专门用来计算稀疏矩阵特征值的，它的时间复杂度和空间复杂度都是 O(m)，其中 m 是矩阵中非零元素的数量。

2.2 Power Method Algorithm

Ⅰ算法描述：

```
1.  函数 Power_Method(A, max_iter, TOL) 返回 lambda
2.    随机初始化一个非零向量 v
3.    对于 i 从 1 到 max_iter
4.      计算 w = A * v
5.      计算 w 的 2-范数，得到 norm_w
6.      计算 new_v = w / norm_w
7.      如果 new_v 和 v 的差的 2-范数小于 TOL，则跳出循环
8.      更新 v = new_v
9.      计算 lambda = v' * A * v
10.   返回 lambda
```

伪代码如上，这个函数实现的是幂法（Power Method），用于求解矩阵A的最大特征值。在每次迭代中，它首先计算矩阵A和向量v的乘积，然后将结果归一化，得到新的向量new_v。如果new_v和v的差的2-范数小于给定的容差TOL，那么就停止迭代，或直到达到最大迭代次数。最后，它计算最大特征值lambda，然后返回。

Ⅱ代码分析：

```
function lambda = Power_Method(A, max_iter,TOL)
    % 初始化一个非零向量 v
    v = rand(size(A, 1), 1);
    for i = 1:max_iter
        w = A * v;                          % 计算 w 的 2-范数
        new_v = w / norm(w);                % sum = 0;
                                            %     for i = 1:length(w)
                                            % sum = sum + w(i)^2;
                                            %     end
                                            % norm_w = sqrt(sum);

        if norm(new_v - v) < TOL            %计算new_v与v的差的2-范数
        break;                              % d = new_v - v;
        end                                 % sum = 0;
       v=new_v;                             % for i = 1:length(d)
       lambda = v' * A * v;                 %     sum = sum + d(i)^2;
    end                                     % end
end                                         % norm_d = sqrt(sum);
```

图 3幂法源代码

输入：矩阵 A，最大迭代次数 max_iter，容差 TOL

输出：最大特征值 lambda

计算范数的代码如注释所见，由于matlab已内置向量范数函数norm，故不自行封装norm函数，之后直接调用matlab的norm函数计算范数，也加快程序速度。

Ⅲ计算结果分析：

```
>> Main

Power_Method_find_max_eigenvalue_10x10 =

   4.7169 + 4.7153i


Power_Method_find_max_eigenvalue_sparse =

   4.8049 + 4.9091i
```

图 4Power Method计算结果

如图，对比图2中的特征值，可见计算结果正确，且实际运行过程中计算速度较快。

Ⅳ计算性能分析：

- 时间复杂度：这段代码的时间复杂度主要取决于两个部分，一是矩阵和向量的乘法 A * v，二是向量的 2-范数计算 norm(new_v - v)。对于一个 n 维的矩阵，矩阵和向量的乘法的时间复杂度是 O(n^2)，向量的 2-范数计算的时间复杂度是 O(n)。因此，这段代码的总时间复杂度是 O(max_iter * n^2)，其中 max_iter 是迭代次数。

- 空间复杂度：这段代码的空间复杂度主要取决于矩阵 A 和向量 v。对于一个 n 维的矩阵，矩阵 A 的空间复杂度是 O(n^2)，向量 v 的空间复杂度是 O(n)。因此，这段代码的总空间复杂度是 O(n^2)。

2.3 QR Algorithm

Ⅰ算法描述：

```
函数 QR_Algorithm(A, k, max_iter) 返回 eigenvalues
    对于 m 从 1 到 max_iter
        获取矩阵 A 的阶数 n
        初始化 Q 和 R 为 n x n 的零矩阵
        对于 i 从 1 到 n
            设置 v = A 的第 i 列
            对于 j 从 1 到 i-1
                计算 R(j, i) = Q 的第 j 列 和 v 的点积
                更新 v = v - R(j, i) * Q 的第 j 列
            计算 R(i, i) = v 的 2-范数
            计算 Q 的第 i 列 = v / R(i, i)
        更新 A = R * Q
    提取 A 的对角线元素，得到 eigenvalues_temp
    将 eigenvalues_temp 降序排列
    提取 eigenvalues_temp 的前 k 个元素，得到 eigenvalues
    返回 eigenvalues
```

这个函数实现的是QR算法，用于求解矩阵A的特征值。在每次迭代中，它首先使用Gram-Schmidt过程进行QR分解，然后更新矩阵A。最后，它提取A的对角线元素作为特征值，将特征值降序排列，然后返回前k个特征值。QR分解的伪代码如下：

```
函数 QR_Decomposition(A) 返回 Q, R
    获取矩阵 A 的阶数 n
    初始化 Q 和 R 为 n x n 的零矩阵
    对于 i 从 1 到 n                    %通过 Gram-Schmidt 过程进行 QR 分解
        设置 v = A 的第 i 列
        对于 j 从 1 到 i-1
            计算 R(j, i) = Q 的第 j 列 和 v 的点积
            更新 v = v - R(j, i) * Q 的第 j 列
        计算 R(i, i) = v 的 2-范数
        计算 Q 的第 i 列 = v / R(i, i)
    返回 Q, R
```

Gram-Schmidt过程是一种将一组向量正交化的方法：1.从输入的向量集合中取出第一个向量，将其归一化，得到第一个正交向量；2.取出下一个向量，减去其在已有正交向量上的投影，得到一个与已有正交向量都正交的新向量，然后将新向量归一化，得到下一个正交向量；重复第2步，直到所有的向量都被处理。
QR分解中，先使用Gram-Schmidt过程对矩阵的列进行正交化，得到正交矩阵Q。然后，计算上三角矩阵R，使得原矩阵A等于Q和R的乘积。

Ⅱ代码分析：

```
function eigenvalues = QR_Algorithm(A, k, max_iter)
    for m = 1:max_iter
        n = size(A, 1);          % 获取矩阵 A 的阶数
        Q = zeros(n);            % 初始化 Q
        R = zeros(n);            % 初始化 R
        % 用 Gram-Schmidt 过程进行 QR 分解
        for i = 1:n
        v = A(:, i);
            for j = 1:i-1
                R(j, i) = Q(:, j)' * v;
                v = v - R(j, i) * Q(:, j);
            end
        R(i, i) = norm(v);
        Q(:, i) = v / R(i, i);
        end
            A = R * Q;           % 更新 A
    end
eigenvalues_temp = diag(A);% 提取对角线上的特征值
eigenvalues_temp = sort((eigenvalues_temp), 'descend');% 降序排列特征值
eigenvalues = eigenvalues_temp(1:k);% 取前k个特征值
end
```

图表 5QR算法源代码

输入：矩阵 A，要求的特征值个数 k，最大迭代次数 max_iter

输出：矩阵 A 的前 k 个特征值

该代码即通过QR分解和迭代的matlab代码实现，将矩阵A通过迭代收敛为上三角矩阵，最后提取对角线上的元素即特征值。

III计算结果分析：

```
QR_Algorithm_find_kth_max_eigenvalue_10x10 =

   4.7169 + 4.7153i
  -0.2369 - 1.3294i
   1.2107 + 0.1710i
   0.7229 - 0.7406i


QR_Algorithm_find_kth_max_eigenvalue_sparse =

   4.8049 + 4.9091i
  -2.5215 - 0.1908i
   2.0037 + 1.5353i
   2.4347 + 0.5490i
  -2.1872 + 1.0877i
```

图 6QR分解算法计算结果

如图，对比图2中的特征值，可见计算结果正确。在实际运行过程中对100x100的矩阵，QR方法速度较慢，符合O(n^3)的时间复杂度

IV计算性能分析：

- 时间复杂度：这段代码的时间复杂度主要取决于两个部分，一是 QR 分解，二是矩阵的乘法。对于一个 n 维的矩阵，QR 分解的时间复杂度是 O(n^3)，矩阵的乘法的时间复杂度也是 O(n^3)。因此，这段代码的总时间复杂度是 O(max_iter * n^3)，其中 max_iter 是迭代次数。

- 空间复杂度：这段代码的空间复杂度主要取决于矩阵 A、Q 和 R。对于一个 n 维的矩阵，矩阵 A、Q 和 R 的空间复杂度都是 O(n^2)。因此，这段代码的总空间复杂度是 O(n^2)

2.4 Arnoldi Algorithm

Ⅰ算法描述：

```
1.  函数 Arnoldi_Algorithm(A, k, dimension) 返回 eigenvalues, Q
2.  获取矩阵 A 的阶数 n
3.  初始化 Q 为 n x dimension 的零矩阵
4.  初始化 H 为 dimension x dimension 的零矩阵
5.  生成一个随机的 n 维向量，存储在 Q 的第一列，并进行归一化
6.  %step7-15 进行 Arnoldi 迭代
7.  对于 j 从 1 到 dimension
8.      计算 v = A * Q 的第 j 列
9.      对于 i 从 1 到 j
10.         计算 H(i, j) = Q 的第 i 列 和 v 的点积
11.         更新 v = v - H(i, j) * Q 的第 i 列
12.     如果 j 小于 dimension
13.         计算 H(j+1, j) = v 的 2-范数
14.         如果 H(j+1, j) 等于 0，则跳出循环
15.         更新 Q 的第 j+1 列 = v / H(j+1, j)
16. 调用 QR_Algorithm 函数求解 H 的特征值，得到 eigenvalues
17. 返回 eigenvalues, Q
```

该算法通过Arnoldi迭代来构造一个正交矩阵Q和一个Hessenberg矩阵H，使得AQ=QH。然后，我们可以通过求解Hessenberg矩阵H的特征值来得到矩阵A的特征值。
在每次迭代中，它首先计算矩阵A和向量Q的第j列的乘积，然后通过Gram-Schmidt过程进行正交化，得到新的向量v。当j小于dimension，如果v的2-范数为0，说明已经找到A的所有特征值，跳出循环；否则，更新Q的第j+1列。最后，它调用QR_Algorithm函数求解Hessenberg矩阵H的特征值，然后返回特征值和Q矩阵。

Ⅱ代码分析：

```
function [eigenvalues, Q] = Arnoldi_Algorithm(A, k, dimension)
    n = size(A, 1);
    Q = zeros(n, dimension);                % 存储Arnoldi迭代过程中的正交阵
    H = zeros(dimension, dimension);        % 存储Hessenberg矩阵
    Q(:, 1) = randn(n, 1);                  % 随机生成初始向量
    Q(:, 1) = Q(:, 1) / norm(Q(:, 1));      % 归一化

    for j = 1:dimension                     % Arnoldi迭代
        v = A * Q(:, j);
        for i = 1:j
            H(i, j) = Q(:, i)' * v;
            v = v - H(i, j) * Q(:, i);      % 正交化
        end
        if j < dimension
            H(j+1, j) = norm(v);
            if H(j+1, j) == 0               % 正交化后向量为0，说明已经找到A的所有特征值
                break;
            end
            Q(:, j+1) = v / H(j+1, j);
        end
    end
    eigenvalues = QR_Algorithm(H, k,10000); %调用QR算法求解特征值
end
```

图表 7Arnoldi算法源码

函数输入：矩阵A，迭代次数k，子空间维度dimension；
函数输出：k个最大特征值，特征向量矩阵Q；
在通过Arnoldi迭代得到dimension维的H矩阵之后，调用之前实现的QR算法来求解H的特征值

Ⅲ计算结果分析：

```
Arnoldi_Algorithm_find_kth_max_eigenvalue_10x10 =

   4.7169 + 4.7153i
  -0.2369 - 1.3294i
   1.2107 + 0.1710i
   0.7229 - 0.7406i
  -1.0276 - 0.0378i
   0.7796 + 0.6419i


Arnoldi_Algorithm_find_kth_max_eigenvalue_sparse =

   4.8049 + 4.9091i
  -2.5215 - 0.1908i
   2.0037 + 1.5353i
   2.4347 + 0.5490i
  -2.1872 + 1.0877i
   2.1134 - 1.1646i
   2.3849 - 0.2183i
```

图 8 Arnoldi算法计算结果

如图，对比图2中的特征值，可见计算结果正确，精度足够。在实际运行过程中对100x100的矩阵，计算速度比QR分解快许多，符合 O(n^2)的时间复杂度

Ⅳ计算性能分析：

- 时间复杂度：Arnoldi 算法的时间复杂度主要取决于两个部分，一是矩阵与向量的乘法，二是 Gram-Schmidt 正交化过程。对于一个 n 阶的矩阵，矩阵与向量的乘法的时间复杂度是 O(n^2)，Gram-Schmidt 正交化过程的时间复杂度也是 O(n^2)。因此，Arnoldi 算法的总时间复杂度是 O(dimension * n^2)，其中 dimension 是我们希望得到的特征值的数量。

- 空间复杂度：Arnoldi 算法的空间复杂度主要取决于矩阵 A、Q 和 H。对于一个 n 阶的矩阵，矩阵 A 的空间复杂度是 O(n^2)，矩阵 Q 的空间复杂度是 O(n * dimension)，矩阵 H 的空间复杂度是 O(dimension^2)。因此，Arnoldi 算法的总空间复杂度是 O(n^2 + n * dimension + dimension^2)。

2.5 IRAM Algorithm

Ⅰ算法描述：

```
function IRAM_Algorithm(A, k, p, v1)
    执行 k 步 Arnoldi 迭代
    如果找到所有的特征值
        计算 Hk 的特征值和特征向量
        计算 A 的特征向量
        返回特征值和特征向量
    计算 Arnoldi 迭代的总步数 m
    对于每一次隐式重启的迭代
        执行 p 步 Arnoldi 迭代
        如果找到所有的特征值
            计算 Hm 的特征值和特征向量
            计算 A 的特征向量
            返回特征值和特征向量
        检测是否发生特征值 deflation
        如果发生特征值 deflation
            提取 Vm 和 Hm 的部分列和子矩阵
            如果 H1 是奇异的
                返回空的特征值和特征向量
            计算 H1 的特征值和特征向量
            计算修正后的 A 矩阵
            对修正后的 A 矩阵重新进行 IRAM 迭代
            合并特征值和特征向量
            返回特征值和特征向量
        计算 Hm 的特征值和特征向量
        对特征值进行排序
        提取剩余的特征值
        初始化 et 向量
        对于每一个选定的特征值
            用 QR shift 更新矩阵 Hm 和 Vm
            更新 et 向量
        计算新的残差 fk
        更新 Hk 和 Vk
```

这段代码实现的是隐式重启Arnoldi方法（IRAM），用于计算矩阵的特征值和特征向量。以下是其算法描述： 1.执行k步Arnoldi迭代。如果在这个过程中找到所有的特征值，那么就计算Hk的特征值和特征向量，然后计算A的特征向量，并返回结果。 2.计算Arnoldi迭代的总步数m。 3.进行隐式重启的迭代。在每一次迭代中，首先执行p步Arnoldi迭代。如果在这个过程中找到所有的特征值，那么就计算Hm的特征值和特征向量，然后计算A的特征向量，并返回结果。 4.检测是否发生特征值deflation。 如果发生了特征值deflation，那么就提取Vm和Hm的部分列和子矩阵，然后计算H1的特征值和特征向量，对修正后的A矩阵重新进行IRAM迭代，最后合并特征值和特征向量，并返回结果。 5.如果没有发生特征值deflation，那么就计算Hm的特征值和特征向量，对特征值进行排序，提取剩余的特征值，然后用QR shift更新矩阵Hm和Vm，计算新的残差fk，最后更新Hk和Vk。 6.重复步骤3-5，直到满足停止条件。

Ⅱ代码分析：

```
%进行k步Arnoldi迭代
[ Hk, Vk, fk, flag ] = IRAM_Arnoldi_iter( A, k, v1);

if flag                         %如果提前找到所有的特征值
    [y, theta] = eig(Hk);       %计算HK的特征值矩阵和特征向量矩阵
    theta = diag(theta);        %将特征值矩阵转换为向量
    x = Vk * y;                 %计算A的特征向量矩阵
    return
end

m = k + p;                      %计算Arnoldi迭代的总步数

for iter = 1:1000               %进行隐式重启的迭代

    %进行p步Arnoldi迭代
    [ Hm, Vm, fm, flag ] = Arnoldi_Implicitly_Restart(A, Hk, Vk, fk, p);

    if flag
        [y, theta] = eig(Hm);
        theta = diag(theta);
        x = Vm * y;
        return
    end
```

先调用IRAM_Arnoldi_iter函数进行初始的迭代

之后进入隐式重启的迭代，调用Arnoldi_Implicitly_Restart函数进行重启迭代

```
% 检测是否发生特征值deflation
subdiag = diag(Hm, -1);                                 %提取Hm的次对角线元素
[subdiag_minval, subdiag_min] = min(abs(subdiag));      %找到次对角线元素的最小值和最小值的索引

if subdiag_minval <= 0.001                              %如果最小值小于0.001，则认为发生了特征值deflation
    V1 = Vm(:, 1:subdiag_min);                          %提取Vm的前subdiag_min列
    H1 = Hm(1:subdiag_min, 1:subdiag_min);              %提取Hm的左上角subdiag_min*subdiag_min子矩阵

    if norm(H1) < 0.01                                  %如果H1的范数小于0.01，则认为H1是奇异的，无法计算特征值
        theta = [];
        x = [];
        return
    end

    [y, theta] = eig(H1);                               %如果H1非奇异，则计算H1的特征值和特征向量
    theta = diag(theta);
    x = V1 * y;


    A_rec  = (eye(length(A)) - V1 * V1') * A;           %计算修正后的A矩阵
    %对修正后的A矩阵重新进行IRAM迭代
    [ theta_rec, x_rec ] = IRAM_Algorithm( A_rec, k, p, Vm(:, subdiag_min + 1));


    theta = [theta; theta_rec];                         %合并特征值
    x = [x, x_rec];                                     %合并特征向量
    return
end
```

之后检测是否发生特征值deflation

若出现deflation，则修正A矩阵，再重新调用IRAM函数计算剩余特征值

```
    %计算Hm的特征值和特征向量
    [y, theta] = eig(Hm);
    theta = diag(theta);
%    arnoldires = norm(fm) * abs(y(m, :));                %计算残差

    [~, ind] = sort(theta, 'descend');                    %对特征值进行排序
    ind = ind(k + 1:m);                                   %提取剩余的特征值

    et = zeros(m, 1);
    et(m) = 1;                                            %et向量用于跟踪最后一个Arnoldi向量在隐式QR算法中的变换。

    %
    %用QR shitf更新矩阵Hm和Vm，使得Hm更接近于一个上Hessenberg矩阵
    % 而Vm的列向量仍然是A的Krylov子空间的基。
    for i = 1:numel(ind)                                  %计算残差向量
        [Q, R] = qr(Hm - theta(ind(i)) * eye(m));         %计算QR分解
        Hm = R * Q + theta(ind(i)) * eye(m);              %更新Hm
        Vm = Vm * Q;                                      %更新Vm
        et = Q' * et;
    end

    beta = Hm(k + 1, k);
    sigma = et(k);
    fk = beta * Vm(:, k + 1) + sigma * fm;                %计算新的残差fk
    Hk = Hm(1:k, 1:k);                                    %更新Hk
    Vk = Vm(:, 1:k);                                      %更新Vk
end
```

计算当前循环得到的Hessenberg矩阵的特征值
进行QR shift操作，得到新的Hk和Vk，循环。

```
        % 获取A的大小
        [m,n] = size(A);
        % 获取Hk的大小
        k = length(Hk);
        % 初始化V和H
        V = zeros(n, k+p);
        V(:,1:k) = Vk;
        H = zeros(k+p, k+p);
        H(1:k,1:k) = Hk;
        % 初始化残差向量
        f = fk;
        % 进行arnoldi迭代
        for j = 1:p
            beta = norm(f);
            if beta == 0            % 如果残差向量为0，说明已经找到了一个特征值，直接返回
                beta
                flag = 1;
                H = H(1:k+j-1,1:k+j-1);
                V = V(:,1:k+j-1);
                return;
            end
            V(:,k+j) = f/beta;      % 正交化
            H(k+j,k+j-1) = beta;    % 更新Hessenberg矩阵

            w = A*V(:,k+j);

            H(1:k+j, k+j) = V(:,1:k+j)'*w;      % 更新Hessenberg矩阵

            f = w - V(:,1:k+j)*H(1:k+j, k+j);   % 更新残差向量
```

IRAM_Arnoldi_iter函数算法与Arnoldi中类似，Arnoldi_Implicitly_Restart函数源码如图所示，基本算法和Arnoldi迭代类似，只不过对应H，V矩阵的维度有所改变

Ⅲ计算结果分析：

```
IRAM_Algorithm_find_kth_max_eigenvalue_sparse =

   4.8049 + 4.9091i
  -2.5215 - 0.1908i
   2.0037 + 1.5353i
   2.4347 + 0.5490i
  -2.1872 + 1.0877i
   2.1134 - 1.1646i
   2.3849 - 0.2183i
  -2.2415 - 0.7166i
  -0.4361 + 2.3096i
  -1.3296 - 1.9276i
   1.5286 + 1.7045i
   1.4228 - 1.7447i
  -0.7169 - 2.1012i
  -0.0690 - 2.2151i
  -1.2806 + 1.7966i
   2.0123 + 0.8558i
   0.8458 + 2.0140i
  -0.1046 + 2.1480i
  -0.5779 + 2.0389i
  -0.7587 + 1.9641i
```

如图，对比图2中的特征值，可见计算结果正确，精度足够。且在实际运行过程中对100x100的矩阵，该算法速度很快

Ⅳ计算性能分析：

- 时间复杂度：这段代码的主要操作包括矩阵的乘法、QR 分解和特征值计算。以下是对这些操作的性能分析： 1.矩阵乘法：矩阵乘法的时间复杂度为 O(n^3)，其中 n 是矩阵的维数。在这段代码中，矩阵乘法主要出现在 Vm = Vm * Q;这一行。由于 Vm 和 Q 都是 m 维的，所以这一操作的时间复杂度为 O(m^3)。2.QR 分解：QR 分解的时间复杂度也为 O(n^3)，其中 n 是矩阵的维数。在这段代码中，QR 分解主要出现在[Q, R] = qr(Hm - theta(ind(i)) * eye(m));这一行。由于 Hm 是 m 维的，所以这一操作的时间复杂度为 O(m^3)。3.特征值计算：特征值计算的时间复杂度为 O(n^3)，其中 n 是矩阵的维数。在这段代码中，特征值计算主要出现在[y, theta] = eig(Hm);这一行。由于 Hm 是 m 维的，所以这一操作的时间复杂度为 O(m^3)。这段代码的总时间复杂度为 O(m^3)。

- 空间复杂度：对于空间复杂度，这段代码主要存储了几个 m 维的矩阵（如 Hm、Vm、Q 和 R）和向量（如 et 和 theta）。因此，这段代码的空间复杂度为 O(m^2)。

三、项目运行与结果

用matlab打开代码文件夹，直接运行Main脚本