QR Algorithm

Jingxiang Wu

December 30, 2021

Contents

1 Traditional QR Algorithm with double and more shifts **2**
1.1 The Double-Implict-Shift Strategy 2
1.2 The Sextuple-Implicit-Shift Strategy 4

2 Aggressive Early Defaltion for The Multishift QR Algorithm **7**
2.1 The small-bulge multishift QR Algorithm 7
2.2 Aggressive Early Deflation and the details for choices of shifts 9

3 The numerical experiments **11**

A. Reference For Test **15**

B. The Matlab Source Code **16**

Chapter 1

Traditional QR Algorithm with double and more shifts

In this chapter,we will talk about the traditional QR Algorithm with double and more shifts, giving complete algorithm and related details of this algorithm.

1.1 The Double-Implict-Shift Strategy

The Double-Implict-Shift Strategy [3, 4]is published by Francis in 1961-1962, which proposed how to compute a real schur-decomposition. First, we take a hessenberg decomposition to a dense matrix for making it an almost upper quasi-triangular matrix, see **Algorithm 1**

Algorithm 1 Hessenberg

Input: A: a hessenberg matrix
Output: H: a hessenberg matrix;Q: orthogonal transform matrix satisfy $AQ = QH$

$Q = I_n$;
for $i = 1$ to $n - 2$ **do**
  $w = household(H(i+1:n, i))$
  $H(i+1:n, i:n) = (I - 2ww^T)H(i+1:n, i:n)$
  $H(1:n, i+1:n) = H(1:n, i+1:n)(I - 2ww^T)$
  $Q(1:n, i+1:n) = Q(1:n, i+1:n)(I - 2ww^T)$
  $H(i+2:n, i) = 0$
end for

And then selected shifts as the eigenvalues of a trailing principal 2×2 submatrix, the detailed algorithm see **Algorithm 2**

Algorithm 2 Double-shift-QR-iteration

Input: H: a hessenberg matrix
Output: H: a hessenberg matrix;W: a n*3 matrix, stored n-1 household vectors

$s = H(n-1, n-1) + H(n, n)$
$t = H(n-1, n-1)H(n, n) - H(n, n-1)H(n, n-1)$
$x = H(1, 1)H(1, 1) + H(1, 2)H(2, 1) - sH(1, 1) + t$
$y = H(2, 1)(H(1, 1) + H(2, 2) - s)$
$z = H(2, 1)H(3, 2)$
$vector = [x; y; z]$
for $k = 0$ to $n - 3$ **do**
  $w = house(vector)$

$q = max[1, k]$
$r = min[k + 4, n]$
$W(1 : 3, k + 1) = w;$
$H(k + 1 : k + 3, q : n) = (I - 2ww^T)H(k + 1 : k + 3, q : n)$
$H(1 : r, k + 1 : k + 3) = H(1 : r, k + 1 : k + 3)(I - 2ww^T)$
$x = H(k + 2, k + 1);$
$y = H(k + 3, k + 1);$
if $k < n - 3$ **then**
$vector = [x; y; z]$
end if
end for
$w = household([x; y]);$
$W(1 : 2, n - 1) = w$
$H(n - 1 : n, n - 2 : n) = (I - 2ww^T)H(n - 1 : n, n - 2 : n)$
$H(1 : n, n - 1 : n) = H(1 : n, n - 1 : n)(I - 2ww^T)$

After some steps iteration, we can find convergence in both upper left and lower right of this matrix, and after each iteration, it converge at most two eigenvalues in the upper left corner and lower right corner of the matrix.

When it converge only one eigenvalue, it must be a real eigenvalue; when it converge two eigenvalue, most cases we get a pair of conjugate complex eigenvalues. But in some cases, the $H(m, m - 1)$ is not small enough while the $H(m - 1, m - 2)$ is small enough, such as:

$$\begin{pmatrix} \times & \times & \times \\ 10^{-15} & \times & \times \\ 0 & 10^{-7} & \times \end{pmatrix}$$

In this case, we can compute a 2×2 matrix $\widetilde{Q}$ in $O(1)$ time and put it onto Q,$H(1 : i - 1, i : m)$ and $H(i : m, m + 1 : n)$, such that:

$$\begin{pmatrix} \times & \times \\ 10^{-7} & \times \end{pmatrix} \widetilde{Q} = \widetilde{Q} \begin{pmatrix} \lambda_1 & \times \\ 0 & \lambda_2 \end{pmatrix}$$

For a 2×2 matrix which have a pair of conjugate complex eigenvalues $a + bi$, in this case, we can compute a 2×2 matrix $\widetilde{Q}$ in $O(1)$ time and put it onto Q,$H(1 : i - 1, i : m)$ and $H(i : m, m + 1 : n)$, such that:

$$\begin{pmatrix} \times & \times \\ \times & \times \end{pmatrix} \widetilde{Q} = \widetilde{Q} \begin{pmatrix} a & m \\ t & a \end{pmatrix}$$

And at last, if $m > i$ still be true, compute the real schur decompetition of last 2×2 matrix. In these three cases, the orthogonal matrix can be analytically obtained. Based on the above process, we can conclude the following **Algorithm 3**

Algorithm 3 Double-shift-QR-algorithm

Input: A:a dense matrix; $flag$:flag equals 0 means this matrix is a hessenberg matrix
Output: E:all eigenvalue of A;H:a real-schur form matrix;Q:orthogonal transform matrix satisfy $AQ = QH$

$Q = I_n;$
if $flag = 1$ **then**
$[Q, H] = hessenberg(H)$
end if

$i = 1$;
$m = n$;
$tol = 10^{-15}$;
while $m - i + 1 > 2$ **do**
  $[W, H(i:m, i:m)]$ =double-shift-QR-iteration($H(i:m, i:m)$);
  put these household vector to orthogonal transform matrix Q,$H(1:i-1, i:m)$ and $H(i:m, m+1:n)$
  Finding converged eigenvalues in the upper left corner of $(H(i:m, i:m))$
  Finding converged eigenvalues in the lower right corner of $(H(i:m, i:m))$
  if There is no converged eigenvalue **then**
    Continue
  end if
  if a real eigenvalue converges **then**
    if $H(i+1, i) < tol$ **then**
      $E(i, 1) = H(i, i)$
      $i = i + 1$
    end if
    if $H(m, m-1) < tol$ **then**
      $E(m, 1) = H(m, m)$
      $m = m - 1$
    end if
  end if
  if two eigenvalues converge **then**
    if $H(i+2, i+1) < tol$ **then**
      Judge whether this 2×2 matrix has two real eigenvalues or a pair of conjugate complex eigenvalues
      Compute eigenvalue of this 2×2 matrix
      $i = i + 2$
    end if
    if $H(m-1, m-2) < tol$ **then**
      Judge whether this 2×2 matrix has two real eigenvalues or a pair of conjugate complex eigenvalues
      Compute eigenvalue of this 2×2 matrix
      $m = m - 2$
    end if
  end if
end while
if $m > i$) **then**
  Judge whether this least 2×2 matrix has two real eigenvalues or a pair of conjugate complex eigenvalues
  Compute real-schur form of this least 2×2 matrix
  Put $U \in R^{2\times 2}$to Q,$H(1:i-1, i:m)$ and $H(i:m, m+1:n)$
end if

(In the Matlab source code, I use the build-in function *schur* to solve 2×2 matrix)

1.2 The Sextuple-Implicit-Shift Strategy

To get faster convergence speed and higher computational efficiency, we are unsatisfied with introducing two shifts each time. According to the same principle, people proposed

Multishift QR Algorithm. In this section, we introduce and implement the Sextuple-Implicit-Shift Strategy.

As we all know, Implicit QR decomposition use a conclusion in Arnoldi process-the first column determines the whole orthogonal matrix Q. So we construct a vector as a start, this vector is:

$$\prod_{i=1}^{k}(H-\lambda_i I)e_1$$

Noticed that the matrix $\prod_{i=1}^{k}(H-\lambda_i I)$ is the value of characteristic polynomial of $k\times k$ trailing principal submatrix at H, we can get characteristic polynomial of $k\times k$ trailing principal submatrix with symbolic computation. Then we can prove that the vector only have k+1 non-zero elements, which is uniquely determined by $H(1:k+1,1:k+1)$, so we can use build-in function *poly* and *polyvalm* in Matlab to compute this vector. Then do household transformation along with the diagonal line like Double-Implict-Shift. The detailed algorithm see **Algorithm 4**:

Algorithm 4 Sextuple-shift-QR-iteration

Input: H: a hessenberg matrix
Output: H: a hessenberg matrix;W: a n*6 matrix, stored n-1 household vectors

$a = poly(H(n-5:n, n-5:n))$
$vector = polyvalm(a, H(1:7, 1:7))$
for $k = 0$ to $n-3$ **do**
  $w = household(vector)$
  $q = max[1, k]$
  $r = min[k+8, n]$
  $W(1:7, k+1) = w;$
  $H(k+1:k+7, q:n) = (I-2ww^T)H(k+1:k+7, q:n)$
  $H(1:r, k+1:k+7) = H(1:r, k+1:k+7)(I-2ww^T)$
  if $k < n-7$ **then**
    $vector = H(k+2:k+8, k+1)$
  end if
end for
for $i = 1$ to 5 **do**
  $w = household(H(n-6+i:n, n-7+i));$
  $W(1:7-i, n-6+i) = w$
  $H(n-6+i:n, n-7+i:n) = (I-2ww^T)H(n-6+i:n, n-7+i:n)$
  $H(1:n, n-6+i:n) = H(1:n, n-6+i:n)(I-2ww^T)$
end for

After some steps iteration, we can find convergence in both upper left and lower right of this matrix, but in Sextuple-shift-QR-iteration, it is possible to converge k eigenvalues in the upper left and lower right corner of this matrix, $k \le 6$. Then we solve schur decomposition for the matrices of order no more than six by Double-shift-QR-Algorithm. Based on the above process, we can conclude the following **Algorithm 5**:

Algorithm 5 Sextuple-shift-QR-algorithm

Input: A:a dense matrix; $flag$:flag equals 0 means this matrix is a hessenberg matrix
Output: E:all eigenvalue of A;H:a real-schur form matrix;Q:orthogonal transform matrix satisfy $AQ = QH$

$Q = I_n;$

if $flag = 1$ **then**
    $[Q, H] = hessenberg(H)$
end if
$i = 1;$
$m = n;$
$tol = 10^{-15};$
while $m - i + 1 > 6$ **do**
    $[W, H(i : m, i : m)]$ =double-shift-QR-iteration($H(i : m, i : m)$);
    put these household vector to orthogonal transform matrix Q,$H(1 : i - 1, i : m)$ and $H(i : m, m + 1 : n)$
    Finding eigenvalues of converged in the upper left corner of $(H(i : m, i : m))$
    Finding eigenvalues of converged in the lower right corner of $(H(i : m, i : m))$
    if There is no converged eigenvalue **then**
        Continue
    end if
    Suppose the number of converged eigenvalues is k,$k \leq 6$
    use Double-shift-QR-algorithm to $H(m - k + 1 : m, m - k + 1 : m)$, put $\widetilde{Q}$ to $H(1 : m - k, m - k + 1 : m)$,$H(m - k + 1 : m, m + 1 : n)$,$Q(1 : n, m - k + 1 : m)$
end while
if $(m > i)$ **then**
    use Double-shift-QR-algorithm to$H(i : m, i : m)$, put $\widetilde{Q}$ to$H(1 : i - 1, i : m)$,$H(i : m, m + 1 : n)$,$Q(1 : n, i : m)$
end if

The numerical experiments see **chapter 3**, and the Matlab source code see **The Appendix B**.

Chapter 2

Aggressive Early Defaltion for The Multishift QR Algorithm

In this chapter, we will talk about The Multishift QR Algorithm based on BLAS 3 performance and Aggressive Early Deflation Strategy, which is a really aggressive but correct strategy. These methods help us to solve dense unsymmetric matrices of size several thousands without losing too much accuracy.

2.1 The small-bulge multishift QR Algorithm

[1] Find the largest non-negative q and the samllest non-negative p such that:

$$H = \begin{array}{cc} \left[\begin{array}{ccc} H_{11} & H_{12} & H_{13} \\ 0 & H_{22} & H_{23} \\ 0 & 0 & H_{33} \end{array}\right] & \begin{array}{c} p \\ n-p-q \\ q \end{array} \\ \begin{array}{ccc} p & n-p-q & q \end{array} & \end{array}$$

Where H_{33} is upper quasi-triangular and H_{22} is unreduced.

For using BLAS 3 performance in QR iteration, we tend to choose more shifts and longer shifts distance each time to update Q, H_{12} and H_{23}. Unfortunately, if we choose too more shifts every time, the computation will be inaccurate, while a large-bulge will cause m times rounding error to a small-bugle. So we choose small-bulge multishift QR Strategy.

First, we choose $2m$ shifts and choose a pair of complex conjugate eigenvalues or two real eigenvalues every time, and put it to the appropriate position. This process will construct a medium-sized orthogonal transformation matrice, use it to update the right side of the long strip-like matrix, which can be computed by BLAS 3.

Then, we move the m small-bugles k steps each time, this process will construct a medium-sized orthogonal transformation matrice, use it to update both the right and the top side of the long strip-like matrix, which can be computed by BLAS 3.At the same time, it should be noted that the remaining space may not be enough to move k steps when we are close to the lower right corner.

At last, remove bugles in the order in which they are introduced, use the medium-sized orthogonal transformation matrice to update the top side of the long strip-like matrix. The detailed algorithm see **Algorithm 6**

Algorithm 6 Double-shift-chasing-iteration

Input: H: a hessenberg matrix;

Output: H: a hessenberg matrix;

m: we use 2m shifts; k: the length for each move;

Ava:do Ava full movings; $least$:do a moving at last with length of least

W: a three-dimensional array, stored orthogonal transform matrix

choose m and k and compute Ava and least
choose 2m eigenvalue
for $i = 1$ to m **do**
    use a pair of complex conjugate eigenvalues or two real eigenvalues to create a vector, transform it to the position $3 * m - i, 3 * m - i$
    compute orthogonal transform matrix $\widetilde{Q}$ with order of $3m + 1$
end for
put $\widetilde{Q}$ to $H(1 : 3m + 1, 3m + 2 : n)$
for $i = 1$ to Ava **do**
    for $t = 1$ to k **do**
        move t^{th} small-bulge down along with the diagonal line k-steps at once
        compute orthogonal transform matrix $\widetilde{Q}$ with order of $3m + k + 1$
    end for
    put $\widetilde{Q}$ to $H(1 : k(i-1), 1 + k(i-1) : 3m + ki + 1)$ and $H(1 + k(i-1) : 3m + ki + 1, 3m + ki + 2 : n)$
end for
for $t = 1$ to $least$ **do**
    move t^{th} small-bulge down along with the diagonal line least-steps at once
    compute orthogonal transform matrix $\widetilde{Q}$ with order of $3m + k + 1$
end for
put $\widetilde{Q}$ to $H(1 : ki, 1 + ki : n)$
for $i = m$ to 1 **do**
    Starting at the bottom of the matrix, apply each small-bulge to the bottom of the matrix.
    compute orthogonal transform matrix $\widetilde{Q}$ with order of $3m + 1$
end for
put $\widetilde{Q}$ to $H(1 : n - 3m - 1, n - 3m : n)$

Suppose the order of the matrix is n, we can get The amount of calculation done with BLAS 3 is:

$$\begin{aligned}&2(3m+1)^2(n-3m-1) + \sum_{i=1}^{\frac{n-3m-1}{k}} (3m+k+1)^2(k(i-1)+n-3m-ki-1) \\ &= \tfrac{n-3m-1}{k}((3m+k+1)^2(n-(3m+k+1)) + 2k(3m+1)^2)\end{aligned}$$

The amount of calculation done with BLAS 2 is:

$$(3m+k+1)^3 \frac{n-3m-1}{k} + 2(3m+1)^3$$

Choose $m = \lfloor \sqrt{n} \rfloor$and $k = 3m$. Then we can simplify the above equation:

$$\begin{aligned}&BLAS-2 : 216n^2 - 162n\sqrt{n} \\ &BLAS-3 : 36n^2\sqrt{n} - 270n^2 + 486n\sqrt{n}\end{aligned}$$

This shows us that the computation in BLAS-3 is much more than BLAS-2 under this choice.

2.2 Aggressive Early Deflation and the details for choices of shifts

[2] The Aggressive Early Deflation, a strategy for trailing principal submatrix convergence, can increase the speed of convergence.

Suppose we use some eigenvalues of H_{33} to do a small-bulge multishift QR iteration, (the order of H_{33} should be bigger than the number of shifts), and H can be expressed as:

$$H = \begin{array}{cc} \left[\begin{array}{ccc} H_{11} & H_{12} & H_{13} \\ 0 & H_{22} & H_{23} \\ 0 & H_{23} & H_{33} \end{array}\right] & \begin{array}{c} n-p-1 \\ 1 \\ p \end{array} \\ \begin{array}{ccc} n-p-1 & 1 & p \end{array} & \end{array}$$

Where H_{23} is a vector with only one non-zero element at the first position.

Now we do an assumable strategy: Do a real-schur decomposition to H_{33}, this action will recursively call the program that solves the real-schur decomposition, until this matrix is small enough to use the Sextuple-Shift-QR-Algorithm.

Just suppose we finish this programm, which means we can call this function to get real-schur decomposition of H_{33} and orthogonal transformation matrix $\widetilde{Q}$, put this matrix to H_{32} and you can get a full vector. After observation, it was found that this vector has many elements close to 0, which usually concentrated at the bottom of the vector, of course, sometimes it appears in the middle of the vector. Meanwhile, there are still some elements of are not near to zero. The index of the element in the vector corresponds to the position of the eigenvalue in the upper quasi-triangular matrix, and the vector elements corresponding to a pair of eigenvalues of a conjugate complex eigenvalues will converges at the same time. Then if converged element is not at the bottom of vector, do a *orderschur* to swap it to the bottom.(This process is not programmed by me, because I observed that in most cases, eigenvalues converge from bottom to top.) If there are enough eigenvalues converge, we put this orthogonal transformation matrix $\widetilde{Q}$ to $\begin{bmatrix} H_{13} \\ H_{23} \end{bmatrix}$ and H_{32}. Then the matrix becomes the following blocked form:

$$H = \begin{array}{cc} \begin{array}{cccc} H_{11} & H_{12} & H_{13} & H_{14} \\ 0 & H_{22} & H_{23} & H_{24} \\ 0 & s & T_{11} & T_{12} \\ 0 & \varepsilon & 0 & T_{22} \end{array} & \begin{array}{c} n-p-1 \\ 1 \\ k \\ r \end{array} \\ \begin{array}{cccc} n-p-1 & 1 & k & r \end{array} & \end{array}$$

And we can find that r eigenvalues converge and s is much bigger than ε, a more specific mathematical representation is $\|s\|_2 < tol \cdot \|\varepsilon\|_2$. Then just igonre ε, we solve r eigenvalues and then just need to transform H to a hessenberg matrix again. This process can use Givens rotations from the bottom of s, it cost $k-1$ Givens rotations to transform H into a hessenberg matrix, then we solve a smaller eigenvalue system.

Based on the above discussion, we can get the **Algorithm 7**

Algorithm 7 Aggressive-Early-Deflation

Input: A:a dense matrix; $flag$:flag equals 0 means this matrix is a hessenberg matrix
Output: E:all eigenvalue of A;H:a real-schur form matrix;Q:orthogonal transform matrix satisfy $AQ = QH$

```
Q = I_n;
if flag = 1 then
    [Q, H] = hessenberg(H)
```

end if
$i = 1$;
$m = n$;
$tol = 10^{-15}$;
while $m - i + 1 > 100$ **do**
    $[W, H(i:m, i:m), m, k, Ava, least]$ =double-shift-chasing-iteration($H(i:m, i:m)$);
    put these orthogonal transform matrixs to orthogonal transform matrix Q,$H(1:i-1, i:m)$ and $H(i:m, m+1:n)$
    choose si is the size of windows.
    Do a real-schur decompetition to $H(m - si + 1 : m, m - si + 1 : m$
    Compute $\widetilde{Q}^T H(m - si + 1 : m, m - 1)$ and find all converged eigenvalues
    if There is not enough eigenvalues converged **then**
        Continue
    end if
    Do an orderschur to $H(m - si + 1 : m, m - si + 1 : m$ and make every converged eigenvalues to be at the bottom
    Put $\widetilde{Q}$ to $H(1 : m - si, m - si + 1 : n)$ and $H(m - si + 1 : n, m + 1 : n)$
    Do Givens rotations to transfrom H to a hessenberg matrix. Put these Givens to Q
end while
use Sextuple-S0hift-QR-Algorithm to$H(i : m, i : m)$, put $\widetilde{Q}$ to$H(1 : i - 1, i : m)$,$H(i : m, m + 1 : n)$,$Q(1 : n, i : m)$

In my Matlab source code, I use full-hessenberg reduction to transform H after deflation.(Because the givens rotations often go wrong) And according to [2], we can use the eigenvalue in T_{11} to do iteration, but I don't program well according to this principle. It may be that the eigenvalue of T_{11} is not the part of the eigenvalue of the trailing principal submatrix.

Chapter 3

The numerical experiments

In this chapter, we will test the running time of traditional QR Algorithm and Aggressive Early Defaltion, and the numerical stability. The test source code see **The Appendix B** and all matrices are initially generated by the build-in *rand* and each element is between-1 and 1 First, we test the performance of The Double-Shift-QR-Algorithm and Sextuple-Shift-QR-Algorithm, select a matrix from 110 to 1000 and select one every 10.

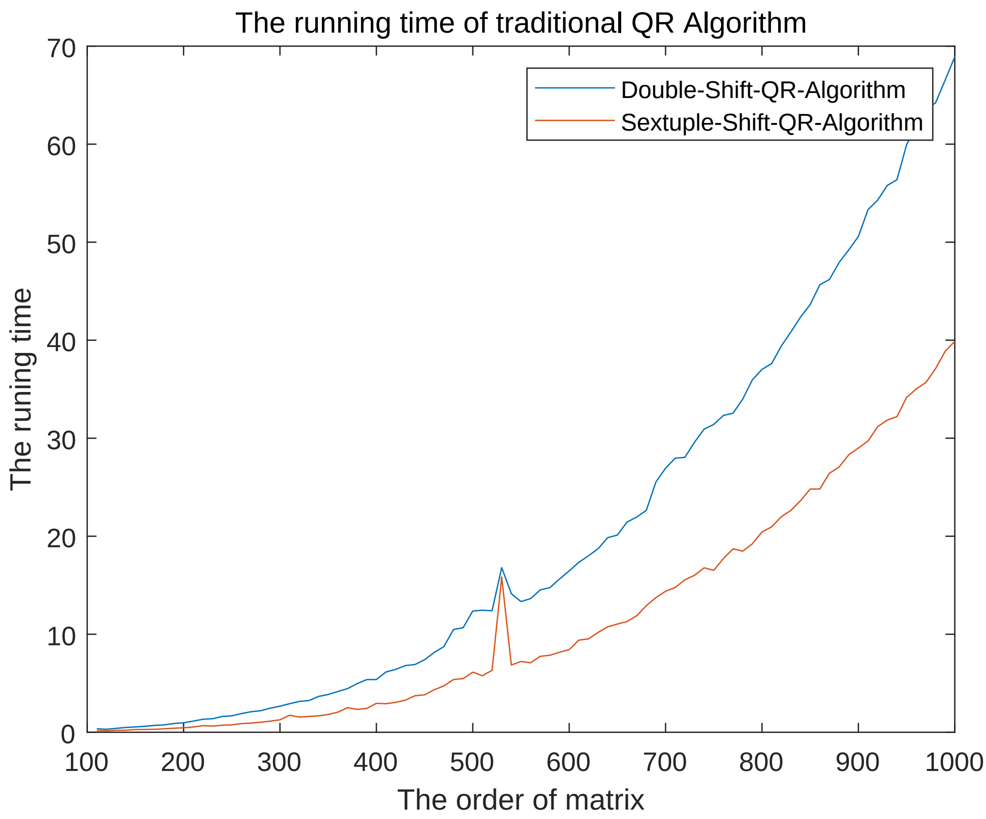


Figure 3.1: The running time of traditional QR Algorithm

This test can shows us that selecting more shifts every time can improve efficiency, so we propose multishift QR Algorithm, and choose small-bugle, which introduced two shifts each time. Then, add Aggressive Early Defaltion in and test again. This time it ranges from 620 to 1200 (since AED does not improve performance significantly in small matrices).

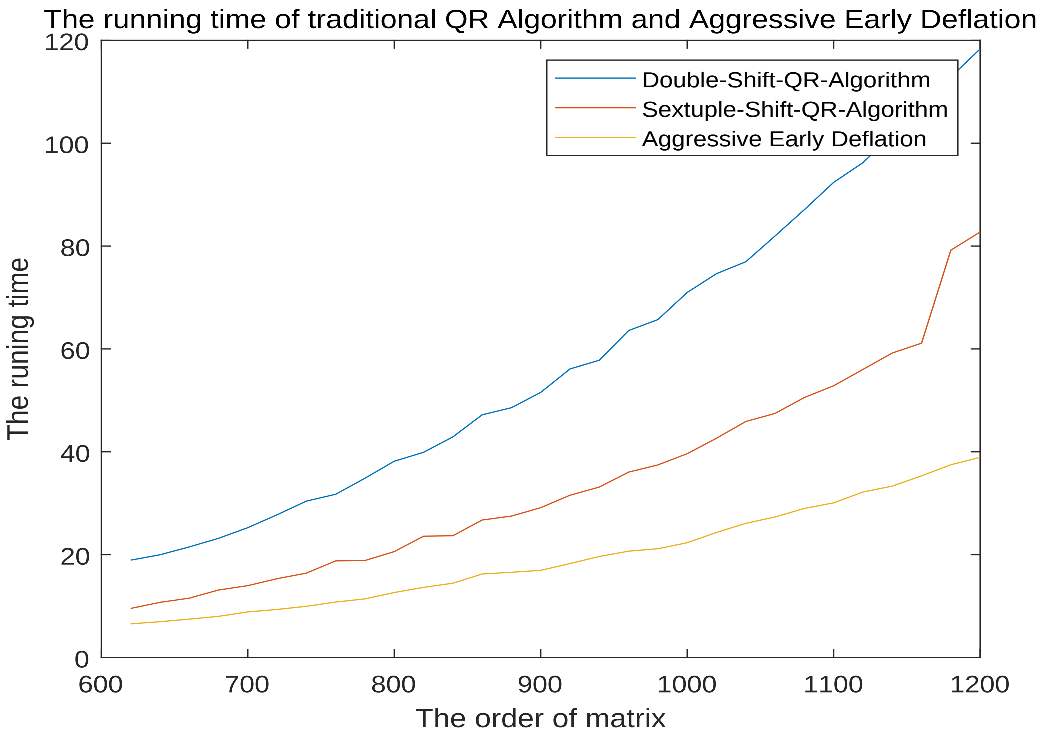


Figure 3.2: The running time of traditional QR Algorithm and Aggressive Early Deflation

In the exploration function of Matlab, we can see that when the matrix order reaches 1000, the time required for hessenberg is close to that for AED processing. Now, we do a numerical experiment to prove it.

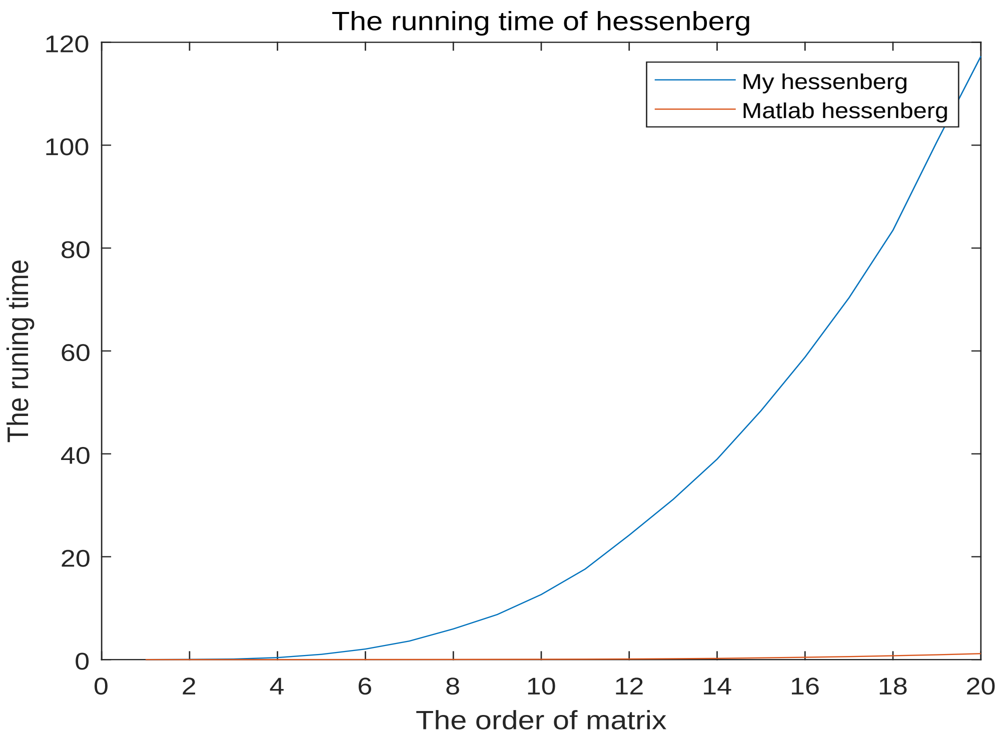


Figure 3.3: The running time of Hessenberg and Matlab function

This can prove that the hessenberg program written by myself does not use BLAS-3 performance when dealing with thousands of matrices, so the performance will be much worse. Then, I use the built-in function *hess* to do following test.

Then we test Aggressive Early Defaltion with *hess* and the built-in function *schur*, select a matrix from 1000 to 5350 and select one every 150.

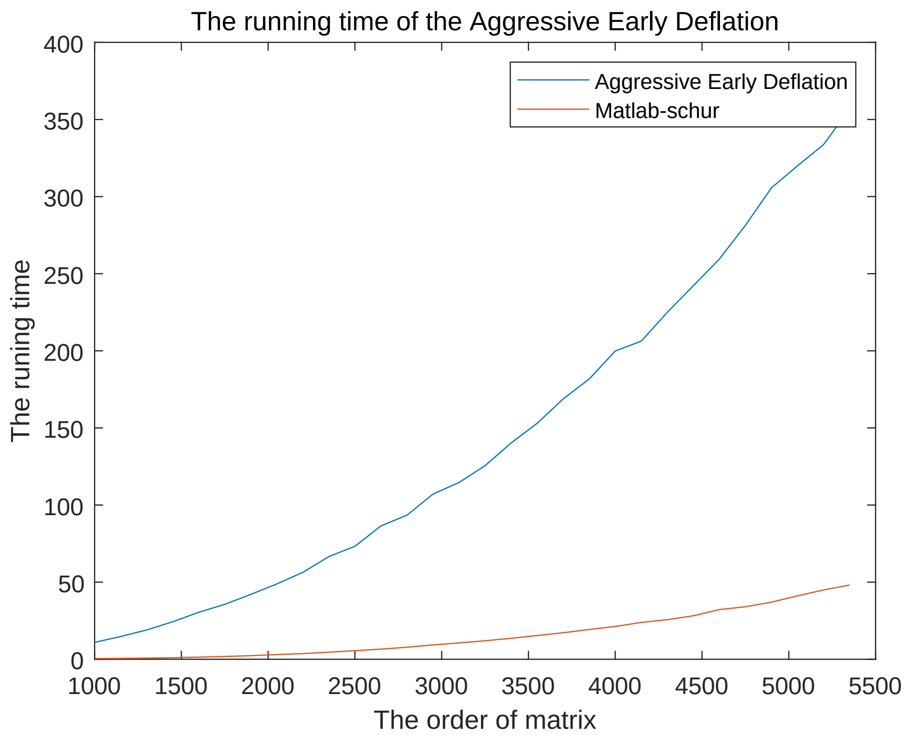
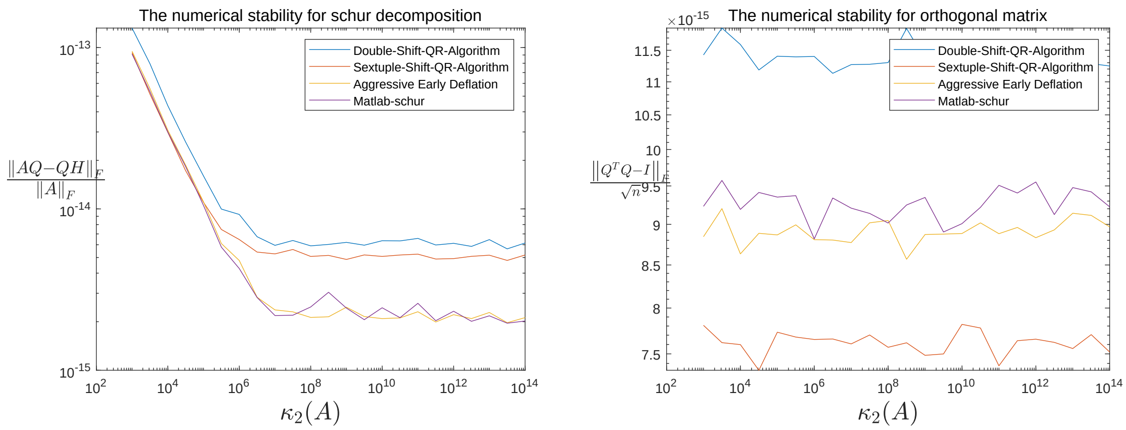


Figure 3.4: The running time of the Aggressive Early Deflation

This test can shows us we can do a real-schur decomposition for a 5000*5000 matrix in about 6 minutes, then let us see the numerical stability.

We choose 23 matrices of order 500, change their condition number from 10^3 to 10^14 by singular value decomposition, then compute $\frac{\|AQ-QH\|_F}{\|A\|_F}$ and $\frac{\left\|Q^TQ-I\right\|_F}{\sqrt{n}}$ Then we can get:



(a) The numerical stability for schur decomposition

(b) The numerical stability for orthogonal matrix

Figure 3.5: The numerical stability

This can show us that the numerical stability is enough. I think maybe because of every orthogonal transformation is made with a home transformation.

As a reference standard, the information about the computer and the time to run the schur decomposition of the built-in matlab function will be given in the **The Appendix A** and all the matlab source code will be given in the **The Appendix B**.

Appendix A.

Reference For Test

We test $schur()$ in Matlab as a reference, see the chart:

The order of matrix	The running time	The order of matrix	The running time
500	0.6778	3500	14.6267
1000	1.0522	4000	21.4383
1500	1.1638	4500	29.2893
2000	3.1954	5000	39.6178
2500	6.4821	5500	53.0618
3000	10.0568	6000	69.6150

And the following chart shows the configuration of my computer

CPU model	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
base frequency	2.6GHz
turbo frequency	5GHz
Core number	6
Threads number	12
L1 cache	384kB
L2 cache	1.5MB
L3 cache	12.0MB
Memory	32GB

Appendix B.

The Matlab Source Code

```
%Do household transformation (I-2ww')x=e
%input:vector x
%output:Household vector w and matrix H
%function [H,w,beta]=household(x)
%n=size(x,1);
%yita=norm(x,inf);
%x=x/yita;
%w=zeros(n,1);
%a=x(2:n,1)'*x(2:n,1);
%w(2:n,1)=x(2:n);
%if a==0
 %   beta=0;
%else
%    alpha=sqrt(w(1,1)^2+a);
%    if x(1)<0
%        w(1)=x(1)-alpha;
%    else
%        w(1)=-a/(x(1)+alpha);
%    end
%    beta=2*w(1,1)^2/(a+w(1,1)^2);
%    w=w/w(1,1);
%end
%H=eye(n);
%H=H-beta*(w*w');
%end
function [w]=household(x)
    n=size(x,1); w=zeros(n,1);
    a=norm(x(2:n),2);
    b=norm(x,2);
    if a==0
        w=0;
    else
        if x(1)<0
            w(1)=x(1)-b;
        else
            w(1)=-a^2/(x(1)+b);
        end
        for i=2:n
            w(i)=x(i);
        end
        t=norm(w,2);
        w=w/t;
    end
end
```

household.m

```
%hessenberg a matrix A
%input:a matrix A
%output:a hessenberg matrix H=QTAQ,the ith household w exists in W ith
%column
function [Q,H]=hessenberg(H)
    [n,~]=size(H);
    Q=eye(n,n);
```

```
    for i=1:n-2
        w=household(H(i+1:n,i));
        H(i+1:n,i:n)=H(i+1:n,i:n)-2*w*(w'*H(i+1:n,i:n));
        H(1:n,i+1:n)=H(1:n,i+1:n)-2*(H(1:n,i+1:n)*w)*w';
        Q(1:n,i+1:n)=Q(1:n,i+1:n)-2*(Q(1:n,i+1:n)*w)*w';
        H(i+2:n,i)=0;
    end
end
```

hessenberg.m

```
%double shift QR iteration
%input:a Hessenberg matrix
%output:after one step QR iteration
%W is 3*n matrix, ith column
function [W,H]=double_shift_QR_iteration(H)
    [n,~]=size(H);
    s=H(n-1,n-1)+H(n,n);
    t=H(n-1,n-1)*H(n,n)-H(n-1,n)*H(n,n-1);
    x=H(1,1)*H(1,1)+H(1,2)*H(2,1)-s*H(1,1)+t;
    y=H(2,1)*(H(1,1)+H(2,2)-s);
    z=H(2,1)*H(3,2);
    vector=[x;y;z];
    W=zeros(3,n-1);
    for k=0:n-3
        w=household(vector);
        W(1:3,k+1)=w;
        q=max([1,k]);
        r=min([k+4,n]);
        H(k+1:k+3,q:n)=H(k+1:k+3,q:n)-2*w*(w'*H(k+1:k+3,q:n));
        H(1:r,k+1:k+3)=H(1:r,k+1:k+3)-2*(H(1:r,k+1:k+3)*w)*w';
        %Q(1:al,k+1:k+3)=Q(1:al,k+1:k+3)-2*(Q(1:al,k+1:k+3)*w)*w';
        x=H(k+2,k+1);
        y=H(k+3,k+1);
        if(k<n-3)
            z=H(k+4,k+1);
        end
        vector=[x;y;z];
    end
    w=household([x;y]);
    W(1:2,n-1)=w;
    H(n-1:n,n-2:n)=H(n-1:n,n-2:n)-2*w*(w'*H(n-1:n,n-2:n));
    H(1:n,n-1:n)=H(1:n,n-1:n)-2*(H(1:n,n-1:n)*w)*w';
end
```

double_shift_QR_iteration.m

```
%double_shift_QR_algorithm
%input:A dense matrix H
%output:A n*2 matrix with all of the eigenvalue of A;
%The first column is the real part and the second column is the second
%column is the imaginary part, which the a+bi is front of a-bi and sort by
%the length of eigenvalues.
function [E,H,Q]=double_shift_QR_algorithm(A,flag)
    [n,~]=size(A);
    D=zeros(n,2);
    E=zeros(n,1);
    H=A;
    if(n==1)
        Q=eye(1);
        E(1,1)=A(1,1);
        return;
    end
    Q=eye(n);
    if flag==1
        %[Q,H]=hessenberg(H);
        [Q,H]=hess(H);
    end
    i=1;
    m=n;
    tol=1e-15;
    while(m-i+1>2)
        [W,H(i:m,i:m)]=double_shift_QR_iteration(H(i:m,i:m));
```

```
        [~,myu]=size(W);
        for tp=1:myu-1
            w=W(1:3,tp);
            Q(1:n,i-1+tp:i+1+tp)=Q(1:n,i-1+tp:i+1+tp)-2*(Q(1:n,i-1+tp:i+1+tp)*w)*w';
            H(1:i-1,i-1+tp:i+1+tp)=H(1:i-1,i-1+tp:i+1+tp)-2*(H(1:i-1,i-1+tp:i+1+tp)*w)*w';
            H(i-1+tp:i+1+tp,m+1:n)=H(i-1+tp:i+1+tp,m+1:n)-2*w*(w'*H(i-1+tp:i+1+tp,m+1:n));
        end
        w=W(1:2,myu);
        Q(1:n,i-1+myu:i+myu)=Q(1:n,i-1+myu:i+myu)-2*(Q(1:n,i-1+myu:i+myu)*w)*w';
        H(1:i-1,i-1+myu:i+myu)=H(1:i-1,i-1+myu:i+myu)-2*(H(1:i-1,i-1+myu:i+myu)*w)*w';
        H(i-1+myu:i+myu,m+1:n)=H(i-1+myu:i+myu,m+1:n)-2*w*(w'*H(i-1+myu:i+myu,m+1:n));
        while(m-i+1>2)
            py=0;
            if(abs(H(m,m-1))<tol)
                py=1;
            end
            if(abs(H(m-1,m-2))<tol)
                py=1;
            end
            if(abs(H(i+1,i))<tol)
                py=1;
            end
            if(abs(H(i+2,i+1))<tol)
                py=1;
            end
            if(py==0)
                break;
            end
            if abs(H(m,m-1))<tol
                H(m,m-1)=0;
                D(m,1)=H(m,m);
                m=m-1;
            else
                if abs(H(m-1,m-2))<tol
                    H(m-1,m-2)=0;
                    t=H(m,m)+H(m-1,m-1);
                    s=H(m,m)*H(m-1,m-1)-H(m,m-1)*H(m-1,m);
                    delta=t^2-4*s;
                    if(delta>=0)
                        D(m,1)=(t+sqrt(abs(delta)))/2;
                        D(m-1,1)=(t-sqrt(abs(delta)))/2;
                    else
                        D(m,1)=t/2;
                        D(m-1,1)=D(m,1);
                        D(m,2)=sqrt(abs(delta))/2;
                        D(m-1,2)=-D(m,2);
                    end
                    [temp_Q,H(m-1:m,m-1:m)]=schur(H(m-1:m,m-1:m));
                    Q(1:n,m-1:m)=Q(1:n,m-1:m)*temp_Q;
                    H(1:m-2,m-1:m)=H(1:m-2,m-1:m)*temp_Q;
                    H(m-1:m,m+1:n)=temp_Q'*H(m-1:m,m+1:n);
                    m=m-2;
                end
            end
            if abs(H(i+1,i))<tol
                H(i+1,i)=0;
                D(i,1)=H(i,i);
                i=i+1;
            else
                if abs(H(i+2,i+1))<tol
                    H(i+2,i+1)=0;
                    t=H(i,i)+H(i+1,i+1);
                    s=H(i,i)*H(i+1,i+1)-H(i,i+1)*H(i+1,i);
                    delta=t^2-4*s;
                    if(delta>=0)
                        D(i,1)=(t+sqrt(abs(delta)))/2;
                        D(i+1,1)=(t-sqrt(abs(delta)))/2;
                    else
                        D(i,1)=t/2;
                        D(i+1,1)=D(i,1);
                        D(i+1,2)=sqrt(abs(delta))/2;
                        D(i,2)=-D(i+1,2);
                    end
```

```
                    [temp_Q,H(i:i+1,i:i+1)]=schur(H(i:i+1,i:i+1));
                    Q(1:n,i:i+1)=Q(1:n,i:i+1)*temp_Q;
                    H(1:i-1,i:i+1)=H(1:i-1,i:i+1)*temp_Q;
                    H(i:i+1,i+2:n)=temp_Q'*H(i:i+1,i+2:n);
                    i=i+2;
                end
            end
        end
    end
%    idea=0;
 %   if(D(i,1)==0)
  %     idea=1;
%    end
 %   if(D(i+1,1)==0)
  %      idea=1;
%    end
 %   if(D(m,1)==0)
  %     idea=1;
   % end
    if(m>i)
        P=H(i:m,i:m);
        temp=eig(P);
        [U,H(i:m,i:m)]=schur(H(i:m,i:m));
        Q(1:n,i:m)=Q(1:n,i:m)*U;
        H(1:i-1,i:m)=H(1:i-1,i:m)*U;
        H(i:m,m+1:n)=U'*H(i:m,m+1:n);
        D(i:m,1)=real(temp);
        D(i:m,2)=imag(temp);
    end
    E(1:n)=D(1:n,1)+1i*D(1:n,2);
    H=triu(H,-1);
    for bella=1:n-1
        if(abs(H(bella+1,bella))<tol)
            H(bella+1,bella)=0;
        end
    end
end
```

double_shift_QR_algorithm.m

```
%The sextuple shift QR iteration
%input:a Hessenberg matrix
%output:after one step QR iteration
%W is 7*n matrix, ith column
function [W,H]=sextuple_shift_QR_iteration (H)
[n,~]=size(H);
a=poly(H(n-5:n,n-5:n));
temp=polyvalm(a,H(1:7,1:7));
vector=temp(1:7,1);
W=zeros(7,n-1);
for k=0:n-7
    w=household(vector);
    q=max([1,k]);
    r=min([k+8,n]);
    H(k+1:k+7,q:n)=H(k+1:k+7,q:n)-2*w*(w'*H(k+1:k+7,q:n));
    H(1:r,k+1:k+7)=H(1:r,k+1:k+7)-2*(H(1:r,k+1:k+7)*w)*w';
    W(1:7,k+1)=w;
    if(k<n-7)
        vector=H(k+2:k+8,k+1);
    end
end
for i=1:5
    w=household(H(n-6+i:n,n-7+i));
    W(1:7-i,n-6+i)=w;
    H(n-6+i:n,n-7+i:n)=H(n-6+i:n,n-7+i:n)-2*w*(w'*H(n-6+i:n,n-7+i:n));
    H(1:n,n-6+i:n)=H(1:n,n-6+i:n)-2*(H(1:n,n-6+i:n)*w)*w';
end
```

sextuple_shift_QR_iteration.m

```
%double_shift_QR_algorithm
%input:A dense matrix H
%output:A n*2 matrix with all of the eigenvalue of A;
```

```
%The first column is the real part and the second column is the second
%column is the imaginary part, which the a+bi is front of a-bi and sort by
%the length of eigenvalues.
function [E,H,Q]=sextuple_shift_QR_algorithm(A,flag)
    [n,~]=size(A);
    D=zeros(n,2);
    E=zeros(n,1);
    H=A;
    Q=eye(n);
    if flag==1
        %[Q,H]=hessenberg(H);
        [Q,H]=hess(H);
    end
    i=1;
    m=n;
    tol=1e-15;
    while(m-i+1>7)
        [W,H(i:m,i:m)]=sextuple_shift_QR_iteration(H(i:m,i:m));
        [~,avavaava]=size(W);
        for tp=1:avavaava-5
            w=W(1:7,tp);
            Q(1:n,i-1+tp:i+5+tp)=Q(1:n,i-1+tp:i+5+tp)-2*(Q(1:n,i-1+tp:i+5+tp)*w)*w';
            H(1:i-1,i-1+tp:i+5+tp)=H(1:i-1,i-1+tp:i+5+tp)-2*(H(1:i-1,i-1+tp:i+5+tp)*w)*w';
            H(i-1+tp:i+5+tp,m+1:n)=H(i-1+tp:i+5+tp,m+1:n)-2*w*(w'*H(i-1+tp:i+5+tp,m+1:n));
        end
        for ilp=1:5
            w=W(1:7-ilp,avavaava-5+ilp);
            Q(1:n,i-1+avavaava+ilp-5:i+avavaava)=Q(1:n,i-1+avavaava+ilp-5:i+avavaava)-2*(Q(1:n
                ,i-1+avavaava+ilp-5:i+avavaava)*w)*w';
            H(1:i-1,i-1+avavaava+ilp-5:i+avavaava)=H(1:i-1,i-1+avavaava+ilp-5:i+avavaava)-2*(H
                (1:i-1,i-1+avavaava+ilp-5:i+avavaava)*w)*w';
            H(i-1+avavaava+ilp-5:i+avavaava,m+1:n)=H(i-1+avavaava+ilp-5:i+avavaava,m+1:n)-2*w
                *(w'*H(i-1+avavaava+ilp-5:i+avavaava,m+1:n));
        end
        while(m-i+1>7)
            py=0;
            pj=0;
            if(abs(H(m,m-1))<tol)
                py=1;
            end
            if(abs(H(m-1,m-2))<tol)
                py=2;
            end
            if(abs(H(m-2,m-3))<tol)
                py=3;
            end
            if(abs(H(m-3,m-4))<tol)
                py=4;
            end
            if(abs(H(m-4,m-5))<tol)
                py=5;
            end
            if(abs(H(m-5,m-6))<tol)
                py=6;
            end
            if(abs(H(i+1,i))<tol)
                pj=1;
            end
            if(abs(H(i+2,i+1))<tol)
                pj=2;
            end
            if(abs(H(i+3,i+2))<tol)
                pj=3;
            end
            if(abs(H(i+4,i+3))<tol)
                pj=4;
            end
            if(abs(H(i+5,i+4))<tol)
                pj=5;
            end
            if(abs(H(i+6,i+5))<tol)
                pj=6;
            end
```

```
            tlpo=0;
            if(py~=0)
                tlpo=tlpo+1;
            end
            if(pj~=0)
                tlpo=tlpo+1;
            end
            if(tlpo==0)
                break;
            end
            if(py~=0)
                switch py
                    case 1
                        H(m,m-1)=0;
                        D(m,1)=H(m,m);
                        m=m-1;
                    case 2
                        H(m-1,m-2)=0;
                        t=H(m,m)+H(m-1,m-1);
                        s=H(m,m)*H(m-1,m-1)-H(m,m-1)*H(m-1,m);
                        delta=t^2-4*s;
                        if(delta>=0)
                            D(m,1)=(t+sqrt(abs(delta)))/2;
                            D(m-1,1)=(t-sqrt(abs(delta)))/2;
                        else
                            D(m,1)=t/2;
                            D(m-1,1)=D(m,1);
                            D(m,2)=sqrt(abs(delta))/2;
                            D(m-1,2)=-D(m,2);
                        end
                        m=m-2;
                    case 3
                        [E0,H(m-2:m,m-2:m),Q0]=double_shift_QR_algorithm(H(m-2:m,m-2:m),0);
                        Q(1:n,m-2:m)=Q(1:n,m-2:m)*Q0;
                        H(m-2:m,m+1:n)=Q0'*H(m-2:m,m+1:n);
                        H(1:m-3,m-2:m)=H(1:m-3,m-2:m)*Q0;
                        D(m-2:m,1)=real(E0);
                        D(m-2:m,2)=imag(E0);
                        m=m-3;
                    case 4
                        [E0,H(m-3:m,m-3:m),Q0]=double_shift_QR_algorithm(H(m-3:m,m-3:m),0);
                        Q(1:n,m-3:m)=Q(1:n,m-3:m)*Q0;
                        H(m-3:m,m+1:n)=Q0'*H(m-3:m,m+1:n);
                        H(1:m-4,m-3:m)=H(1:m-4,m-3:m)*Q0;
                        D(m-3:m,1)=real(E0);
                        D(m-3:m,2)=imag(E0);
                        m=m-4;
                    case 5
                        [E0,H(m-4:m,m-4:m),Q0]=double_shift_QR_algorithm(H(m-4:m,m-4:m),0);
                        Q(1:n,m-4:m)=Q(1:n,m-4:m)*Q0;
                        H(m-4:m,m+1:n)=Q0'*H(m-4:m,m+1:n);
                        H(1:m-5,m-4:m)=H(1:m-5,m-4:m)*Q0;
                        D(m-4:m,1)=real(E0);
                        D(m-4:m,2)=imag(E0);
                        m=m-5;
                    case 6
                        [E0,H(m-5:m,m-5:m),Q0]=double_shift_QR_algorithm(H(m-5:m,m-5:m),0);
                        Q(1:n,m-5:m)=Q(1:n,m-5:m)*Q0;
                        H(m-5:m,m+1:n)=Q0'*H(m-5:m,m+1:n);
                        H(1:m-6,m-5:m)=H(1:m-6,m-5:m)*Q0;
                        D(m-5:m,1)=real(E0);
                        D(m-5:m,2)=imag(E0);
                        m=m-6;
                end
            end
            if (pj~=0)
                switch pj
                    case 1
                        H(i+1,i)=0;
                        D(i,1)=H(i,i);
                        i=i+1;
                    case 2
                        H(i+2,i+1)=0;
```

```
                        t=H(i,i)+H(i+1,i+1);
                        s=H(i,i)*H(i+1,i+1)-H(i,i+1)*H(i+1,i);
                        delta=t^2-4*s;
                        if(delta>=0)
                            D(i,1)=(t+sqrt(abs(delta)))/2;
                            D(i+1,1)=(t-sqrt(abs(delta)))/2;
                        else
                            D(i,1)=t/2;
                            D(i+1,1)=D(i,1);
                            D(i+1,2)=sqrt(abs(delta))/2;
                            D(i,2)=-D(i+1,2);
                        end
                        i=i+2;
                    case 3
                        [E0,H(i:i+2,i:i+2),Q0]=double_shift_QR_algorithm(H(i:i+2,i:i+2),0);
                        Q(1:n,i:i+2)=Q(1:n,i:i+2)*Q0;
                        H(i:i+2,i+3:n)=Q0'*H(i:i+2,i+3:n);
                        H(1:i-1,i:i+2)=H(1:i-1,i:i+2)*Q0;
                        D(i:i+2,1)=real(E0);
                        D(i:i+2,2)=imag(E0);
                        i=i+3;
                    case 4
                        [E0,H(i:i+3,i:i+3),Q0]=double_shift_QR_algorithm(H(i:i+3,i:i+3),0);
                        Q(1:n,i:i+3)=Q(1:n,i:i+3)*Q0;
                        H(i:i+3,i+4:n)=Q0'*H(i:i+3,i+4:n);
                        H(1:i-1,i:i+3)=H(1:i-1,i:i+3)*Q0;
                        D(i:i+3,1)=real(E0);
                        D(i:i+3,2)=imag(E0);
                        i=i+4;
                    case 5
                        [E0,H(i:i+4,i:i+4),Q0]=double_shift_QR_algorithm(H(i:i+4,i:i+4),0);
                        Q(1:n,i:i+4)=Q(1:n,i:i+4)*Q0;
                        H(i:i+4,i+5:n)=Q0'*H(i:i+4,i+5:n);
                        H(1:i-1,i:i+4)=H(1:i-1,i:i+4)*Q0;
                        D(i:i+4,1)=real(E0);
                        D(i:i+4,2)=imag(E0);
                        i=i+5;
                    case 6
                        [E0,H(i:i+5,i:i+5),Q0]=double_shift_QR_algorithm(H(i:i+5,i:i+5),0);
                        Q(1:n,i:i+5)=Q(1:n,i:i+5)*Q0;
                        H(i:i+5,i+6:n)=Q0'*H(i:i+5,i+6:n);
                        H(1:i-1,i:i+5)=H(1:i-1,i:i+5)*Q0;
                        D(i:i+5,1)=real(E0);
                        D(i:i+5,2)=imag(E0);
                        i=i+6;
                end
            end
        end
    end
    if(m>i)
        [temp,H(i:m,i:m),Q0]=double_shift_QR_algorithm(H(i:m,i:m),0);
        Q(1:n,i:m)=Q(1:n,i:m)*Q0;
        H(i:m,m+1:n)=Q0'*H(i:m,m+1:n);
        H(1:i-1,i:m)=H(1:i-1,i:m)*Q0;
        D(i:m,1)=real(temp);
        D(i:m,2)=imag(temp);
    end
    E(1:n)=D(1:n,1)+1i*D(1:n,2);
    H=triu(H,-1);
    for bella=1:n-1
        if(abs(H(bella+1,bella))<tol)
            H(bella+1,bella)=0;
        end
    end
end
```

sextuple_shift_QR_algorithm.m

```
%chasing QR
function [W0,H,m,k,Ava,least]=double_shift_chasing_iteration(H)
[n,~]=size(H);
m=floor(sqrt(n));
k=3*m;
```

```
%some_eigenvalue=size(D,1);
%and_eigenvalue=3*m-some_eigenvalue;
Ava=floor((n-3*m-1)/k);
W0=zeros(3*m+k+1,3*m+k+1,Ava+3);
%start
W=eye(3*m+1);
%E0=D(1:some_eigenvalue,1)+1i*D(1:some_eigenvalue,2);
%all_eigenvalue=eig(H(n-3*m+1:n,n-3*m+1:n));
[Eig,~,~]=aggressive_early_deflation(H(n-3*m+1:n,n-3*m+1:n),0);
%[E1,E2,~,~]=choose(eig(H(n-3*m+1:n,n-3*m+1:n)));
[E1,E2,~,~]=choose(Eig);
Eigenvalue=[E2;E1];
for i=1:m
     s=real(Eigenvalue(2*i-1,1)+Eigenvalue(2*i,1));
     t=real(Eigenvalue(2*i-1,1)*Eigenvalue(2*i,1));
     x=H(1,1)*H(1,1)+H(1,2)*H(2,1)-s*H(1,1)+t;
     y=H(2,1)*(H(1,1)+H(2,2)-s);
     z=H(2,1)*H(3,2);
     vector=[x;y;z];
     for Diana=0:3*(m-i)
         w=household(vector);
         q=max([1,Diana]);
         r=min([Diana+4,3*m+1]);
         H(Diana+1:Diana+3,q:(3*m+1))=H(Diana+1:Diana+3,q:(3*m+1))-2*w*(w'*H(Diana+1:Diana+3,q
             :(3*m+1)));
         H(1:r,Diana+1:Diana+3)=H(1:r,Diana+1:Diana+3)-2*(H(1:r,Diana+1:Diana+3)*w)*w';
         vector=H(Diana+2:Diana+4,Diana+1);
         W(1:3*m+1,Diana+1:Diana+3)=W(1:3*m+1,Diana+1:Diana+3)-2*(W(1:3*m+1,Diana+1:Diana+3)*w
             )*w';
     end
end
H(1:3*m+1,(3*m+2):n)=W'*H(1:3*m+1,(3*m+2):n);
W0(1:3*m+1,1:3*m+1,1)=W;
%middle
for i=1:Ava
    W=eye(3*m+k+1);
    for t=1:k
        VEC=zeros(3,m);
        for pen=m:-1:1
            position=1+k*(i-1)+t-1+3*(pen-1);
            vector=H(position+1:position+3,position);
            w=household(vector);
            VEC(1:3,pen)=w;
            H(position+1:position+3,position:3*m+k*i+1)= H(position+1:position+3,position:3*m+
                k*i+1)- 2*w*(w'*H(position+1:position+3,position:3*m+k*i+1));
            H(1+k*(i-1):position+4,position+1:position+3)=H(1+k*(i-1):position+4,position+1:
                position+3)-2*(H(1+k*(i-1):position+4,position+1:position+3)*w)*w';
        end
        for plp=1:m
            w=VEC(1:3,plp);
            W(1:3*m+k+1,t+3*plp-2:t+3*plp)=W(1:3*m+k+1,t+3*plp-2:t+3*plp)-2*(W(1:3*m+k+1,t+3*
                plp-2:t+3*plp)*w)*w';
        end
    end
    H(1:k*(i-1),1+k*(i-1):3*m+k*i+1)=H(1:k*(i-1),1+k*(i-1):3*m+k*i+1)*W;
    H(1+k*(i-1):3*m+k*i+1,3*m+k*i+2:n)=W'*H(1+k*(i-1):3*m+k*i+1,3*m+k*i+2:n);
    W0(1:1+3*m+k,1:1+3*m+k,i+1)=W;
end
least=n-k*Ava-3*m-1;
W=eye(3*m+least+1);
for t=1:least
    VEC=zeros(3,m);
        for pen=m:-1:1
            position=1+k*i+t-1+3*(pen-1);
            vector=H(position+1:position+3,position);
            w=household(vector);
            VEC(1:3,pen)=w;
            H(position+1:position+3,position:n)= H(position+1:position+3,position:n)- 2*w*(w'*
                H(position+1:position+3,position:n));
            H(1+k*i:position+4,position+1:position+3)=H(1+k*i:position+4,position+1:position
                +3)-2*(H(1+k*i:position+4,position+1:position+3)*w)*w';
        end
        for plp=1:m
```

```
            w=VEC(1:3,plp);
            W(1:3*m+least+1,t+3*plp-2:t+3*plp)=W(1:3*m+least+1,t+3*plp-2:t+3*plp)-2*(W(1:3*m+least+1,t+3*plp-2:t+3*plp)*w)*w';
        end
end
H(1:k*i,1+k*i:n)=H(1:k*i,1+k*i:n)*W;
W0(1:1+3*m+least,1:1+3*m+least,Ava+2)=W;
%end:
W=eye(3*m+1);
for bella=m:-1:1
    position=n-3*m+3*(bella-1);
    for Carol=position:n-3
        r=min([Carol+4,n]);
        w=household(H(Carol+1:Carol+3,Carol));
        H(Carol+1:Carol+3,Carol:n)=H(Carol+1:Carol+3,Carol:n)-2*w*(w'*H(Carol+1:Carol+3,Carol:n));
        H(n-3*m:r,Carol+1:Carol+3)=H(n-3*m:r,Carol+1:Carol+3)-2*(H(n-3*m:r,Carol+1:Carol+3)*w)*w';
        W(1:3*m+1,Carol+2-n+3*m:Carol+4-n+3*m)=W(1:3*m+1,Carol+2-n+3*m:Carol+4-n+3*m)-2*(W(1:3*m+1,Carol+2-n+3*m:Carol+4-n+3*m)*w)*w';
    end
    w=household(H(n-1:n,n-2));
    H(n-3*m:n,n-1:n)=H(n-3*m:n,n-1:n)-2*(H(n-3*m:n,n-1:n)*w)*w';
    H(n-1:n,n-2:n)=H(n-1:n,n-2:n)-2*w*(w'*H(n-1:n,n-2:n));
    W(1:3*m+1,3*m:3*m+1)=W(1:3*m+1,3*m:3*m+1)-2*(W(1:3*m+1,3*m:3*m+1)*w)*w';
end
H(1:n-3*m-1,n-3*m:n)=H(1:n-3*m-1,n-3*m:n)*W;
W0(1:1+3*m,1:1+3*m,Ava+3)=W;
m=floor(sqrt(n));
k=3*m;
Ava=floor((n-3*m-1)/k);
```

double_shift_chasing_iteration.m

```
%search eigenvalue in a real schur
%input: a real schur matrix H
%output:A n*2 matrix D
function D=eig_search(H)
n=size(H,1);
D=zeros(n,2);
i=1;
while(i<n+1)
    if(i==n||H(i+1,i)==0)
        D(i,1)=H(i,i);
        i=i+1;
    else
        t=H(i,i)+H(i+1,i+1);
        s=H(i,i)*H(i+1,i+1)-H(i,i+1)*H(i+1,i);
        delta=t^2-4*s;
        if(delta>=0)
            D(i,1)=(t+sqrt(abs(delta)))/2;
            D(i+1,1)=(t-sqrt(abs(delta)))/2;
        else
            D(i,1)=t/2;
            D(i+1,1)=D(i,1);
            D(i+1,2)=sqrt(abs(delta))/2;
            D(i,2)=-D(i+1,2);
        end
        i=i+2;
    end
end
end
```

eig_search.m

```
%choose eigenvalue
%input:E is a sorted eigenvalue from smalll to largest, k is the number we
%needed and k<n
%output:a E1 with real eigenvalue and a E2 with complex eigenvalue
function [E1,E2,t,m]=choose(E)
n=size(E,1);
E1=zeros(n,1);
E2=zeros(n,1);
```

```
t=1;%t -E1
m=1;%m -E2
for i=1:n
    if(imag(E(n-i+1,1))==0)
        E1(t,1)=real(E(n-i+1,1));
        t=t+1;
    else
        E2(m,1)=E(n-i+1,1);
        m=m+1;
    end
end
E1=E1(1:t-1,1);
E2=E2(1:m-1,1);
t=t-1;
m=m-1;
```

choose.m

```
%Aggressive early deflation method
%input:A dense matrix A, flag for hessenberg decompesition
%output:E,Q,H
function [E,Q,H]=aggressive_early_deflation(A,flag)
    [n,~]=size(A);
    D=zeros(n,2);
    E=zeros(n,1);
    H=A;
    Q=eye(n);
    if flag==1
        %[Q,H]=hessenberg(H);
        [Q,H]=hess(H);
    end
    i=1;
    m=n;
    tol1=1e-15;
    tol2=1e-15;
    %some_eigenvalue=H(2:1,1:2);
    while(m-i+1>100)
        [W0,H(i:m,i:m),lpa,k,tense,least]=double_shift_chasing_iteration(H(i:m,i:m));
        %some_eigenvalue=H(2:1,1:2);
        position2=i+3*lpa;
        size_l=3*lpa+1;
        Q(1:n,i:position2)=Q(1:n,i:i+3*lpa)*W0(1:size_l,1:size_l,1);
        H(1:i-1,i:position2)=H(1:i-1,i:position2)*W0(1:size_l,1:size_l,1);
        H(i:position2,m+1:n)=W0(1:size_l,1:size_l,1)'*H(i:position2,m+1:n);
        size_l=3*lpa+k+1;
        for laps=2:tense+1
            position1=i+(laps-2)*k;
            position2=3*lpa+(laps-1)*k+i;
            Q(1:n,position1:position2)=Q(1:n,position1:position2)*W0(1:size_l,1:size_l,laps);
            H(1:i-1,position1:position2)=H(1:i-1,position1:position2)*W0(1:size_l,1:size_l,
                laps);
            H(position1:position2,m+1:n)=W0(1:size_l,1:size_l,laps)'*H(position1:position2,m
                +1:n);
        end
        position1=m-(3*lpa+least);
        size_l=1+3*lpa+least;
        Q(1:n,position1:m)=Q(1:n,position1:m)*W0(1:size_l,1:size_l,tense+2);
        H(1:i-1,position1:m)=H(1:i-1,position1:m)*W0(1:size_l,1:size_l,tense+2);
        H(position1:m,m+1:n)=W0(1:size_l,1:size_l,tense+2)'*H(position1:m,m+1:n);
        position1=m-3*lpa;
        size_l=1+3*lpa;
        Q(1:n,position1:m)=Q(1:n,position1:m)*W0(1:size_l,1:size_l,tense+3);
        H(1:i-1,position1:m)=H(1:i-1,position1:m)*W0(1:size_l,1:size_l,tense+3);
        H(position1:m,m+1:n)=W0(1:size_l,1:size_l,tense+3)'*H(position1:m,m+1:n);
        if abs(H(i+1,i))<tol1
            H(i+1,i)=0;
            D(i,1)=H(i,i);
            i=i+1;
        else
            if abs(H(i+2,i+1))<tol1
                H(i+2,i+1)=0;
                t=H(i,i)+H(i+1,i+1);
                s=H(i,i)*H(i+1,i+1)-H(i,i+1)*H(i+1,i);
```

```
                delta=t^2-4*s;
                if(delta>=0)
                    D(i,1)=(t+sqrt(abs(delta)))/2;
                    D(i+1,1)=(t-sqrt(abs(delta)))/2;
                else
                    D(i,1)=t/2;
                    D(i+1,1)=D(i,1);
                    D(i+1,2)=sqrt(abs(delta))/2;
                    D(i,2)=-D(i+1,2);
                end
                i=i+2;
            end
        end
        [~,Q0,temp_matrix]=aggressive_early_deflation(H(m-3*lpa+1:m,m-3*lpa+1:m),1);
        %[Q0,temp_matrix]=schur(H(m-3*lpa+1:m,m-3*lpa+1:m));
        temp_vector=Q0'*H(m-3*lpa+1:m,m-3*lpa);
        ava=0;
        while(ava~=3*lpa&&abs(temp_vector(3*lpa-ava,1))<tol2)
            temp_vector(3*lpa-ava,1)=0;
            ava=ava+1;
        end
        if(ava==0)
            continue;
        else
            if(ava~=3*lpa&&temp_matrix(3*lpa-ava+1,3*lpa-ava)~=0)
            temp_vector(3*lpa-ava,1)=0;
            ava=ava+1;
            end
            if(ava<lpa)
                continue;
            else
                H(m-3*lpa+1:m,m-3*lpa+1:m)=temp_matrix;
                H(m-3*lpa+1:m,m-3*lpa)=temp_vector;
                D(m-ava+1:m,1:2)=eig_search(H(m-ava+1:m,m-ava+1:m));
                %some_eigenvalue=eig_search(H(m-3*lpa+1:m-ava,m-3*lpa+1:m-ava));
                Q(1:n,m-3*lpa+1:m)=Q(1:n,m-3*lpa+1:m)*Q0;
                H(1:m-3*lpa,m-3*lpa+1:m)=H(1:m-3*lpa,m-3*lpa+1:m)*Q0;
                H(m-3*lpa+1:m,m+1:n)=Q0'*H(m-3*lpa+1:m,m+1:n);
                %[Q0,H(m-3*lpa:m-ava,m-3*lpa:m-ava)]=hessenberg(H(m-3*lpa:m-ava,m-3*lpa:m-ava)
                    );
                [Q0,H(m-3*lpa:m-ava,m-3*lpa:m-ava)]=hess(H(m-3*lpa:m-ava,m-3*lpa:m-ava));
                Q(1:n,m-3*lpa:m-ava)=Q(1:n,m-3*lpa:m-ava)*Q0;
                H(1:m-3*lpa-1,m-3*lpa:m-ava)=H(1:m-3*lpa-1,m-3*lpa:m-ava)*Q0;
                H(m-3*lpa:m-ava,m-ava+1:n)=Q0'*H(m-3*lpa:m-ava,m-ava+1:n);
                m=m-ava;
            end
        end
    end
    [E0,H(i:m,i:m),Q0]=sextuple_shift_QR_algorithm(H(i:m,i:m),0);
    D(i:m,1)=real(E0);
    D(i:m,2)=imag(E0);
    Q(1:n,i:m)=Q(1:n,i:m)*Q0;
    H(1:i-1,i:m)=H(1:i-1,i:m)*Q0;
    H(i:m,m+1:n)=Q0'*H(i:m,m+1:n);
    E(1:n)=D(1:n,1)+1i*D(1:n,2);
end
```

aggressive_early_deflation.m

```
%project-traditional-running-time
clear;
clc;
time_double=zeros(90,1);
time_sextuple=zeros(90,1);
for i=110:10:1000
        A=rand(i,i);
        tic;
        [E,H,Q]=double_shift_QR_algorithm(A,1);
        time_double((i-100)/10,1)=toc;
        tic;
        [E,H,Q]=sextuple_shift_QR_algorithm(A,1);
        time_sextuple((i-100)/10,1)=toc;
end
```

```
index=[110:10:1000]';
plot(index,time_double);
hold on;
plot(index,time_sextuple);
title('The running time of traditional QR Algorithm');
xlabel('The order of matrix');
ylabel('The runing time');
legend('Double-Shift-QR-Algorithm','Sextuple-Shift-QR-Algorithm');
```

project_traditional_running_time.m

```
%project-AED-running-time1
clear;
clc;
time_double=zeros(30,1);
time_sextuple=zeros(30,1);
time_AED=zeros(30,1);
for i=620:20:1200
        A=rand(i,i);
        tic;
        [E,H,Q]=double_shift_QR_algorithm(A,1);
        time_double((i-600)/20,1)=toc;
        tic;
        [E,H,Q]=sextuple_shift_QR_algorithm(A,1);
        time_sextuple((i-600)/20,1)=toc;
        tic;
        [E,Q,H]=aggressive_early_deflation(A,1);
        time_AED((i-600)/20,1)=toc;
end
index=[620:20:1200]';
plot(index,time_double);
hold on;
plot(index,time_sextuple);
hold on;
plot(index,time_AED);
title('The running time of traditional QR Algorithm and Aggressive Early Deflation');
xlabel('The order of matrix');
ylabel('The runing time');
legend('Double-Shift-QR-Algorithm','Sextuple-Shift-QR-Algorithm','Aggressive Early Deflation')
    ;
```

project_AED_running_time1.m

```
%project_hessenberg
time_my=zeros(20,1);
time_matlab=zeros(20,1);
for n=100:100:2000
    temp_time_my=zeros(5,1);
    temp_time_matlab=zeros(5,1);
    for lp=1:5
        A=rand(n,n);
        tic;
        [Q,H]=hessenberg(A,1);
        temp_time_my(lp,1)=toc;
        tic;
        [Q,H]=hess(A);
        temp_time_matlab(lp,1)=toc;
    end
    time_my((n-100)/100+1,1)=sum(temp_time_my)/5;
    time_matlab((n-100)/100+1,1)=sum(temp_time_matlab)/5;
end
plot([1:20],time_my);
hold on;
plot([1:20],time_matlab);
title('The running time of hessenberg');
xlabel('The order of matrix');
ylabel('The runing time');
legend('My hessenberg','Matlab hessenberg');
```

project_hessenberg.m

```
%project-AED-running-time2
```

```
clear;
clc;
time_AED=zeros(30,1);
time_Matlab=zeros(30,1);
for i=1000:150:5350
    A=rand(i,i);
    tic;
    [E,Q,H]=aggressive_early_deflation(A,1);
    time_AED((i-850)/150,1)=toc;
    tic;
    [Q,H]=schur(A);
    time_Matlab((i-850)/150,1)=toc;
end
index=[1000:150:5350]';
plot(index,time_AED);
hold on;
plot(index,time_Matlab);
title('The running time of the Aggressive Early Deflation');
xlabel('The order of matrix');
ylabel('The runing time');
legend('Aggressive Early Deflation','Matlab-schur');
```

project_AED_running_time2.m

```
%project_ill-conditioned matrix
A=-1+2*rand(500,500,23);
matrix_norm=zeros(23,1);
condition_number=zeros(23,1);
for i=1:23
    [S,V,D]=svd(A(1:500,1:500,i));
    V(1,1)=10^((i+5)/4);
    V(500,500)=10^((-i-5)/4);
    A(1:500,1:500,i)=S'*V*D;
    matrix_norm(i,1)=V(1,1);
    condition_number(i,1)=V(1,1)^2;
end
double_=zeros(23,1);
sextuple_=zeros(23,1);
AED_=zeros(23,1);
Matlab_=zeros(23,1);
double_Q=zeros(23,1);
sextuple_Q=zeros(23,1);
AED_Q=zeros(23,1);
Matlab_Q=zeros(23,1);
for i=1:23
    T=A(1:500,1:500,i);
    [E,H,Q]=double_shift_QR_algorithm(T,1);
    double_(i,1)=norm(T*Q-Q*H,'fro');
    double_Q(i,1)=norm(Q'*Q-eye(500),'fro');
    [E,H,Q]=sextuple_shift_QR_algorithm(T,1);
    sextuple_(i,1)=norm(T*Q-Q*H,'fro');
    sextuple_Q(i,1)=norm(Q'*Q-eye(500),'fro');
    [E,Q,H]=aggressive_early_deflation(T,1);
    AED_(i,1)=norm(T*Q-Q*H,'fro');
    AED_Q(i,1)=norm(Q'*Q-eye(500),'fro');
    [Q,H]=schur(T);
    Matlab_(i,1)=norm(T*Q-Q*H,'fro');
    Matlab_Q(i,1)=norm(Q'*Q-eye(500),'fro');
end
double_=double_./matrix_norm;
sextuple_=sextuple_./matrix_norm;
AED_=AED_./matrix_norm;
Matlab_=Matlab_./matrix_norm;
double_Q=double_Q./sqrt(500);
sextuple_Q=sextuple_Q./sqrt(500);
AED_Q=AED_Q./sqrt(500);
Matlab_Q=Matlab_Q./sqrt(500);
hold off;
loglog(condition_number,double_);
hold on;
loglog(condition_number,sextuple_);
hold on;
loglog(condition_number,AED_);
```

```
hold on;
loglog(condition_number,Matlab_);
xlabel('$\kappa_2(A)$','Interpreter','latex','FontSize',18);
ylabel('$\frac{\|AQ-QH\|_F}{\|A\|_F}$','Interpreter','latex','rotation',0,'FontSize',18);
title('The numerical stability for schur decomposition','FontSize',12);
legend('Double-Shift-QR-Algorithm','Sextuple-Shift-QR-Algorithm','Aggressive Early Deflation',
    'Matlab-schur');
hold off;
loglog(condition_number,double_Q);
hold on;
loglog(condition_number,sextuple_Q);
hold on;
loglog(condition_number,AED_Q);
hold on;
loglog(condition_number,Matlab_Q);
hold on;
legend('Double-Shift-QR-Algorithm','Sextuple-Shift-QR-Algorithm','Aggressive Early Deflation',
    'Matlab-schur');
xlabel('$\kappa_2(A)$','Interpreter','latex','FontSize',18);
ylabel('$\frac{\|Q^TQ-I\|_F}{\sqrt{n}}$','Interpreter','latex','rotation',0,'FontSize',15);
title('The numerical stability for orthogonal matrix','FontSize',12);
```

project_the_numerical_stability.m

Reference

[1] Karen Braman and Ralph Byers. The multishift qr algorithm. part i: Maintaining well-focused shifts and level 3 performance. *SIAM Journal on Matrix Analysis and Applications*, 23, 09 2002.

[2] Karen Braman and Ralph Byers. The multishift qr algorithm. part ii: Aggressive early deflation. *SIAM Journal on Matrix Analysis and Applications*, 23, 09 2002.

[3] J. Francis. The qr transformation a unitary analogue to the lr transformation–part 1. *The Computer Journal - CJ*, 4:265–271, 03 1961.

[4] J. G. F. Francis. The QR Transformation—Part 2. *The Computer Journal*, 4(4):332–345, 01 1962.