Report on Implementing Symmetric QR Algorithm

Name: Lin Zikai (林子开)

School: Data Science, Fudan University

ID: 21307110161

## Background

The symmetric QR algorithm is one of the most classical iteration methods for symmetric eigenvalue problem in numerical linear algebra. It consists of tridiagonalization, Wilkinson Shift and implicit iteration, which are important techniques that are supposed to be obtained after learning this course. Moreover, in many situations, problems of symmetric eigenvalues will relatively more frequently occur. Therefore, the topic of implementing symmetric QR algorithm is chosen.

## Code Structure

The symmetric QR algorithm is principally composed of preparing functions and the main function.

The preparing functions include the ***upper\_hessenber*** function, the ***block\_size*** function, and the ***one\_step\_QR\_with\_Wilkinson\_shift*** function. And the main function is called ***QR\_Algorithm\_for\_symmatric\_matrix***. All functions above are stored separately. Thus, it is convenient to exam each of them.

Now we can have a closer and detailed look at the implementation. First, generate a random symmetric real matrix, A. All the elements of A are uniformly distributed between 0 and 1.

Secondly, turn the symmetric matrix A into an upper Hessenberg matrix, T. This process needs to invoke the function ***upper\_hessenber***. Since this process can preserve the symmetricity, the matrix T is tridiagonal.

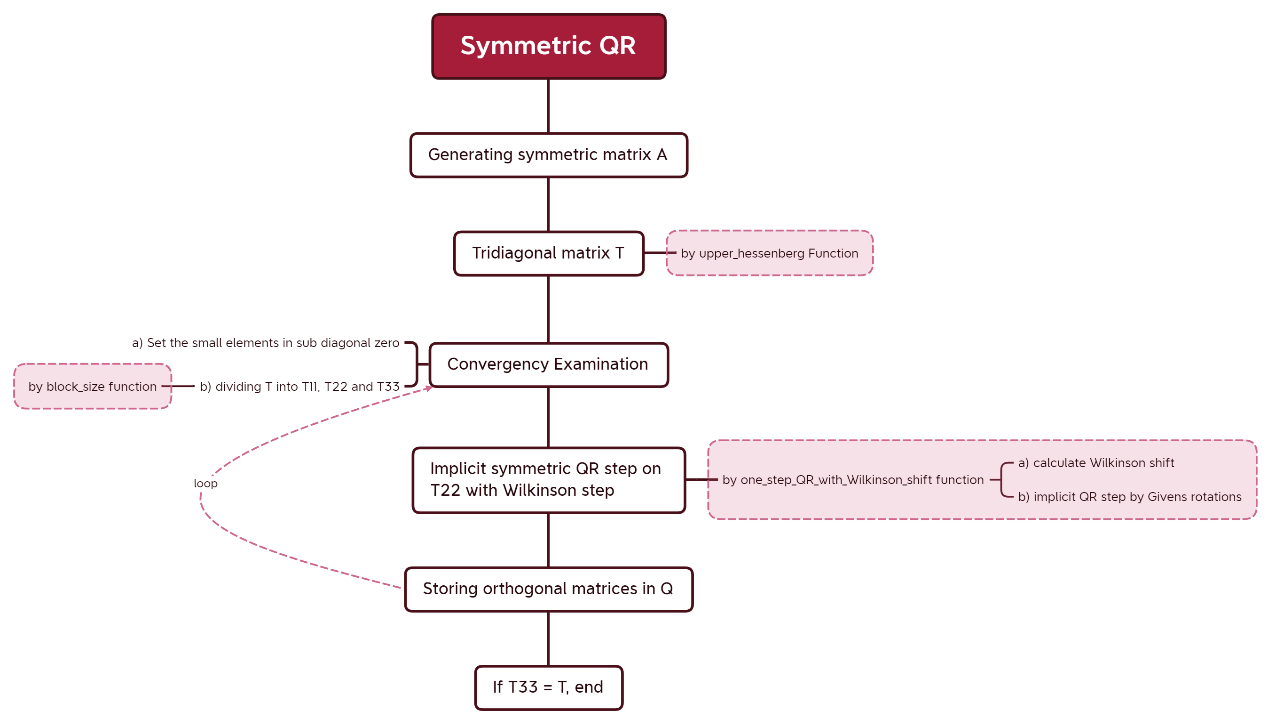
After obtaining T, we have entered the critical part of codes. Before iteration, the matrix T is blocked, which means its convergency is examined. If the elements in the sub diagonal is small enough (the criteria on “small enough” will be omitted here), the algorithm will set them zero. And then, the function ***block\_size*** will divide T into three submatrices, namely, T11, T22, T33. T33 consists of all the eigenvalues we have obtained and will not be involved in the iteration afterwards. T22 is the largest tridiagonal matrix without any zero in its sub diagonal. That is because, when no zero appears in the sub diagonal, the convergency is insured. T11 is the remaining part. When T33 is n by n, the algorithm will stop since all eigenvalues have been obtained.

After blocking the matrix T, only the T22 will be iterated, which invokes the function ***one\_step\_QR\_with\_Wilkinson\_shift***. It first calculates the famous Wilkinson shift and apply the implicit iteration symmetric QR step to T22. Every step of the implicit iteration is a simply Givens rotation doing the bubble-chasing job, but all the Givens rotations add up to a virtually QR step which tries to bring the eigenvalues of T22 out. Now we get back to the convergency examining step to see whether any new eigenvalue is found.

During all the steps above, all the orthogonal matrices are stored in ***Q***, which will show the corresponding eigenvectors if the user need them.

The last step visualizes the convergency history as well as the average iteration steps for an eigenvalue, and then calculate the error by 2-norm of (Q\*T\*Q'-A), where T is now a diagonal matrix with all the eigenvalues.

For clarity, I draw a picture to show the algorithm. Every block with red background is separately stored in a MATLAB file.



**Picture 1: the big picture of my symmetric QR algorithm**

## Detailed algorithm

**First, the *upper\_Hessenberg* function is as follows.**

%% the function "upper\_hessenberg" using householder orthogonalization to make matrix A an upper Hessenberg matrix

% 最后返回A，U，满足：H=U'AU，其中H是上海森伯格阵

function [A,U] = upper\_hessenberg(A)

n = size(A);

n = n(1,1);

U = eye(n);

for i = 1:n-2

u = A(i+1:n,i);

u(1,1) = u(1,1) + sign(u(1,1)) \* norm(u);

if norm(u)==0

continue

end

w = [zeros(i,1);u];

Q = eye(n)-(2/(w'\*w))\*(w\*w'); %generate the Householder orthogonalizer

A = Q\*A\*Q';

U = U\*Q';

end

end

**Secondly, the *block\_size* function is as follow.**

function [l,m] = block\_size(T)

n = size(T);

n = n(1,1);

% 分块【分块部分有bug】

% 首先确定m

flg = 0;

for i = n-1:-1:1

if T(i+1,i) ~= 0

flg = i; %flg所指的是T22的最右侧元素

break

end

end

if flg ~=0

m = n-flg-1;

%only for test

else

m = n;

%only for test

% 注意：此时已经变成对角阵，应当结束循环，跳出循环输出有关信息

end

% 确定l【这一步有bug？】

flg = 0;

for j = n-m-1:-1:1

% 这一步是在确定m后且没有结束循环的情况下进行的，也就是仍有一个三对角阵T22

% 此时n-m所指向的就是三对角阵最右侧的位置

if T(j+1,j) == 0

flg = j; % 此时j所指的是T11最右侧的位置

break;

end

end

if flg == 0

% 此时从1到i+1都是三对角阵的范围

l = 0;

else

% 此时从1到j都是T11的部分

l = flg;

end

end

**Thirdly, the *one\_step\_QR\_with\_Wilkinson\_shift* function is as follows.**

%% 带Wilkinson位移的隐式QR迭代

function [A,G\_accumulated] = one\_step\_QR\_with\_Wilkinson\_shift(A)

% 返回A与G\_accumulated，满足关系：A(output) = G\_accumulated' \* A(input) \* G\_accumulated

n = size(A);

n = n(1,1);

sigma = (A(n-1,n-1)-A(n,n))/2;

Wilkinson\_shift = A(n,n) - (A(n,n-1)^2) / (sigma + sign(sigma) \* sqrt(sigma^2 + A(n,n-1)^2));

x = A(1,1)-Wilkinson\_shift;

% x = A(1,1)-A(n,n);

y = A(2,1);

G\_accumulated = eye(n);

for k = 1:n-1

%生成Givens旋转中的角度

%[c,s] = givens(x,y);

if y == 0

c = 1; s = 0;

else

if abs(y)>abs(x)

tao = x/y; s = 1/sqrt(1+tao^2); c = s\*tao;

else

tao = y/x; c=1/sqrt(1+tao^2); s=c\*tao;

end

end

%生成Givens旋转矩阵

G = eye(n); G(k,k) = c; G(k+1,k+1) = c; G(k,k+1) = -s; G(k+1,k) = s;

%驱赶"气泡"

A = (G'\*A)\*G;

G\_accumulated = G\_accumulated \* G;

%为下一次迭代的Givens矩阵所需参数做准备

if k<n-1

x = A(k+1,k);

y = A(k+2,k);

end

end

end

**Most importantly, the main function called *QR\_Algorithm\_for\_symmatric\_matrix*, is as follows.**

%% QR\_Algorithm\_for\_symmatric\_matrix

%% 准备工作：生成对称矩阵A

n = 20; %n为矩阵阶数

A = zeros(n);

for i = 1:n

for j = 1:i

A(j,i)=-2+4\*rand();

A(i,j) = A(j,i);

end

end

% M用于记录T33阶数的变化（即已经算出的特征值的个数）

M=zeros(n,1);

%% 准备工作：三对角化（调用自己写的函数）

% T与U满足：T=U'AU

[T,U] = upper\_hessenberg(A); %对A进行上海森伯格化，由于A是对称的，故实际上为三对角化

%% QR算法的主体部分

tol = 1e-14;

Q = eye(n); %用于储存所有的Givens旋转阵

max = 1000; %迭代次数保护

m=0; % m是T33阶数，即已经算出的特征值的个数

for count = 1:max

% 一、收敛性判定

% 把足够小的次对角元变成0

for i = 1:n-m-1

if abs(T(i+1,i))<=(abs(T(i,i))+abs(T(i+1,i+1)))\*tol

T(i+1,i) = 0;

T(i,i+1) = 0;

end

end

% 二、对T进行分块（调用自己写的函数）

% 从左上角到右下角分别记作T11，T22，T33，其中T33是对角阵，T22是最大的不可约三对角阵

% 分块的目的以及细节可参见北大教材《数值线性代数》对称QR算法有关章节

[l,m] = block\_size(T); %l是T11阶数，m是T33阶数

M(count,1) = m; %记录已经得到的特征值的个数

if m==n % 此时已经是对角阵，得到全部特征值

break;

end

% 三、对T22进行一次Wilkinson位移隐式QR迭代（调用自己写的函数）

[T(l+1:n-m,l+1:n-m),G] = one\_step\_QR\_with\_Wilkinson\_shift(T(l+1:n-m,l+1:n-m));

% 四、储存旋转阵

Q = Q\*blkdiag(eye(l),G,eye(m)); %Q用于存储Givens旋转阵

end %最外层循环结束（有迭代次数保护的那个）

Q = U\*Q; %把三对角化时的U也包含进来，此处的Q满足 A=QDQ',其中D=T为对角阵，元素为特征值，得到A的谱分解

%% 对称QR算法成果展示

plot([1:count],M,"-o"); %迭代次数和T33（已经算出的特征值）大小的关系

xlabel("rounds of iteration")

ylabel("the number of obtained eigenvalue")

title({['QR algorithm for symmatric matrix'];['matrix size:',num2str(n),' by ',num2str(n)];['tol = ',num2str(tol)]});

disp(["Average iter for an eigenvalue:",count/n])

disp(["Depature:",norm(Q\*T\*Q'-A)])

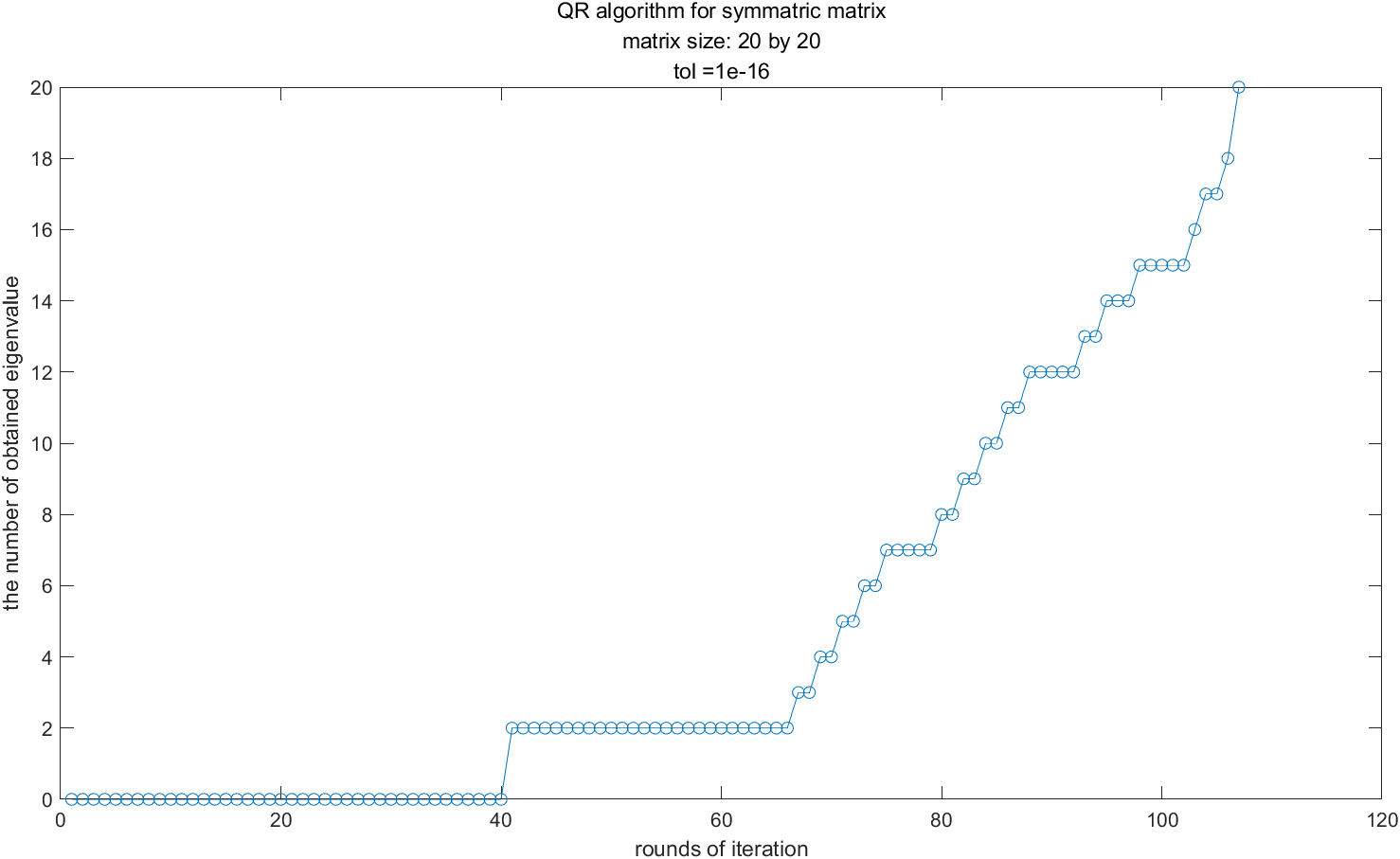
**The details of functions above can be found in the MATLAB files of the same package where this report belongs!**

## Numerical experiments

**Set the size of A 20 by 20.**

Firstly, set the **tol = 1e-16**. It is a strict tolerance, considering that only when **abs(T(i+1,i))<=(abs(T(i,i))+abs(T(i+1,i+1)))\*tol** can the elements in sub diagonal be set zero.

Now run the algorithm and we get the following result.



**Picture 2: the typical convergence history when the size of A is 20 by 20 and tol = 1e-16**

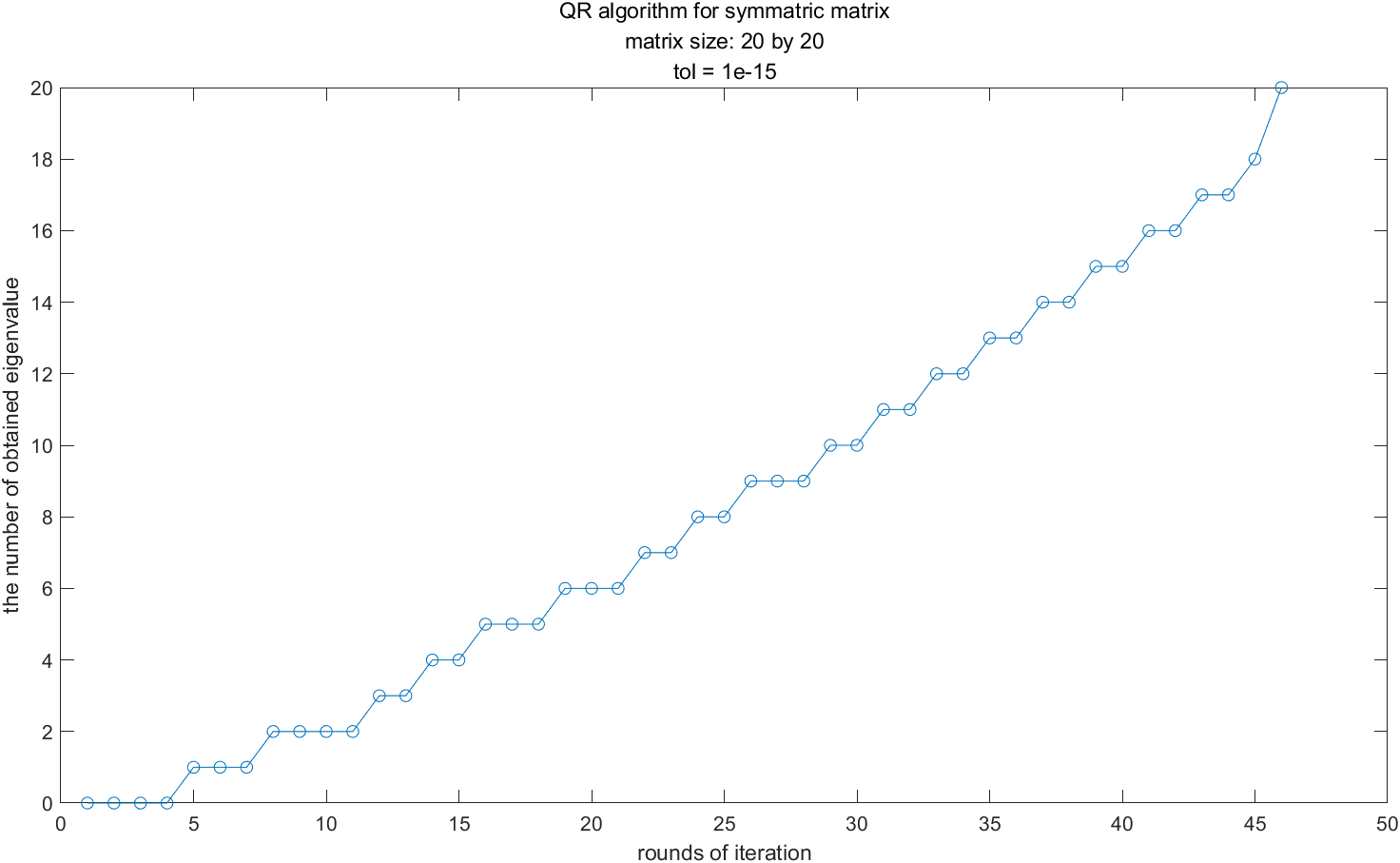
The **average iteration** for an eigenvalue is **5.35**(it costs too much). And the **error**, i.e., 2-norm of (Q\*T\*Q'-A), is **8.7322e-14**.

The matrix A in this experiment can be observed in the file “A of size 20 by 20 in experiment 1”.

After many experiments with tol = 1e-16, it is certain that the convergence history is the **typical case**. Therefore, more results will not be presented here.

Now set **tol = 1e-15** and here are the **different** results.

A **good** case with tol = 1e-15 looks like this:

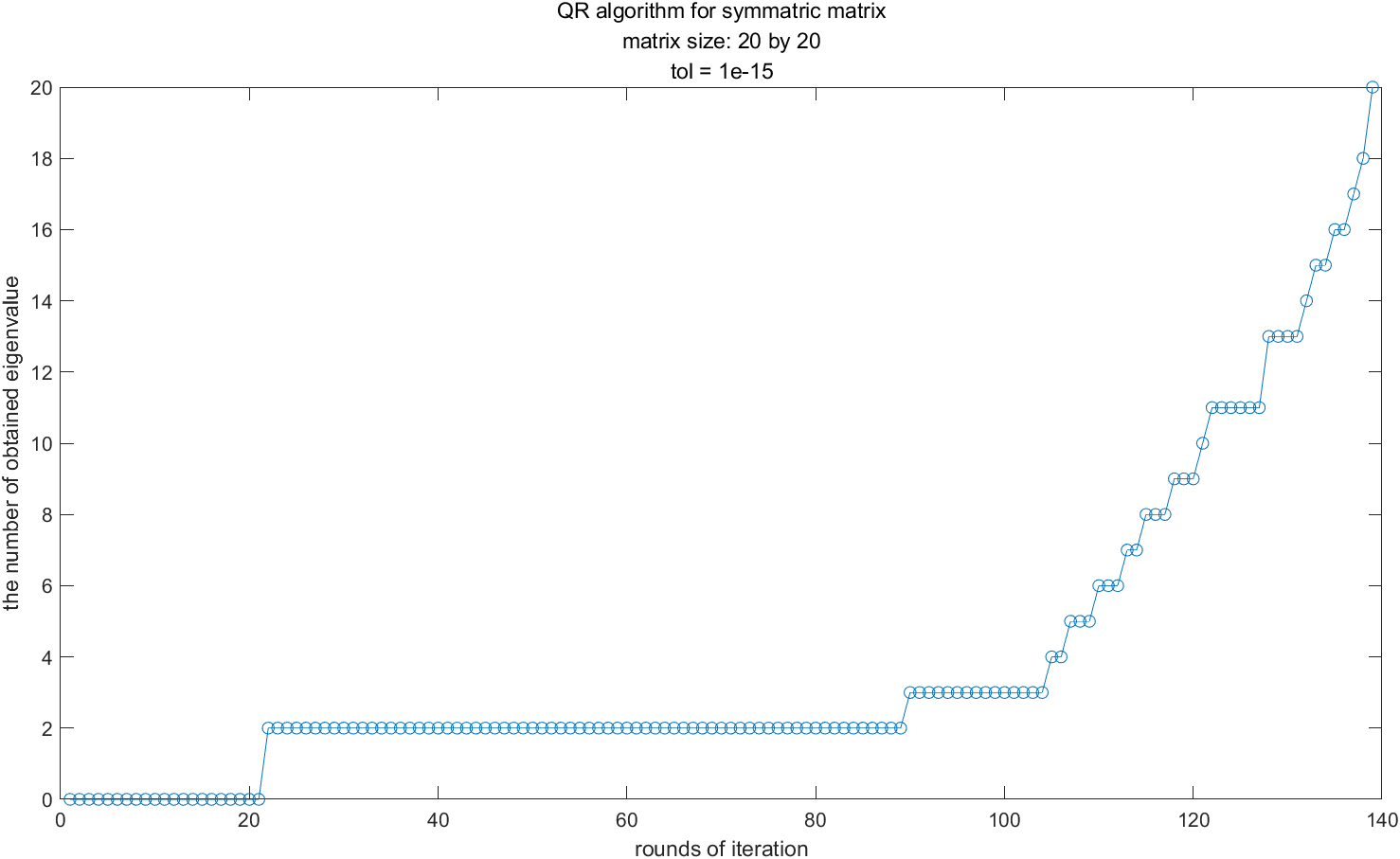


**Picture 3: the good convergence history when the size of A is 20 by 20 and tol = 1e-15**

The **average iteration** for an eigenvalue is **2.3** (much cheaper). And the **error**, i.e., 2-norm of (Q\*T\*Q'-A), is **4.1854e-14** (still as good as the last experiment).

The matrix A in this experiment can be observed in the file “A of size 20 by 20 in experiment 2”.

Nevertheless, here comes a **bad case** with tol = 1e-15:

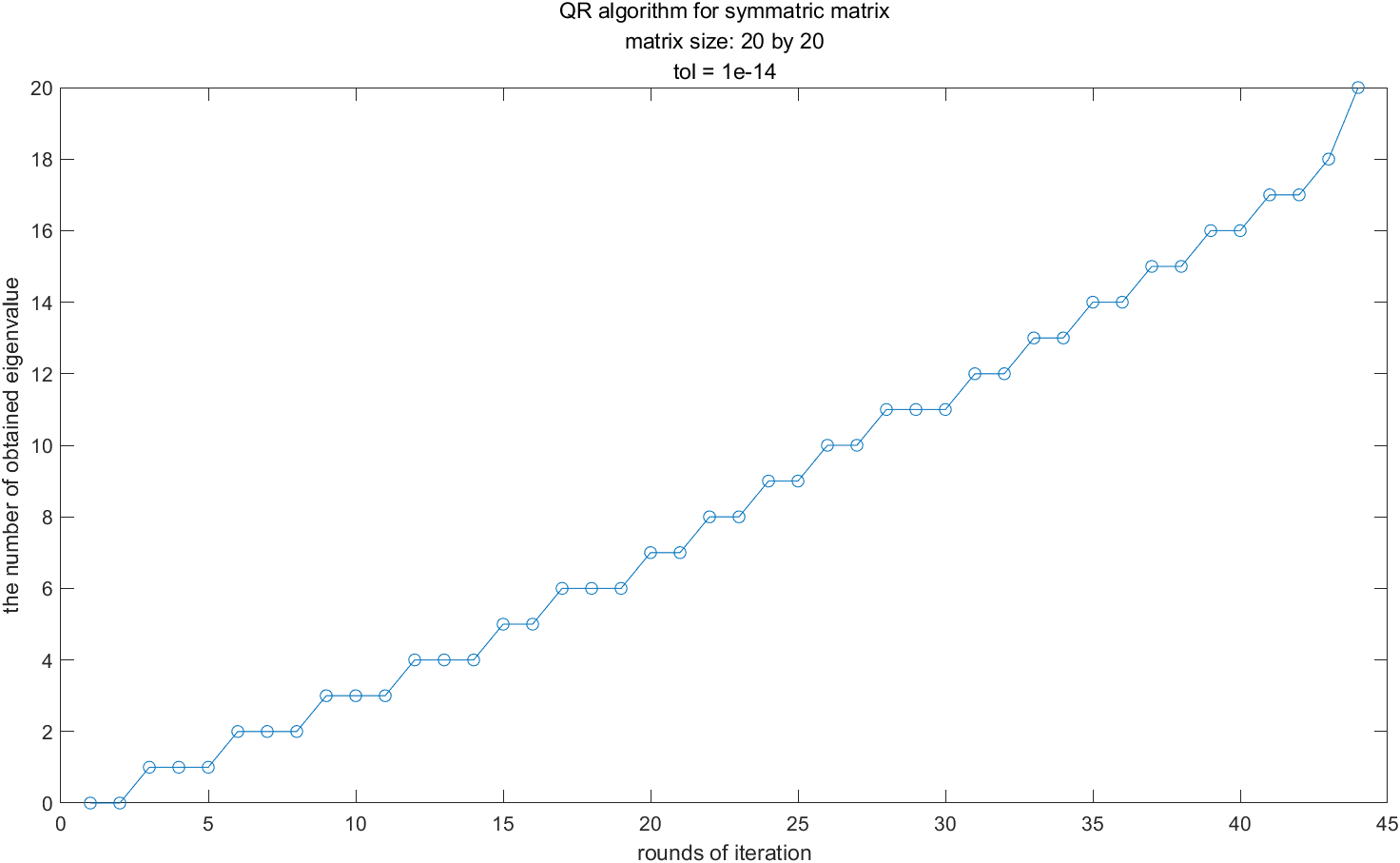


**Picture 4: the bad convergence history when the size of A is 20 by 20 and tol = 1e-15**

The **average iteration** for an eigenvalue is **6.95** (unbearable). And the **error**, i.e., 2-norm of (Q\*T\*Q'-A), is **2.1044e-13** (not that good).

The matrix A in this experiment can be observed in the file “A of size 20 by 20 in experiment 3”.

Now set **tol = 1e-14**. This “tol” is too tolerant, or maybe too “benevolent” to my computer. The result is as follows.



**Picture 5: the typical convergence history when the size of A is 20 by 20 and tol = 1e-14**

The **average iteration** for an eigenvalue is **2.2**(good). And the **error**, i.e., 2-norm of (Q\*T\*Q'-A), is **2.8603e-14** (God, still as good).

The matrix A in this experiment can be observed in the file “A of size 20 by 20 in experiment 4”.

When tol = 1e-14, the convergence history above is typical and thus I shall not present more results here. I have observed that when tol = 1e-14, the average iteration is usually cheap (a little more than 2) and the error (i.e., 2-norm of (Q\*T\*Q'-A)), is as good as the error when tol = 1e-16.

## Discussion

The discussion will focus on tol that determines whether the elements in the sub diagonal is zero and that strongly influence the convergence history.

According to James W. Demmel’s famous textbook[[1]](#footnote-1), the number of average iterations, namely, the QR steps for an eigenvalue, is usually **2**. Nonetheless, according to Golub and Van Loan’s famous work[[2]](#footnote-2), the structure of my algorithm is similar to the pseudocodes in their book except that I separately write important functions like ***upper\_hessenberg*** as a black box, mainly for clarity. The only tricky part is that the tol is left for me to devise.

When tol = 1e-16 or 1e-15, it seems too strict to my computer, since the errors (i.e., 2-norm of (Q\*T\*Q'-A)) are no better than the result when tol = 1e = 14 and it costs too many QR step averagely. Correspondingly, in the picture 2 and picture 4 are some “plat stages” where no eigenvalue could be found.

We can have a closer observation now. Considering the critical codes below that strongly influence the convergency examination:

% 一、收敛性判定

% 把足够小的次对角元变成0

for i = 1:n-m-1

if abs(T(i+1,i))<=(abs(T(i,i))+abs(T(i+1,i+1)))\*tol

T(i+1,i) = 0;

T(i,i+1) = 0;

end

end

**……**

% 三、对T22进行一次Wilkinson位移隐式QR迭代（调用自己写的函数）

[T(l+1:n-m,l+1:n-m),G] =

one\_step\_QR\_with\_Wilkinson\_shift(T(l+1:n-m,l+1:n-m));

When tol is too strict (i.e., 1e-15 or less), even T22 has converged after 2 QR steps, the machine error makes the elements in the sub diagonal of T22 still too large such that they cannot be set zero. That means there would be many useless “plat stages”. In other word, within the “plat stages”, the algorithm cannot make those elements in the sub diagonal smaller when they are O(εmachine), even if T22 has virtually converged. When we are more “tolerant”, or probably “benevolent”, setting tol = 1e-14, the algorithm can **set** **the elements small enough in the sub diagonal zero in time** **without losing accuracy**, compared with the case when tol = 1e-16. As far as I am concerned, a reasonable assumption is that unfortunately the machine error of my computer is only around 1e-14.

Another reason (however, far less possible) is that the QR step might makes those elements in the top left or in the middle but not in the bottom right, converge toward zero. Since only T33 (on the tail) is responsible for storing all the eigenvalues obtained, and since the T22 is the largest tridiagonal sub matrix of T, the algorithm can help nothing but iterate once again. In this case, even some eigenvalues not in the bottom right corner of T22 has been found, the size of T33 doesn’t change and we will see the “plat stages” in the convergence history picture.

## Reference

[1]（美）德梅尔：《应用数值线性代数》，王国荣译，北京：人民邮电出版社，2007.6

[2] Golub G H, Van Loan C F. Matrix Computation. 4th edition. Baltimore: The Johns Hopkins University Press, 2013

[3] 徐树方，高立，张平文：《数值线性代数》，第二版，北京：北京大学出版社，2013.1

1. （美）德梅尔：《应用数值线性代数》，王国荣译，北京：人民邮电出版社，2007.6，第179页 [↑](#footnote-ref-1)
2. Gene H. Golub and Charles F. Van Loan: *Matrix Computation*, 4th edition, the Johns Hopkins University Press, 2013, P463 [↑](#footnote-ref-2)