QR ALGORITHM

The best algorithm for eigenvalue computation

The QR Algorithm

The QR algorithm is one of the most important algorithm for eigenvalue computation.

It is generally quite useful, but has a major limitation, it can only be applied to dense matrices because to compute the algorithm in a faster way we need to use a similarity transformation that tends to fill in the sparse matrix entries, making it dense and so useless, so we cannot operate on really big matrices.

The QR Algorithm

The computation of the algorithm consist of 2 separate stages:

- The first step is a similarity transformation into an Hessenberg Form from the original matrix (so that we preserve the eigenvalue).
- The second consist in a sequence of QR iterations applied to the Hessenberg matrix.

The QR Algorithm – Use cases

We have already said that we can use the algorithm to compute eigenvalue, but generally this algorithm is not used to compute huge dense eigenvalue problem due to the relatively high computational cost.

The majority of QR use comes from its application in combination with other algorithm to solve smaller “internal” auxiliary eigenvalue problems

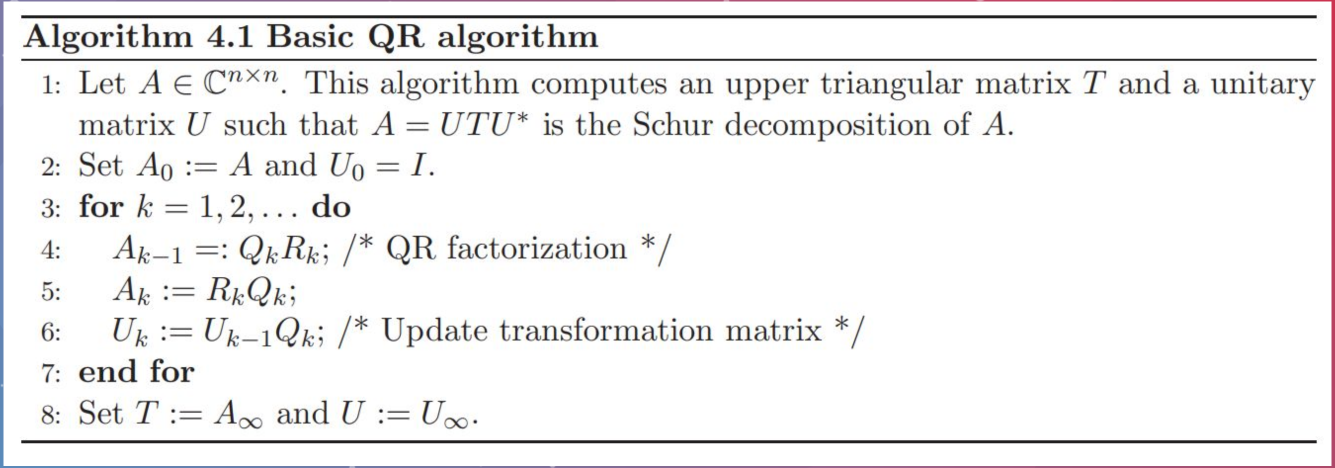
The Basic QR

The first iteration of the QR algorithm comes from the intuition of Francis, who expanded on the ideas of Rutishauser.

The latter was using a the LR factorization, but it was unstable without pivoting, so the QR factorization ultimately was better.

To better understand the QR that is used today we will see various level of optimization, starting with its basic form.

The basic QR



Algorithm 4.1 Basic QR algorithm

```
1: Let $A \in \mathbb{C}^{n\times n}$. This algorithm computes an upper triangular matrix $T$ and a unitary
   matrix $U$ such that $A = UTU^*$ is the Schur decomposition of $A$.
2: Set $A_0 := A$ and $U_0 = I$.
3: for $k = 1, 2, \ldots$ do
4:     $A_{k-1} =: Q_k R_k$; /* QR factorization */
5:     $A_k := R_k Q_k$;
6:     $U_k := U_{k-1} Q_k$; /* Update transformation matrix */
7: end for
8: Set $T := A_\infty$ and $U := U_\infty$.
```

We can see that in this implementation the computation consists of just a sequence of multiplication using the QR factorization, without any similarity transformation.

The Basic QR - In depth

To understand the logic we first notice that:

$$A_k = R_k Q_k = Q_k^* A_{k-1} Q_k,$$

Hence A_k and A_{k-1} are unitarily similar, now if we assume that the sequence $\{A_k\}$, under certain assumption does converge to an upper triangular, and that the eigenvalues are mutually different in magnitude so that we can number them as follows:

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$$

Then the elements below the diagonal will converge to zero like:

$$|a_{ij}^{(k)}| = \mathcal{O}(|\lambda_i/\lambda_j|^k), \qquad i > j.$$

The Basic QR - In depth

Now from the previous pseudocode we see that:

$$A_k = Q_k^* A_{k-1} Q_k = Q_k^* Q_{k-1}^* A_{k-2} Q_{k-1} Q_k = \cdots = Q_k^* \cdots Q_1^* A_0 \underbrace{Q_1 \cdots Q_k}_{U_k}.$$

With the previous assumption we see that $\{A_k\}$ converges to an upper triangular matrix, and U_k converges to the matrix of Schur vector.

The Basic QR - Consideration

After analyzing this first iteration of the algorithm we can draw the following conclusion:

- The algorithm converges very slowly, in fact it can be arbitrarily slow if eigenvalues are very close to each other.

- It is very expensive, because each iteration requires the computation of the QR factorization of a full nxn matrix, so it has cost $O(n^3)$, this must be combined with the various iteration for convergence, even if we assume them to be n the total cost would be $O(n^4)$

Hessenberg QR Algorithm - Improving the cost

The Hessenberg form is a matrix structure which is close to upper triangular that is preserved by the QR algorithm.

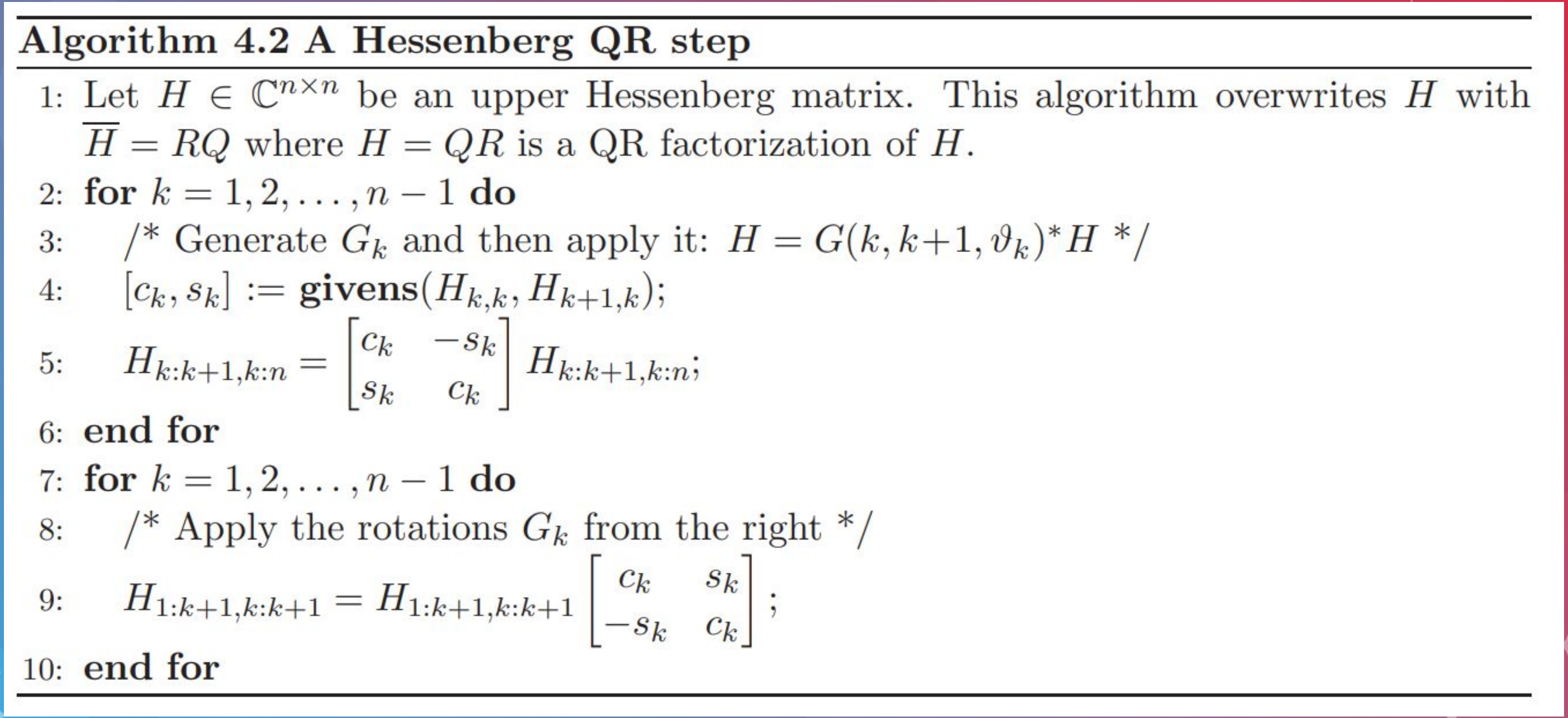
By definition a matrix H is a Hessenberg matrix if its elements below the lower off-diagonal are zero,

$$h_{ij} = 0, \qquad i > j + 1.$$

Knowing this we can try to use an Hessenberg matrix in combination with Givens Rotation to compute the QR algorithm more efficiently.

Hessenberg QR Algorithm

So, to compute a single Hessenberg QR step, we can use the following pseudocode:



Algorithm 4.2 A Hessenberg QR step

```
1: Let $H \in \mathbb{C}^{n \times n}$ be an upper Hessenberg matrix. This algorithm overwrites $H$ with
   $\overline{H} = RQ$ where $H = QR$ is a QR factorization of $H$.
2: for $k = 1, 2, \ldots, n-1$ do
3:     /* Generate $G_k$ and then apply it: $H = G(k, k+1, \vartheta_k)^* H$ */
4:     $[c_k, s_k] :=$ givens$(H_{k,k}, H_{k+1,k})$;
5:     $H_{k:k+1,k:n} = \begin{bmatrix} c_k & -s_k \\ s_k & c_k \end{bmatrix} H_{k:k+1,k:n}$;
6: end for
7: for $k = 1, 2, \ldots, n-1$ do
8:     /* Apply the rotations $G_k$ from the right */
9:     $H_{1:k+1,k:k+1} = H_{1:k+1,k:k+1} \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}$;
10: end for
```

Hessenberg QR Algorithm - Cost

Neglecting the determination of the parameter for the Givens rotation, then each of the for loops requires

$$\sum_{i=1}^{n-1} 6i = 6\frac{n(n-1)}{2} \approx 3n^2$$

We can optionally add the computation of U with the following code.

```
1: for k=1,2,...,n-1 do
2:     $U_{1:n,k:k+1} = U_{1:n,k:k+1} \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}$;
3: end for
```

This has the same cost of the previous 2 loops.

Hessenberg QR Algorithm - Cost

Putting all costs together we can see that the total cost of the Hessenberg QR Algorithm is:

$12n^2$ compared to the $7/3\ n^3$

This means we have gained a factor of O(n) changing the input from a dense matrix to a Hessenberg matrix.

This is a step in the right direction, but there is still the problem of slow convergence when eigenvalue are really close in value.

Hessenberg QR Algorithm - Householder Reflector

Up until now we used Givens rotation to convert a matrix into an upper Hessenberg form, but there is a more efficient way to do it.

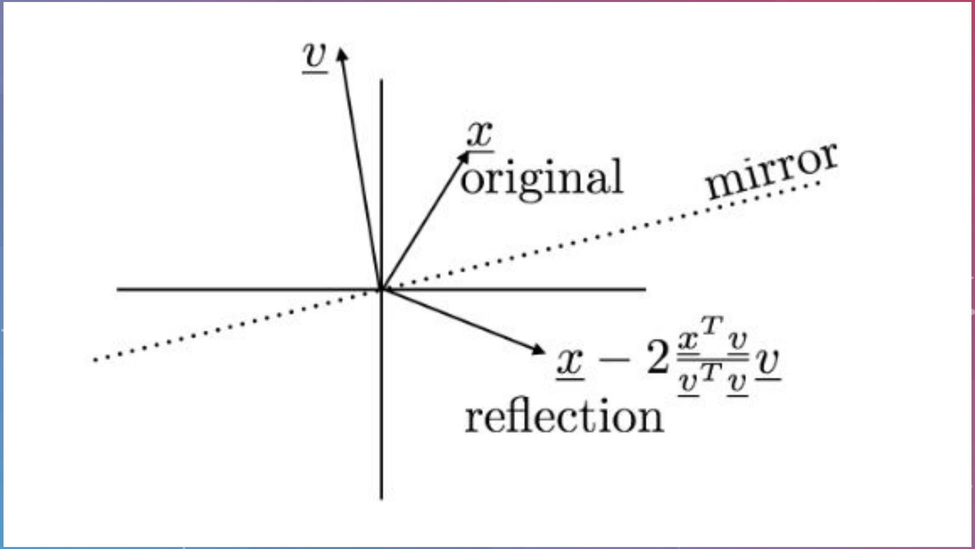
We can use the Householder reflectors to annihilate more then one matrix entry at a time (given's could do only one at a time).

The Householder Reflector

The idea behind the reflector is the following:

We want to find an orthogonal matrix P that, given a column vector z, puts to zero all the element of z that are below a certain row.

P must be chosen in such a way that it reflects the vector through a mirror



$\underline{v}$
$\underline{x}$
original
mirror
$\underline{x} - 2\frac{\underline{x}^T\underline{v}}{\underline{v}^T\underline{v}}\underline{v}$
reflection

The Householder Reflector

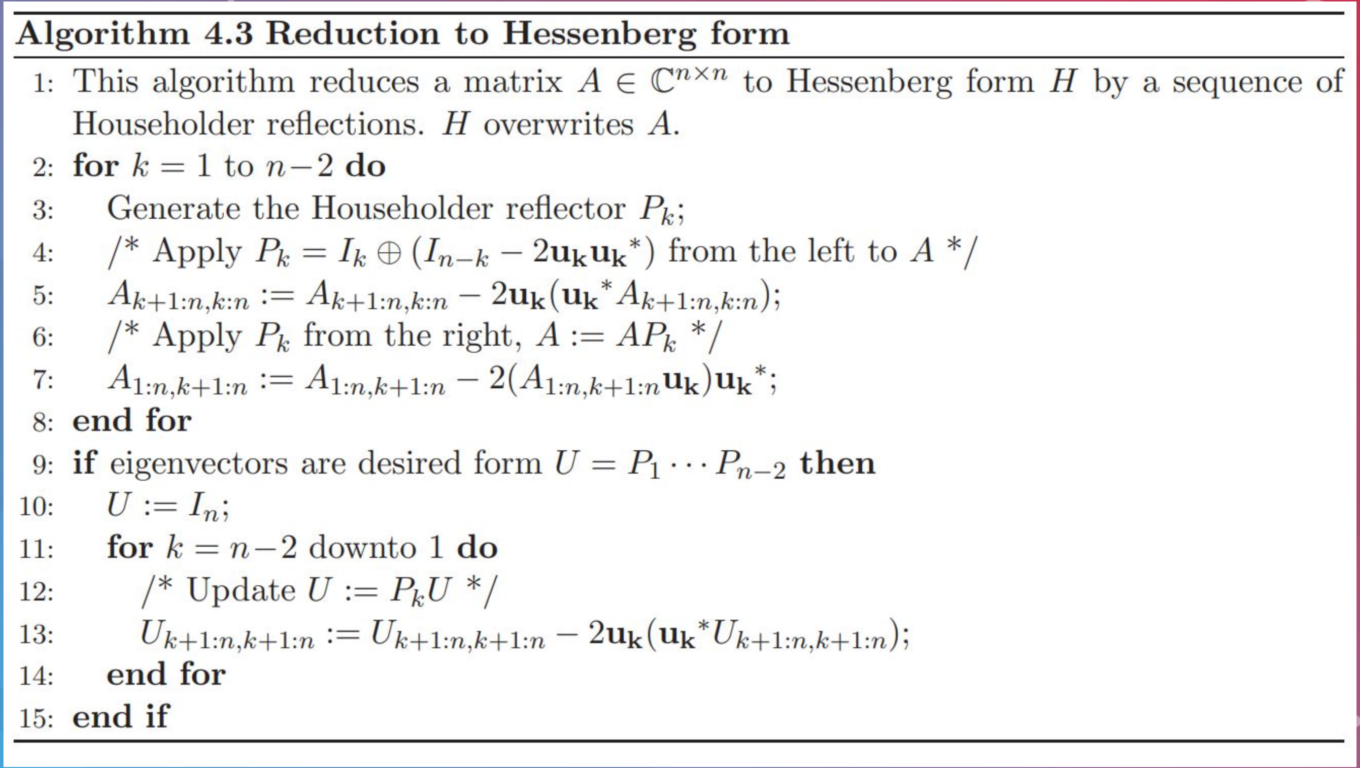
We see from the previous image that we can use the vector v to compute the direction and the norm of P.

This will make us able to compute the householder reflector with the following formula:

$$\underline{x} - 2\frac{\underline{v}^T\,\underline{x}}{\underline{v}^T\,\underline{v}}\underline{v} = \underbrace{\left(I - 2\frac{\underline{v}\,\underline{v}^T}{\underline{v}^T\,\underline{v}}\right)}_{\text{Householder relfection}}\,\underline{x},$$

Hessenberg QR Algorithm – Householder Pseudocode

Having said that we can write the pseudocode for the QR Algorithm using Householder Reflector.



Algorithm 4.3 **Reduction to Hessenberg form**

```
This algorithm reduces a matrix $A \in \mathbb{C}^{n\times n}$ to Hessenberg form $H$ by a sequence of Householder reflections. $H$ overwrites $A$.
for $k = 1$ to $n-2$ do
    Generate the Householder reflector $P_k$;
    /* Apply $P_k = I_k \oplus (I_{n-k} - 2\mathbf{u_k}\mathbf{u_k}^*)$ from the left to $A$ */
    $A_{k+1:n,k:n} := A_{k+1:n,k:n} - 2\mathbf{u_k}(\mathbf{u_k}^* A_{k+1:n,k:n})$;
    /* Apply $P_k$ from the right, $A := AP_k$ */
    $A_{1:n,k+1:n} := A_{1:n,k+1:n} - 2(A_{1:n,k+1:n}\mathbf{u_k})\mathbf{u_k}^*$;
end for
if eigenvectors are desired form $U = P_1 \cdots P_{n-2}$ then
    $U := I_n$;
    for $k = n-2$ downto $1$ do
        /* Update $U := P_k U$ */
        $U_{k+1:n,k+1:n} := U_{k+1:n,k+1:n} - 2\mathbf{u_k}(\mathbf{u_k}^* U_{k+1:n,k+1:n})$;
    end for
end if
```

Hessenberg QR Algorithm - Householder Cost

The total cost for the hessenberg reduction using this technique is :

$$14/3\ n^3$$

Adding this value to the cost of the QR step previously analyzed we get the following cost:

$$14/3\ n^3 + 12\ n^3$$

Which is still an improvement over the basic QR iteration.

Shifted QR - Improving the convergence

Up until now we have reduced the cost of the QR algorithm, now we will see that it is also possible to dramatically improve the convergence time, and to do so we will need to introduce **shifts** to the algorithm, with a subsequent deflation step.

But what is a shift?

A shift is a scalar mu that allows us to compute the QR factorization of :

H -mu*I = QR

instead of H, this will result in the following QR step, followed by a deflation step.

$$
\begin{aligned}
&H_k - \mu_k I = Q_k R_k \ \text{(QR Factorization)} \\
&H_{k+1} = R_k Q_k + \mu_k I
\end{aligned}
$$

Shifted QR Algorithm

We cannot choose a perfect shift mu for the algorithm because we do not know the value of the eigenvalue, so we now introduce an heuristic to estimate it.

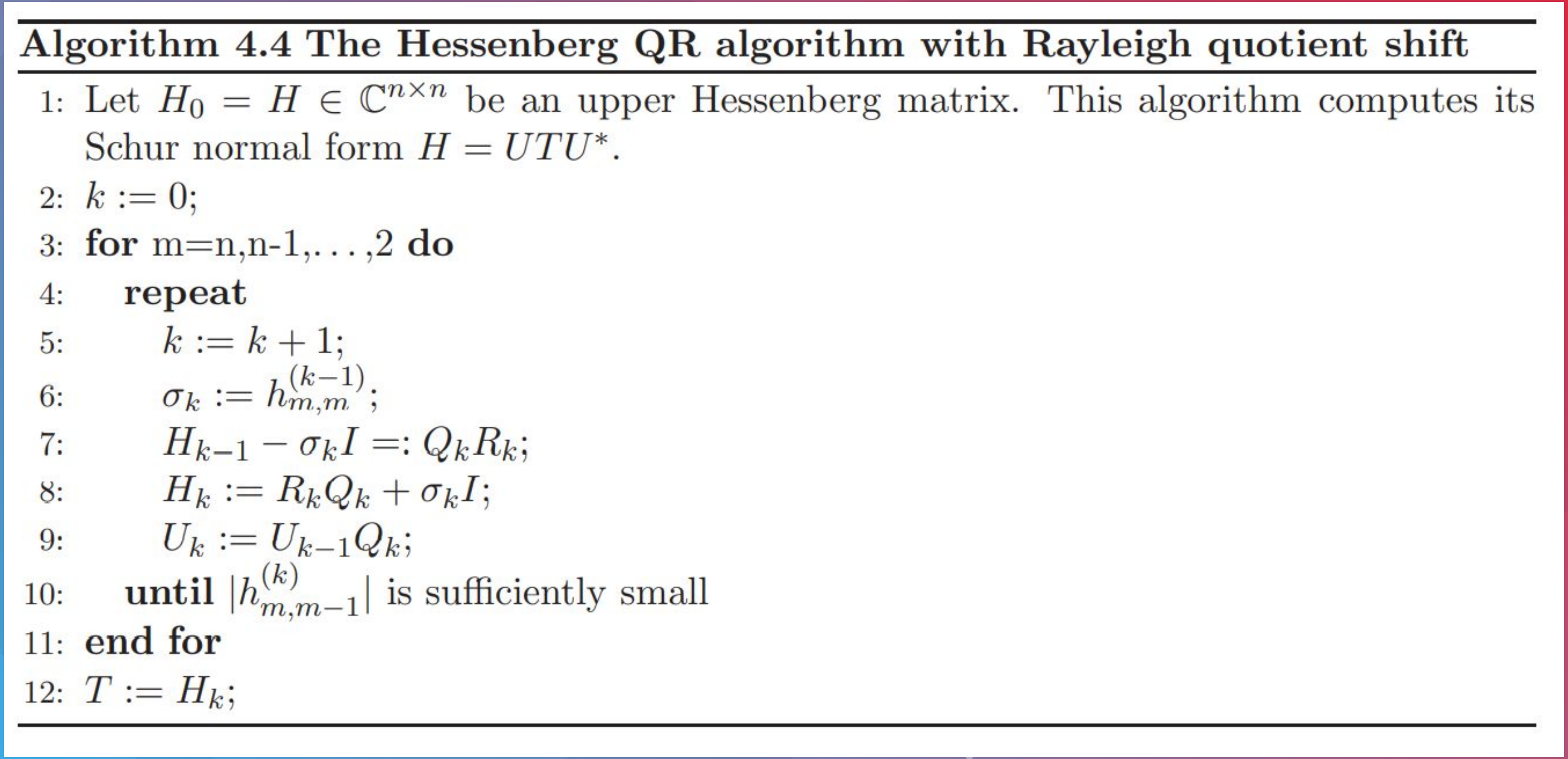
There are many heuristics for this, but we will use the **Rayleigh quotient shift**.

So we have that the shift in k-th step is equal to the last diagonal element:

$$\sigma_k := h_{n,n}^{(k-1)} = \mathbf{e}_n^* H^{(k-1)} \mathbf{e}_n .$$

Shifted QR Algorithm – Pseudocode

We now show the pseudocode for this algorithm.



Algorithm 4.4 **The Hessenberg QR algorithm with Rayleigh quotient shift**

```
1: Let $H_0 = H \in \mathbb{C}^{n\times n}$ be an upper Hessenberg matrix. This algorithm computes its
   Schur normal form $H = UTU^*$.
2: $k := 0$;
3: for m=n,n-1,...,2 do
4:     repeat
5:         $k := k + 1$;
6:         $\sigma_k := h^{(k-1)}_{m,m}$;
7:         $H_{k-1} - \sigma_k I =: Q_k R_k$;
8:         $H_k := R_k Q_k + \sigma_k I$;
9:         $U_k := U_{k-1} Q_k$;
10:    until $|h^{(k)}_{m,m-1}|$ is sufficiently small
11: end for
12: $T := H_k$;
```