Chapter 4

The QR Algorithm

The QR algorithm computes a Schur decomposition of a matrix. It is certainly one of the most important algorithm in eigenvalue computations [9]. However, it is applied to *dense* (or: *full*) matrices only.

The QR algorithm consists of two separate stages. First, by means of a similarity transformation, the original matrix is transformed in a finite number of steps to Hessenberg form or – in the Hermitian/symmetric case – to real tridiagonal form. This first stage of the algorithm prepares its second stage, the actual QR iterations that are applied to the Hessenberg or tridiagonal matrix. The overall complexity (number of floating points) of the algorithm is $\mathcal{O}(n^3)$, which we will see is not entirely trivial to obtain.

The major limitation of the QR algorithm is that already the first stage generates usually complete fill-in in general sparse matrices. It can therefore not be applied to large sparse matrices, simply because of excessive memory requirements. On the other hand, the QR algorithm computes all eigenvalues (and eventually eigenvectors) which is rarely desired in sparse matrix computations anyway.

The treatment of the QR algorithm in these lecture notes on large scale eigenvalue computation is justified in two respects. First, there are of course large or even huge *dense* eigenvalue problems. Second, the QR algorithm is employed in most other algorithms to solve 'internal' small auxiliary eigenvalue problems.

4.1 The basic QR algorithm

In 1958 Rutishauser [10] of ETH Zurich experimented with a similar algorithm that we are going to present, but based on the LR factorization, i.e., based on Gaussian elimination without pivoting. That algorithm was not successful as the LR factorization (nowadays called LU factorization) is not stable without pivoting. Francis [5] noticed that the QR factorization would be the preferred choice and devised the QR algorithm with many of the bells and whistles used nowadays.

Before presenting the complete picture, we start with a basic iteration, given in Algorithm 4.1, discuss its properties and improve on it step by step until we arrive at Francis' algorithm.

We notice first that

$$A_k = R_k Q_k = Q_k^* A_{k-1} Q_k, \tag{4.1}$$

and hence A_k and A_{k-1} are unitarily similar. The matrix sequence $\{A_k\}$ converges (under certain assumptions) towards an upper triangular matrix [11]. Let us assume that the

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: $\quad A_{k-1} =: Q_k R_k$; /* QR factorization */
5: $\quad A_k := R_k Q_k$;
6: $\quad U_k := U_{k-1} Q_k$; /* Update transformation matrix */
7: **end for**
8: Set $T := A_\infty$ and $U := U_\infty$.

eigenvalues are mutually different in magnitude and we can therefore number the eigenvalues such that $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$. Then – as we will show in Chapter 8 – the elements of A_k below the diagonal converge to zero like

$$|a_{ij}^{(k)}| = \mathcal{O}(|\lambda_i/\lambda_j|^k), \qquad i > j. \tag{4.2}$$

From (4.1) we see that

$$A_k = Q_k^* A_{k-1} Q_k = Q_k^* Q_{k-1}^* A_{k-2} Q_{k-1} Q_k = \cdots = Q_k^* \cdots Q_1^* A_0 \underbrace{Q_1 \cdots Q_k}_{U_k}. \tag{4.3}$$

With the same assumption on the eigenvalues, A_k tends to an upper triangular matrix and U_k converges to the matrix of Schur vectors.

4.1.1 Numerical experiments

We conduct two MATLAB experiments to illustrate the convergence rate given in (4.2). To that end, we construct a random 4×4 matrix with eigenvalues 1, 2, 3, and 4.

```
D = diag([4 3 2 1]);
rand('seed',0);
format short e
S=rand(4); S = (S - .5)*2;
A = S*D/S      % A_0 = A = S*D*S^{-1}
for i=1:20,
   [Q,R] = qr(A);  A = R*Q
end
```

This yields the matrix sequence

```
A( 0) =  [ -4.4529e-01   4.9063e+00  -8.7871e-01   6.3036e+00]
         [ -6.3941e+00   1.3354e+01   1.6668e+00   1.1945e+01]
         [  3.6842e+00  -6.6617e+00  -6.0021e-02  -7.0043e+00]
         [  3.1209e+00  -5.2052e+00  -1.4130e+00  -2.8484e+00]

A( 1) =  [  5.9284e+00   1.6107e+00   9.3153e-01  -2.2056e+01]
         [ -1.5294e+00   1.8630e+00   2.0428e+00   6.5900e+00]
         [  1.9850e-01   2.5660e-01   1.7088e+00   1.2184e+00]
         [  2.4815e-01   1.5265e-01   2.6924e-01   4.9975e-01]

A( 2) =  [  4.7396e+00   1.4907e+00  -2.1236e+00   2.3126e+01]
```

```
          [ -4.3101e-01   2.4307e+00   2.2544e+00  -8.2867e-01]
          [  1.2803e-01   2.4287e-01   1.6398e+00  -1.8290e+00]
          [ -4.8467e-02  -5.8164e-02  -1.0994e-01   1.1899e+00]

A( 3) =   [  4.3289e+00   1.0890e+00  -3.9478e+00  -2.2903e+01]
          [ -1.8396e-01   2.7053e+00   1.9060e+00  -1.2062e+00]
          [  6.7951e-02   1.7100e-01   1.6852e+00   2.5267e+00]
          [  1.3063e-02   2.2630e-02   7.9186e-02   1.2805e+00]

A( 4) =   [  4.1561e+00   7.6418e-01  -5.1996e+00   2.2582e+01]
          [ -9.4175e-02   2.8361e+00   1.5788e+00   2.0983e+00]
          [  3.5094e-02   1.1515e-01   1.7894e+00  -2.9819e+00]
          [ -3.6770e-03  -8.7212e-03  -5.7793e-02   1.2184e+00]

A( 5) =   [  4.0763e+00   5.2922e-01  -6.0126e+00  -2.2323e+01]
          [ -5.3950e-02   2.9035e+00   1.3379e+00  -2.5358e+00]
          [  1.7929e-02   7.7393e-02   1.8830e+00   3.2484e+00]
          [  1.0063e-03   3.2290e-03   3.7175e-02   1.1372e+00]

A( 6) =   [  4.0378e+00   3.6496e-01  -6.4924e+00   2.2149e+01]
          [ -3.3454e-02   2.9408e+00   1.1769e+00   2.7694e+00]
          [  9.1029e-03   5.2173e-02   1.9441e+00  -3.4025e+00]
          [ -2.6599e-04  -1.1503e-03  -2.1396e-02   1.0773e+00]

A( 7) =   [  4.0189e+00   2.5201e-01  -6.7556e+00  -2.2045e+01]
          [ -2.1974e-02   2.9627e+00   1.0736e+00  -2.9048e+00]
          [  4.6025e-03   3.5200e-02   1.9773e+00   3.4935e+00]
          [  6.8584e-05   3.9885e-04   1.1481e-02   1.0411e+00]

A( 8) =   [  4.0095e+00   1.7516e-01  -6.8941e+00   2.1985e+01]
          [ -1.5044e-02   2.9761e+00   1.0076e+00   2.9898e+00]
          [  2.3199e-03   2.3720e-02   1.9932e+00  -3.5486e+00]
          [ -1.7427e-05  -1.3602e-04  -5.9304e-03   1.0212e+00]

A( 9) =   [  4.0048e+00   1.2329e-01  -6.9655e+00  -2.1951e+01]
          [ -1.0606e-02   2.9845e+00   9.6487e-01  -3.0469e+00]
          [  1.1666e-03   1.5951e-02   1.9999e+00   3.5827e+00]
          [  4.3933e-06   4.5944e-05   3.0054e-03   1.0108e+00]

A(10) =   [  4.0024e+00   8.8499e-02  -7.0021e+00   2.1931e+01]
          [ -7.6291e-03   2.9899e+00   9.3652e-01   3.0873e+00]
          [  5.8564e-04   1.0704e-02   2.0023e+00  -3.6041e+00]
          [ -1.1030e-06  -1.5433e-05  -1.5097e-03   1.0054e+00]

A(11) =   [  4.0013e+00   6.5271e-02  -7.0210e+00  -2.1920e+01]
          [ -5.5640e-03   2.9933e+00   9.1729e-01  -3.1169e+00]
          [  2.9364e-04   7.1703e-03   2.0027e+00   3.6177e+00]
          [  2.7633e-07   5.1681e-06   7.5547e-04   1.0027e+00]

A(12) =   [  4.0007e+00   4.9824e-02  -7.0308e+00   2.1912e+01]
          [ -4.0958e-03   2.9956e+00   9.0396e-01   3.1390e+00]
          [  1.4710e-04   4.7964e-03   2.0024e+00  -3.6265e+00]
          [ -6.9154e-08  -1.7274e-06  -3.7751e-04   1.0014e+00]

A(13) =   [  4.0003e+00   3.9586e-02  -7.0360e+00  -2.1908e+01]
          [ -3.0339e-03   2.9971e+00   8.9458e-01  -3.1558e+00]
          [  7.3645e-05   3.2052e-03   2.0019e+00   3.6322e+00]
          [  1.7298e-08   5.7677e-07   1.8857e-04   1.0007e+00]

A(14) =   [  4.0002e+00   3.2819e-02  -7.0388e+00   2.1905e+01]
```

```
          [ -2.2566e-03   2.9981e+00   8.8788e-01   3.1686e+00]
          [  3.6855e-05   2.1402e-03   2.0014e+00  -3.6359e+00]
          [ -4.3255e-09  -1.9245e-07  -9.4197e-05   1.0003e+00]

A(15) =   [  4.0001e+00   2.8358e-02  -7.0404e+00  -2.1902e+01]
          [ -1.6832e-03   2.9987e+00   8.8305e-01  -3.1784e+00]
          [  1.8438e-05   1.4284e-03   2.0010e+00   3.6383e+00]
          [  1.0815e-09   6.4192e-08   4.7062e-05   1.0002e+00]

A(16) =   [  4.0001e+00   2.5426e-02  -7.0413e+00   2.1901e+01]
          [ -1.2577e-03   2.9991e+00   8.7953e-01   3.1859e+00]
          [  9.2228e-06   9.5295e-04   2.0007e+00  -3.6399e+00]
          [ -2.7039e-10  -2.1406e-08  -2.3517e-05   1.0001e+00]

A(17) =   [  4.0000e+00   2.3503e-02  -7.0418e+00  -2.1900e+01]
          [ -9.4099e-04   2.9994e+00   8.7697e-01  -3.1917e+00]
          [  4.6126e-06   6.3562e-04   2.0005e+00   3.6409e+00]
          [  6.7600e-11   7.1371e-09   1.1754e-05   1.0000e+00]

A(18) =   [  4.0000e+00   2.2246e-02  -7.0422e+00   2.1899e+01]
          [ -7.0459e-04   2.9996e+00   8.7508e-01   3.1960e+00]
          [  2.3067e-06   4.2388e-04   2.0003e+00  -3.6416e+00]
          [ -1.6900e-11  -2.3794e-09  -5.8750e-06   1.0000e+00]

A(19) =   [  4.0000e+00   2.1427e-02  -7.0424e+00  -2.1898e+01]
          [ -5.2787e-04   2.9997e+00   8.7369e-01  -3.1994e+00]
          [  1.1535e-06   2.8265e-04   2.0002e+00   3.6421e+00]
          [  4.2251e-12   7.9321e-10   2.9369e-06   1.0000e+00]

A(20) =   [  4.0000e+00   2.0896e-02  -7.0425e+00   2.1898e+01]
          [ -3.9562e-04   2.9998e+00   8.7266e-01   3.2019e+00]
          [  5.7679e-07   1.8846e-04   2.0002e+00  -3.6424e+00]
          [ -1.0563e-12  -2.6442e-10  -1.4682e-06   1.0000e+00]
```

Looking at the element-wise quotients of the last two matrices one recognizes the convergence rates claimed in (4.2).

```
A(20)./A(19) =  [  1.0000   0.9752   1.0000  -1.0000]
                [  0.7495   1.0000   0.9988  -1.0008]
                [  0.5000   0.6668   1.0000  -1.0001]
                [ -0.2500  -0.3334  -0.4999   1.0000]
```

The elements above and on the diagonal are relatively stable.

If we run the same little MATLAB script but with the initial diagonal matrix D replaced by

```
D = diag([5 2 2 1]);
```

then we obtain

```
A(19) =   [  5.0000e+00   4.0172e+00  -9.7427e+00  -3.3483e+01]
          [ -4.2800e-08   2.0000e+00   2.1100e-05  -4.3247e+00]
          [  1.3027e-08   7.0605e-08   2.0000e+00   2.1769e+00]
          [  8.0101e-14  -2.4420e-08   4.8467e-06   1.0000e+00]

A(20) =   [  5.0000e+00   4.0172e+00  -9.7427e+00   3.3483e+01]
          [ -1.7120e-08   2.0000e+00   1.0536e-05   4.3247e+00]
          [  5.2106e-09   3.3558e-08   2.0000e+00  -2.1769e+00]
          [ -1.6020e-14   1.2210e-08  -2.4234e-06   1.0000e+00]
```

So, again the eigenvalues are visible on the diagonal of A_{20}. The element-wise quotients of A_{20} relative to A_{19} are

```
A(20)./A(19) =  [  1.0000   1.0000   1.0000  -1.0000]
                [  0.4000   1.0000   0.4993  -1.0000]
                [  0.4000   0.4753   1.0000  -1.0000]
                [ -0.2000  -0.5000  -0.5000   1.0000]
```

Notice that (4.2) does not state a rate for the element at position $(3, 2)$.

These little numerical tests are intended to demonstrate that the convergence rates given in (4.2) are in fact seen in a real run of the basic QR algorithm. The conclusions we can draw are the following:

1. The convergence of the algorithm is slow. In fact it can be arbitrarily slow if eigenvalues are very close to each other.

2. The algorithm is expensive. Each iteration step requires the computation of the QR factorization of a full $n \times n$ matrix, i.e., each single iteration step has a complexity $\mathcal{O}(n^3)$. Even if we assume that the number of steps is proportional to n we would get an $\mathcal{O}(n^4)$ complexity. The latter assumption is not even assured, see point 1 of this discussion.

In the following we want to improve on both issues. First we want to find a matrix structure that is preserved by the QR algorithm and that lowers the cost of a single iteration step. Then, we want to improve on the convergence properties of the algorithm.

4.2 The Hessenberg QR algorithm

A matrix structure that is close to upper triangular form and that is preserved by the QR algorithm is the Hessenberg form.

Definition 4.1 A matrix H is a Hessenberg matrix if its elements below the lower off-diagonal are zero,

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

Theorem 4.2 *The Hessenberg form is preserved by the QR algorithms.*

Proof. We give a constructive proof, i.e., given a Hessenberg matrix H with QR factorization $H = QR$, we show that $\overline{H} = RQ$ is again a Hessenberg matrix.

The **Givens rotation** or **plane rotation** $G(i, j, \vartheta)$ is defined by

$$G(i,j,\vartheta) := \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ \leftarrow i \\ \\ \leftarrow j \\ \\ \\ \end{matrix} \tag{4.4}$$
$$\qquad\qquad\qquad\quad \uparrow \qquad\quad \uparrow$$
$$\qquad\qquad\qquad\quad i \qquad\quad j$$

where $c = \cos(\vartheta)$ and $s = \sin(\vartheta)$. Pre-multiplication by $G(i, j, \vartheta)$ amounts to a counter-clockwise rotation by ϑ radians in the (i, j) coordinate plane. Clearly, a Givens rotation is

an orthogonal matrix. For a unitary version see [4]. If $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} = G(i,j,\vartheta)^*\mathbf{x}$, then

$$y_k = \begin{cases} cx_i - sx_j, & k = i \\ sx_i + cx_j, & k = j \\ x_k, & k \neq i,j \end{cases}$$

We can force y_j to be zero by setting

$$c = \frac{x_i}{\sqrt{|x_i|^2 + |x_j|^2}}, \qquad s = \frac{-x_j}{\sqrt{|x_i|^2 + |x_j|^2}}. \tag{4.5}$$

Thus, it is a simple matter to zero a *single specific* entry in a vector by using a Givens rotation[1].

Now, let us look at a Hessenberg matrix H. We can show the principle procedure by means of a 4×4 example.

$$H = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G(1,2,\vartheta_1)^*\cdot} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}$$

$$\xrightarrow{G(2,3,\vartheta_2)^*\cdot} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G(3,4,\vartheta_3)^*\cdot} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} = R$$

So, with $G_k = G(k, k+1, \vartheta_k)$, we get

$$\underbrace{G_3^* G_2^* G_1^*}_{Q^*} H = R \qquad \Longleftrightarrow \qquad H = QR.$$

Multiplying Q and R in reversed order gives

$$\overline{H} = RQ = RG_1G_2G_3,$$

or, pictorially,

$$R = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{\cdot G(1,2,\vartheta_1)} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix}$$

$$\xrightarrow{\cdot G(2,3,\vartheta_2)} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{\cdot G(3,4,\vartheta_1)} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} = \overline{H}$$

More generally, if H is $n \times n$, $n-1$ Givens rotations $G_1, \ldots, G_{n-1}$ are needed to transform H to upper triangular form. Applying the rotations from the right restores the Hessenberg form. ■

Remark 4.1. The Hessenberg nonzero pattern isn't the only pattern that is preserved by the QR algoritm, see [2], however it is the most simple one. □

[1] For a stable way to compute Givens rotations see Algorithm 5.1.3 in [6].

4.2.1 A numerical experiment

We repeat one of the previous two MATLAB experiments

```
D = diag([4 3 2 1]);
rand('seed',0);
S=rand(4); S = (S - .5)*2;
A = S*D/S      % A_0 = A = S*D*S^{-1}
H = hess(A);  % built-in MATLAB function: generates
              % unitarily similar Hessenberg matrix
for i=1:30,
   [Q,R] = qr(H);  H = R*Q
end
```

This yields the matrix sequence

```
H( 0) =  [ -4.4529e-01  -1.8641e+00  -2.8109e+00   7.2941e+00]
         [  8.0124e+00   6.2898e+00   1.2058e+01  -1.6088e+01]
         [  0.0000e+00   4.0087e-01   1.1545e+00  -3.3722e-01]
         [  0.0000e+00   0.0000e+00  -1.5744e-01   3.0010e+00]

H( 5) =  [  4.0763e+00  -2.7930e+00  -7.1102e+00   2.1826e+01]
         [  5.6860e-02   2.4389e+00  -1.2553e+00  -3.5061e+00]
         [              -2.0209e-01   2.5681e+00  -2.1805e+00]
         [                            4.3525e-02   9.1667e-01]

H(10) =  [  4.0024e+00  -6.2734e-01  -7.0227e+00  -2.1916e+01]
         [  7.6515e-03   2.9123e+00  -9.9902e-01   3.3560e+00]
         [              -8.0039e-02   2.0877e+00   3.3549e+00]
         [                           -7.1186e-04   9.9762e-01]

H(15) =  [  4.0001e+00  -1.0549e-01  -7.0411e+00   2.1902e+01]
         [  1.6833e-03   2.9889e+00  -8.9365e-01  -3.2181e+00]
         [              -1.2248e-02   2.0111e+00  -3.6032e+00]
         [                            2.0578e-05   9.9993e-01]

H(20) =  [  4.0000e+00  -3.1163e-02  -7.0425e+00  -2.1898e+01]
         [  3.9562e-04   2.9986e+00  -8.7411e-01   3.2072e+00]
         [              -1.6441e-03   2.0014e+00   3.6377e+00]
         [                           -6.3689e-07   1.0000e-00]

H(25) =  [  4.0000e+00  -2.1399e-02  -7.0428e+00   2.1897e+01]
         [  9.3764e-05   2.9998e+00  -8.7056e-01  -3.2086e+00]
         [              -2.1704e-04   2.0002e+00  -3.6423e+00]
         [                            1.9878e-08   1.0000e-00]

H(30) =  [  4.0000e+00  -2.0143e-02  -7.0429e+00  -2.1897e+01]
         [  2.2247e-05   3.0000e+00  -8.6987e-01   3.2095e+00]
         [              -2.8591e-05   2.0000e+00   3.6429e+00]
         [                           -6.2108e-10   1.0000e-00]
```

Finally we compute the element-wise quotients of the last two matrices.

```
H(30)./H(29) =  [  1.0000   0.9954   1.0000  -1.0000]
                [  0.7500   1.0000   0.9999  -1.0000]
                [           0.6667   1.0000  -1.0000]
                [                   -0.5000   1.0000]
```

Again the elements in the lower off-diagonal reflect nicely the convergence rates in (4.2).

4.2.2 Complexity

We give the algorithm for a single Hessenberg-QR-step in a MATLAB-like way, see Algorithm 4.2. By

$$H_{k:j,m:n} \in \mathbb{C}^{(j-k+1)\times(n-m+1)}$$

we denote the submatrix of H consisting of rows k through j and columns m through n.

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**

If we neglect the determination of the parameters c_k and s_k, see (4.5), then *each* of the two loops requires

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

A **flop** is a floating point operation $(+, -, \times, /)$. We do not distinguish between them, although they may slightly differ in their execution time on a computer. Optionally, we also have to execute the operation $U_k := U_{k-1}Q_k$ of Algorithm 4.1. This is achieved by a loop similar to the second loop in Algorithm 4.2. Since all the rows and columns of U are

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**

involved, executing the loop costs

$$\sum_{i=1}^{n-1} 6n \approx 6n^2 \quad \text{flops.}$$

Altogether, a QR step with a Hessenberg matrix, including the update of the unitary transformation matrix, requires $12n^2$ floating point operations. This has to be set in relation to a QR step with a full matrix that costs $\frac{7}{3}n^3$. Consequently, we have gained a factor of $\mathcal{O}(n)$ in terms of operations by moving from dense to Hessenberg form. However, we may still have very slow convergence if one of the quotients $|\lambda_k|/|\lambda_{k+1}|$ is close to 1.

4.3 The Householder reduction to Hessenberg form

In the previous section we discovered that it is a good idea to perform the QR algorithm with Hessenberg matrices instead of full matrices. But we have not discussed how we transform a full matrix (by means of similarity transformations) into Hessenberg form. We catch up on this issue in this section.

4.3.1 Householder reflectors

Givens rotations are designed to zero a single element in a vector. Householder reflectors are more efficient if a number of elements of a vector are to be zeroed at once. Here, we follow the presentation given in [6].

Definition 4.3 A matrix of the form

$$P = I - 2\mathbf{u}\mathbf{u}^*, \qquad \|\mathbf{u}\| = 1,$$

is called a **Householder reflector**.

It is easy to verify that Householder reflectors are *Hermitian* and that $P^2 = I$. From this we deduce that P is *unitary*. It is clear that we only have to store the **Householder vector u** to be able to multiply a vector (or a matrix) with P,

$$P\mathbf{x} = \mathbf{x} - \mathbf{u}(2\mathbf{u}^*\mathbf{x}). \tag{4.6}$$

This multiplication only costs $4n$ flops where n is the length of the vectors.

A task that we repeatedly want to carry out with Householder reflectors is to transform a vector $\mathbf{x}$ on a multiple of $\mathbf{e}_1$,

$$P\mathbf{x} = \mathbf{x} - \mathbf{u}(2\mathbf{u}^*\mathbf{x}) = \alpha\mathbf{e}_1.$$

Since P is unitary, we must have $\alpha = \rho\|\mathbf{x}\|$, where $\rho \in \mathbb{C}$ has absolute value one. Therefore,

$$\mathbf{u} = \frac{\mathbf{x} - \rho\|\mathbf{x}\|\mathbf{e}_1}{\|\mathbf{x} - \rho\|\mathbf{x}\|\mathbf{e}_1\|} = \frac{1}{\|\mathbf{x} - \rho\|\mathbf{x}\|\mathbf{e}_1\|}\begin{bmatrix} x_1 - \rho\|\mathbf{x}\| \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

We can freely choose ρ provided that $|\rho| = 1$. Let $x_1 = |x_1|e^{i\phi}$. To avoid numerical cancellation we set $\rho = -e^{i\phi}$.

In the real case, one commonly sets $\rho = -\text{sign}(x_1)$. If $x_1 = 0$ we can set ρ in any way.

4.3.2 Reduction to Hessenberg form

Now we show how to use Householder reflectors to reduce an arbitrary square matrix to Hessenberg form. We show the idea by means of a 5×5 example. In the first step of the reduction we introduce zeros in the first column below the second element,

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_1*} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow{*P_1} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} = P_1^* A P_1.$$

Notice that $P_1 = P_1^*$ since it is a Householder reflector! It has the structure

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & I_4 - 2\mathbf{u}_1\mathbf{u}_1^* \end{bmatrix}.$$

The Householder vector $\mathbf{u}_1$ is determined such that

$$(I - 2\mathbf{u}_1\mathbf{u}_1^*) \begin{bmatrix} a_{21} \\ a_{31} \\ a_{41} \\ a_{51} \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \text{with} \quad \mathbf{u}_1 = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}.$$

The multiplication of P_1 from the left inserts the desired zeros in column 1 of A. The multiplication from the right is necessary in order to have similarity. Because of the nonzero structure of P_1 the first column of P_1A is not affected. Hence, the zeros stay there.

The reduction continues in a similar way:

$$P_1AP_1 = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_2 * / * P_2} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix}$$

$$\xrightarrow{P_3 * / * P_3} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix} = P_3P_2P_1A\underbrace{P_1P_2P_3}_{U}.$$

Algorithm 4.3 gives the details for the general $n \times n$ case. In step 4 of this algorithm, the Householder reflector is generated such that

$$(I - 2\mathbf{u}_k\mathbf{u}_k^*) \begin{bmatrix} a_{k+1,k} \\ a_{k+2,k} \\ \vdots \\ a_{n,k} \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \text{with} \quad \mathbf{u}_k = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-k} \end{bmatrix} \quad \text{and} \quad |\alpha| = \|\mathbf{x}\|$$

according to the considerations of the previous subsection. The Householder vectors are stored at the locations of the zeros. Therefore the matrix $U = P_1 \cdots P_{n-2}$ that effects the similarity transformation from the full A to the Hessenberg H is computed after all Householder vectors have been generated, thus saving $(2/3)n^3$ flops. The overall complexity of the reduction is

- Application of P_k from the left: $\sum_{k=1}^{n-2} 4(n-k-1)(n-k) \approx \frac{4}{3}n^3$
- Application of P_k from the right: $\sum_{k=1}^{n-2} 4(n)(n-k) \approx 2n^3$

Algorithm 4.3 Reduction to Hessenberg form

1: 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.
2: **for** $k = 1$ to $n-2$ **do**
3: Generate the Householder reflector P_k;
4: /* Apply $P_k = I_k \oplus (I_{n-k} - 2\mathbf{u_k}\mathbf{u_k}^*)$ from the left to A */
5: $A_{k+1:n,k:n} := A_{k+1:n,k:n} - 2\mathbf{u_k}(\mathbf{u_k}^* A_{k+1:n,k:n})$;
6: /* Apply P_k from the right, $A := AP_k$ */
7: $A_{1:n,k+1:n} := A_{1:n,k+1:n} - 2(A_{1:n,k+1:n}\mathbf{u_k})\mathbf{u_k}^*$;
8: **end for**
9: **if** eigenvectors are desired form $U = P_1 \cdots P_{n-2}$ **then**
10: $U := I_n$;
11: **for** $k = n-2$ downto 1 **do**
12: /* Update $U := P_k U$ */
13: $U_{k+1:n,k+1:n} := U_{k+1:n,k+1:n} - 2\mathbf{u_k}(\mathbf{u_k}^* U_{k+1:n,k+1:n})$;
14: **end for**
15: **end if**

- Form $U = P_1 \cdots P_{n-2}$: $\sum_{k=1}^{n-2} 4(n-k)(n-k) \approx \frac{4}{3}n^3$

Thus, the reduction to Hessenberg form costs $\frac{10}{3}n^3$ flops without forming the transformation matrix and $\frac{14}{3}n^3$ including forming this matrix.

4.4 Improving the convergence of the QR algorithm

We have seen how the QR algorithm for computing the Schur form of a matrix A can be executed more economically if the matrix A is first transformed to Hessenberg form. Now we want to show how the convergence of the Hessenberg QR algorithm can be improved dramatically by introducing **(spectral) shifts** into the algorithm.

Lemma 4.4 *Let H be an **irreducible** Hessenberg matrix, i.e., $h_{i+1,i} \neq 0$ for all $i = 1, \ldots, n-1$. Let $H = QR$ be the QR factorization of H. Then for the diagonal elements of R we have*

$$|r_{kk}| > 0, \qquad \text{for all } k < n.$$

Thus, if H is singular then $r_{nn} = 0$.

Proof. Let us look at the k-th step of the Hessenberg QR factorization. For illustration, let us consider the case $k = 3$ in a 5×5 example, where the matrix has the structure

$$\begin{bmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

The plus-signs indicate elements that have been modified. In step 3, the (nonzero) element h_{43} will be zeroed by a Givens rotation $G(3, 4, \varphi)$ that is determined such that

$$\begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} \tilde{h}_{kk} \\ h_{k+1,k} \end{bmatrix} = \begin{bmatrix} r_{kk} \\ 0 \end{bmatrix}.$$

Because the Givens rotation preserves vector lengths, we have

$$|r_{kk}|^2 = |\tilde{h}_{kk}|^2 + |h_{k+1,k}|^2 \geq |h_{k+1,k}|^2 > 0,$$

which confirms the claim. ■

We apply this Lemma to motivate a further strategy to speed up the convergence of the QR algorithm.

Let λ be an eigenvalue of the irreducible Hessenberg matrix H. Let us check what happens it we perform

---

1: $H - \lambda I = QR$ /* QR factorization */
2: $\overline{H} = RQ + \lambda I$

First we notice that $\overline{H} \sim H$. In fact,

$$\overline{H} = Q^*(H - \lambda I)Q + \lambda I = Q^*HQ.$$

Second, by Lemma 4.4 we have

$$H - \lambda I = QR, \qquad \text{with} \qquad R = \begin{bmatrix} \ddots \\ 0 \end{bmatrix}.$$

Thus,

$$RQ = \begin{bmatrix} \ddots \\ 0\,0 \end{bmatrix}$$

and

$$\overline{H} = RQ + \lambda I = \begin{bmatrix} \ddots \\ 0\,\lambda \end{bmatrix} = \begin{bmatrix} \overline{H}_1 & \mathbf{h}_1 \\ \mathbf{0}^T & \lambda \end{bmatrix}.$$

So, if we apply a QR step with a **perfect shift** to a Hessenberg matrix, the eigenvalue drops out. We then could **deflate**, i.e., proceed the algorithm with the smaller matrix $\overline{H}_1$.

Remark 4.2. We could prove the existence of the Schur decomposition in the following way. (1) transform the arbitrary matrix to Hessenberg form. (2) Do the perfect shift Hessenberg QR with the eigenvalues which we known to exist one after the other. □

4.4.1 A numerical example

We use a matrix of a previous MATLAB experiments to show that perfect shifts actually work.

```
D = diag([4 3 2 1]); rand('seed',0);
S=rand(4); S = (S - .5)*2;
A = S*D/S;
format short e
H = hess(A)
[Q,R] = qr(H - 2*eye(4))
H1 = R*Q + 2*eye(4)
format long
lam = eig(H1(1:3,1:3))
```

MATLAB produces the output

```
H =  [ -4.4529e-01  -1.8641e+00  -2.8109e+00   7.2941e+00]
     [  8.0124e+00   6.2898e+00   1.2058e+01  -1.6088e+01]
     [               4.0087e-01   1.1545e+00  -3.3722e-01]
     [                           -1.5744e-01   3.0010e+00]

Q =  [ -2.9190e-01  -7.6322e-01  -4.2726e-01  -3.8697e-01]
     [  9.5645e-01  -2.3292e-01  -1.3039e-01  -1.1810e-01]
     [               6.0270e-01  -5.9144e-01  -5.3568e-01]
     [                           -6.7130e-01   7.4119e-01]

R =  [  8.3772e+00   4.6471e+00   1.2353e+01  -1.7517e+01]
     [               6.6513e-01  -1.1728e+00  -2.0228e+00]
     [                            2.3453e-01  -1.4912e+00]
     [                                        -2.4425e-14]

H1 =  [  3.9994e+00  -3.0986e-02   2.6788e-01  -2.3391e+01]
      [  6.3616e-01   1.1382e+00   1.9648e+00  -9.4962e-01]
      [               1.4135e-01   2.8623e+00  -1.2309e+00]
      [                            1.6396e-14   2.0000e+00]

lam =  [9.99999999999993e-01 4.00000000000003e+00 3.00000000000000e+00]
```

4.4.2 QR algorithm with shifts

This considerations indicate that it may be good to introduce shifts into the QR algorithm. However, we cannot choose perfect shifts because we do not know the eigenvalues of the matrix! We therefore need heuristics how to *estimate* eigenvalues. One such heuristic is the **Rayleigh quotient shift**: Set the shift σ_k in the k-th step of the QR algorithm equal to the last diagonal element:

$$\sigma_k := h_{n,n}^{(k-1)} = \mathbf{e}_n^* H^{(k-1)} \mathbf{e}_n. \tag{4.7}$$

Algorithm 4.4 The Hessenberg QR algorithm with Rayleigh quotient shift

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

Algorithm 4.4 implements this heuristic. Notice that the shift changes in each iteration step! Notice also that **deflation** is incorporated in Algorithm 4.4. As soon as the last lower off-diagonal element is sufficiently small, it is *declared* zero, and the algorithm proceeds with a smaller matrix. In Algorithm 4.4 the 'active portion' of the matrix is $m \times m$.

Lemma 4.4 guarantees that a zero is produced at position $(n, n-1)$ in the Hessenberg matrix H if the shift equals an eigenvalue of H. What happens, if $h_{n,n}$ is a *good* approximation to an eigenvalue of H? Let us assume that we have an irreducible Hessenberg matrix

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \varepsilon & h_{n,n} \end{bmatrix},$$

where ε is a small quantity. If we perform a shifted Hessenberg QR step, we first have to factor $H - h_{n,n}I$, $QR = H - h_{n,n}I$. After $n-2$ steps of this factorization the R-factor is almost upper triangular,

$$\begin{bmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & 0 & \alpha & \beta \\ 0 & 0 & 0 & \varepsilon & 0 \end{bmatrix}.$$

From (4.5) we see that the last Givens rotation has the nontrivial elements

$$c_{n-1} = \frac{\alpha}{\sqrt{|\alpha|^2 + |\varepsilon|^2}}, \qquad s_{n-1} = \frac{-\varepsilon}{\sqrt{|\alpha|^2 + |\varepsilon|^2}}.$$

Applying the Givens rotations from the right one sees that the last lower off-diagonal element of $\overline{H} = RQ + h_{n,n}I$ becomes

$$\bar{h}_{n,n-1} = \frac{\varepsilon^2 \beta}{\alpha^2 + \varepsilon^2}. \tag{4.8}$$

So, we have *quadratic convergence* unless α is also tiny.

A second even more often used shift strategy is the **Wilkinson shift**:

$$\sigma_k := \text{eigenvalue of } \begin{bmatrix} h_{n-1,n-1}^{(k-1)} & h_{n-1,n}^{(k-1)} \\ h_{n,n-1}^{(k-1)} & h_{n,n}^{(k-1)} \end{bmatrix} \text{ that is closer to } h_{n,n}^{(k-1)}. \tag{4.9}$$

4.4.3 A numerical example

We give an example for the Hessenberg QR algorithm with shift, but without deflation. The MATLAB code

```
D = diag([4 3 2 1]);
rand('seed',0);
S=rand(4); S = (S - .5)*2;
A = S*D/S;
H = hess(A)

for i=1:8,
   [Q,R] = qr(H-H(4,4)*eye(4)); H = R*Q+H(4,4)*eye(4);
end
```

produces the output

```
H( 0) =  [ -4.4529e-01  -1.8641e+00  -2.8109e+00   7.2941e+00]
         [  8.0124e+00   6.2898e+00   1.2058e+01  -1.6088e+01]
         [  0.0000e+00   4.0087e-01   1.1545e+00  -3.3722e-01]
         [  0.0000e+00   0.0000e+00  -1.5744e-01   3.0010e+00]

H( 1) =  [  3.0067e+00   1.6742e+00  -2.3047e+01  -4.0863e+00]
         [  5.2870e-01   8.5146e-01   1.1660e+00  -1.5609e+00]
         [              -1.7450e-01   3.1421e+00  -1.1140e-01]
         [                           -1.0210e-03   2.9998e+00]

H( 2) =  [  8.8060e-01  -4.6537e-01   9.1630e-01   1.6146e+00]
         [ -1.7108e+00   5.3186e+00   2.2839e+01  -4.0224e+00]
         [              -2.2542e-01   8.0079e-01   5.2445e-01]
         [                           -1.1213e-07   3.0000e+00]

H( 3) =  [  1.5679e+00   9.3774e-01   1.5246e+01   1.2703e+00]
         [  1.3244e+00   2.7783e+00   1.7408e+01   4.1764e+00]
         [               3.7230e-02   2.6538e+00  -7.8404e-02]
         [                            8.1284e-15   3.0000e+00]

H( 4) =  [  9.9829e-01  -7.5537e-01  -5.6915e-01   1.9031e+00]
         [ -3.2279e-01   5.1518e+00   2.2936e+01  -3.9104e+00]
         [              -1.6890e-01   8.4993e-01   3.8582e-01]
         [                           -5.4805e-30   3.0000e+00]

H( 5) =  [  9.3410e-01  -3.0684e-01   3.0751e+00  -1.2563e+00]
         [  3.5835e-01   3.5029e+00   2.2934e+01   4.1807e+00]
         [               3.2881e-02   2.5630e+00  -7.2332e-02]
         [                            1.1313e-59   3.0000e+00]

H( 6) =  [  1.0005e+00  -8.0472e-01  -8.3235e-01   1.9523e+00]
         [ -7.5927e-02   5.1407e+00   2.2930e+01  -3.8885e+00]
         [              -1.5891e-01   8.5880e-01   3.6112e-01]
         [                          -1.0026e-119   3.0000e+00]

H( 7) =  [  9.7303e-01  -6.4754e-01  -8.9829e-03  -1.8034e+00]
         [  8.2551e-02   3.4852e+00   2.3138e+01   3.9755e+00]
         [               3.3559e-02   2.5418e+00  -7.0915e-02]
         [                           3.3770e-239   3.0000e+00]

H( 8) =  [  1.0002e+00  -8.1614e-01  -8.9331e-01   1.9636e+00]
         [ -1.8704e-02   5.1390e+00   2.2928e+01  -3.8833e+00]
         [              -1.5660e-01   8.6086e-01   3.5539e-01]
         [                                    0   3.0000e+00]
```

The numerical example shows that the shifted Hessenberg QR algorithm can work very nicely. In this example the (4,3) element is about 10^{-30} after 3 steps. (We could stop there.) The example also nicely shows a quadratic convergence rate.

4.5 The double shift QR algorithm

The shifted Hessenberg QR algorithm does not always work so nicely as in the previous example. If α in (4.8) is $\mathcal{O}(\varepsilon)$ then $h_{n,n-1}$ can be large. (A small α indicates a near singular $H_{1:n-1,1:n-1}$.)

Another problem occurs if real Hessenberg matrices have complex eigenvalues. We know that for reasonable convergence rates the shifts must be complex. If an eigenvalue λ has been found we can execute a single perfect shift with $\bar{\lambda}$. It is (for rounding errors) unprobable however that we will get back to a real matrix.

Since the eigenvalues come in complex conjugate pairs it is natural to search for a pair of eigenvalues right-away. This is done by collapsing two shifted QR steps in one **double step** with the two shifts being complex conjugates of each other.

Let σ_1 and σ_2 be two eigenvalues of the real matrix (cf. Wilkinson shift (4.9))

$$G = \begin{bmatrix} h_{n-1,n-1}^{(k-1)} & h_{n-1,n}^{(k-1)} \\ h_{n,n-1}^{(k-1)} & h_{n,n}^{(k-1)} \end{bmatrix} \in \mathbb{R}^{2\times 2}.$$

If $\sigma_1 \in \mathbb{C}\setminus\mathbb{R}$ then $\sigma_2 = \bar{\sigma}_1$. Let us perform two QR steps using σ_1 and σ_2 as shifts. Setting $k = 1$ for convenience we get

$$\begin{aligned} H_0 - \sigma_1 I &= Q_1 R_1, \\ H_1 &= R_1 Q_1 + \sigma_1 I, \\ H_1 - \sigma_2 I &= Q_2 R_2, \\ H_2 &= R_2 Q_2 + \sigma_2 I. \end{aligned} \tag{4.10}$$

From the second and third equation in (4.10) we obtain

$$R_1 Q_1 + (\sigma_1 - \sigma_2) I = Q_2 R_2.$$

Multiplying this equation with Q_1 from the left and with R_1 from the right we get

$$\begin{aligned} Q_1 R_1 Q_1 R_1 + (\sigma_1 - \sigma_2) Q_1 R_1 &= Q_1 R_1 (Q_1 R_1 + (\sigma_1 - \sigma_2) I) \\ &= (H_0 - \sigma_1 I)(H_0 - \sigma_2 I) = Q_1 Q_2 R_2 R_1. \end{aligned}$$

Because $\sigma_2 = \bar{\sigma}_1$ we have

$$M := (H_0 - \sigma_1 I)(H_0 - \bar{\sigma}_1 I) = H_0^2 - 2\mathrm{Re}(\sigma) H_0 + |\sigma|^2 I = Q_1 Q_2 R_2 R_1.$$

Therefore, $(Q_1 Q_2)(R_2 R_1)$ is the QR factorization of a *real matrix*. We can choose (scale) Q_1 and Q_2 such that $Z := Q_1 Q_2$ is real orthogonal. (Then also $R_2 R_1$ is real.) By consequence,

$$H_2 = (Q_1 Q_2)^* H_0 (Q_1 Q_2) = Z^T H_0 Z$$

is real.

A procedure to compute H_2 by avoiding complex arithmetic could consist of three steps:

1. Form the real matrix $M = H_0^2 - sH_0 + tI$ with $s = 2\mathrm{Re}(\sigma) = \mathrm{trace}(G) = h_{n-1,n-1}^{(k-1)} + h_{n,n}^{(k-1)}$ and $t = |\sigma|^2 = \det(G) = h_{n-1,n-1}^{(k-1)} h_{n,n}^{(k-1)} - h_{n-1,n}^{(k-1)} h_{n,n-1}^{(k-1)}$. Notice that M has two lower off-diagonals,
$$M = \begin{bmatrix} \ddots & \ddots & \ddots & \\ \ddots & \ddots & \ddots & \\ \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ & & \ddots & \ddots \end{bmatrix}.$$
2. Compute the QR factorization $M = ZR$,
3. Set $H_2 = Z^T H_0 Z$.

This procedure is however too expensive since item 1, i.e., forming H^2 requires $\mathcal{O}(n^3)$ flops.

A remedy for the situation is provided by the Implicit Q Theorem.

Theorem 4.5 ***(The implicit Q theorem)*** *Let $A \in \mathbb{R}^{n\times n}$. Let $Q = [\mathbf{q}_1, \ldots, \mathbf{q}_n]$ and $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ be orthogonal matrices that both similarly transform A to Hessenberg form, $H = Q^T AQ$ and $G = V^T AV$. Let k denote the smallest positive integer for which $h_{k+1,k} = 0$, with $k = n$ if H is irreducible.*

If $\mathbf{q}_1 = \mathbf{v}_1$ then $\mathbf{q}_i = \pm\mathbf{v}_i$ and $|h_{i,i-1}| = |g_{i,i-1}|$ for $i = 2, \ldots, k$. If $k < n$, then $g_{k+1,k} = 0$.

Proof. [6] Let $W = V^T Q$. Clearly, W is orthogonal, and $GW = WH$.

We first show that the first k columns of W form an upper triangular matrix, i.e.,

$$\mathbf{w}_i = W\mathbf{e}_i \in \text{span}\{\mathbf{e}_1, \ldots, \mathbf{e}_i\}, \qquad i \le k. \tag{4.11}$$

(Notice that orthogonal upper triangular matrices are diagonal with diagonal entries ± 1.)

This is proced inductively. For $i = 1$ we have $\mathbf{w}_1 = \mathbf{e}_1$ by the assumption that $\mathbf{q}_1 = \mathbf{v}_1$. For $1 < i \le k$ we assume that (4.11) is true for $\mathbf{w}_i$ and use the equality $GW = WH$. The $(i-1)$-th column of this equation reads

$$G\mathbf{w}_{i-1} = GW\mathbf{e}_{i-1} = WH\mathbf{e}_{i-1} = \sum_{j=1}^{i} \mathbf{w}_j h_{j,i-1}.$$

Since $h_{i,i-1} \neq 0$ we have

$$\mathbf{w}_i h_{i,i-1} = G\mathbf{w}_{i-1} - \sum_{j=1}^{i-1} \mathbf{w}_j h_{j,i-1} \in \text{span}\{\mathbf{e}_1, \ldots \mathbf{e}_i\},$$

as G is a Hessenberg matrix. So, the upper-left $k \times k$ block of W is upper triangular. Since the columns of W are orthogonal we conclude that $\mathbf{w}_i = \pm\mathbf{e}_i$, $i \le k$.

Since $\mathbf{w}_i = \pm V^T Q\mathbf{e}_i = V^T\mathbf{q}_i = \pm\mathbf{e}_i$ we see that $\mathbf{q}_i$ is orthogonal to all columns of V except the i-th. Therefore, we must have $\mathbf{q}_i = \pm\mathbf{v}_i$. Further,

$$h_{i,i-1} = \mathbf{e}_i^T H\mathbf{e}_{i-1} = \mathbf{e}_i^T Q^T AQ\mathbf{e}_{i-1} = \mathbf{e}_i^T Q^T VGV^T Q\mathbf{e}_{i-1} = \mathbf{w}_i^T G\mathbf{w}_{i-1} = \pm g_{i,i-1},$$

thus, $|h_{i,i-1}| = |g_{i,i-1}|$. If $h_{k+1,k} = 0$ then

$$g_{k+1,k} = \mathbf{e}_{k+1}^T G\mathbf{e}_k = \pm\,\mathbf{e}_{k+1}^T GW\mathbf{e}_k = \pm\,\mathbf{e}_{k+1}^T WH\mathbf{e}_k = \pm\,\mathbf{e}_{k+1}^T \sum_{j=1}^{k} \mathbf{w}_j h_{j,k} = 0.$$

since $\mathbf{e}_{k+1}^T\mathbf{w}_j = \pm\mathbf{e}_{k+1}^T\mathbf{e}_j = 0$ for $j \le k$. ∎

Golub and van Loan [6, p.347] write that *"The gist of the implicit Q theorem is that if $Q^T AQ = H$ and $Z^T AZ = G$ are both unreduced Hessenberg matrices and Q and Z have the same first column, then G and H are "essentially equal" in the sense that $G = DHD$ with $D = \text{diag}(\pm 1, \ldots, \pm 1)$."*

We apply the Implicit Q Theorem in the following way: We want to compute the Hessenberg matrix $H_{k+1} = Z^T H_{k-1} Z$ where ZR is the QR factorization of $M = H_{k-1}^2 - sH_{k-1} + tI$. The Implicit Q Theorem now tells us that we essentially get H_{k+1} by any orthogonal similarity transformation $H_{k-1} \to Z_1^* H_{k-1} Z_1$ provided that $Z_1^* HZ_1$ is Hessenberg and $Z_1\mathbf{e}_1 = Z\mathbf{e}_1$.

Let P_0 be the Householder reflector with

$$P_0^T M\mathbf{e}_1 = P_0^T(H_{k-1}^2 - 2\mathrm{Re}(\sigma)H_{k-1} + |\sigma|^2 I)\,\mathbf{e}_1 = \alpha\,\mathbf{e}_1.$$

Since only the first three elements of the first column $M\mathbf{e}_1$ of M are nonzero, P_0 has the structure

$$P_0 = \begin{bmatrix} \times & \times & \times & & & \\ \times & \times & \times & & & \\ \times & \times & \times & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix}.$$

So,

$$H'_{k-1} := P_0^T H_{k-1} P_0 = \left[\begin{array}{ccc|cccc} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ + & \times & \times & \times & \times & \times & \times \\ \hline + & + & \times & \times & \times & \times & \times \\ & & & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \end{array}\right].$$

We now reduce $P_0^T H_{k-1} P_0$ similarly to Hessenberg form the same way as we did earlier, by a sequence of Householder reflectors $P_1, \ldots, P_{n-2}$. However, $P_0^T H_{k-1} P_0$ is a Hessenberg matrix up to the **bulge** at the top left. We take into account this structure when forming the $P_i = I - 2\mathbf{p}_i\mathbf{p}_i^T$. So, the structures of P_1 and of $P_1^T P_0^T H_{k-1} P_0 P_1$ are

$$P_1 = \begin{bmatrix} 1 & & & & & & \\ & \times & \times & \times & & & \\ & \times & \times & \times & & & \\ & \times & \times & \times & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & 1 \end{bmatrix}, \quad H''_{k-1} = P_1^T H'_{k-1} P_1 = \left[\begin{array}{c|ccc|ccc} \times & \times & \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times & \times \\ 0 & + & \times & \times & \times & \times & \times \\ \hline & + & + & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \end{array}\right].$$

The transformation with P_1 has chased the bulge one position down the diagonal. The consecutive reflectors push it further by one position each until it falls out of the matrix at the end of the diagonal. Pictorially, we have

$$H'''_{k-1} = P_2^T H''_{k-1} P_2 = \left[\begin{array}{cc|ccc|cc} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ \hline & \times & \times & \times & \times & \times & \times \\ & 0 & \times & \times & \times & \times & \times \\ & 0 & + & \times & \times & \times & \times \\ \hline & & + & + & \times & \times & \times \\ & & & & & \times & \times \end{array}\right]$$

$$H''''_{k-1} = P_3^T H'''_{k-1} P_3 = \left[\begin{array}{ccc|ccc|c} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times & \times \\ \hline & & \times & \times & \times & \times & \times \\ & & 0 & \times & \times & \times & \times \\ & & 0 & + & \times & \times & \times \\ \hline & & & + & + & \times & \times \end{array}\right]$$

$$H''''''_{k-1} = P_4^T H''''_{k-1} P_4 = \left[\begin{array}{ccccccc} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times & \times \\ \hline & & & \times & \times & \times & \times \\ & & & 0 & \times & \times & \times \\ & & & 0 & + & \times & \times \end{array}\right]$$

$$H''''''_{k-1} = P_5^T H''''''_{k-1} P_5 = \left[\begin{array}{ccccccc} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times & \times \\ & & & \times & \times & \times & \times \\ \hline & & & & \times & \times & \times \\ & & & & 0 & \times & \times \end{array}\right]$$

It is easy to see that the Householder vector $\mathbf{p}_i$, $i < n-2$, has only three nonzero elements at position $i+1, i+2, i+3$. Of $\mathbf{p}_{n-2}$ only the last two elements are nonzero. Clearly, $P_0 P_1 \cdots P_{n-2}\mathbf{e}_1 = P_0\mathbf{e}_1 = M\mathbf{e}_1/\alpha$.

Remark 4.3. Notice that in Algorithm 4.5 a double step is taken also if the eigenvalues of

$$G = \begin{bmatrix} h_{qq} & h_{qp} \\ h_{pq} & h_{pp} \end{bmatrix}$$

are real. As in the complex case we set $s = \mathrm{trace}(G)$ and $t = \det(G)$. □

4.5.1 A numerical example

We consider a simple MATLAB implementation of the Algorithm 4.5 to compute the eigenvalues of the real matrix

$$A = \begin{bmatrix} 7 & 3 & 4 & -11 & -9 & -2 \\ -6 & 4 & -5 & 7 & 1 & 12 \\ -1 & -9 & 2 & 2 & 9 & 1 \\ -8 & 0 & -1 & 5 & 0 & 8 \\ -4 & 3 & -5 & 7 & 2 & 10 \\ 6 & 1 & 4 & -11 & -7 & -1 \end{bmatrix}$$

that has the spectrum

$$\sigma(A) = \{1 \pm 2i, 3, 4, 5 \pm 6i\}.$$

The intermediate output of the code was (after some editing) the following:

```
>> H=hess(A)

H(0) =

    7.0000    7.2761    5.8120   -0.1397    9.0152    7.9363
   12.3693    4.1307   18.9685   -1.2071   10.6833    2.4160
         0   -7.1603    2.4478   -0.5656   -4.1814   -3.2510
         0         0   -8.5988    2.9151   -3.4169    5.7230
         0         0         0    1.0464   -2.8351  -10.9792
         0         0         0         0    1.4143    5.3415
```

Algorithm 4.5 The Francis double step QR algorithm

Let $H_0 = H \in \mathbb{R}^{n\times n}$ be an upper Hessenberg matrix. This algorithm computes its real Schur form $H = UTU^T$ using the Francis double step QR algorithm. T is a quasi upper triangular matrix.
$p := n$; /* p indicates the 'active' matrix size. */
while $p > 2$ **do**
  $q := p - 1$;
  $s := H_{q,q} + H_{p,p}$; $t := H_{q,q}H_{p,p} - H_{q,p}H_{p,q}$;
  /* compute first 3 elements of first column of M */
  $x := H_{1,1}^2 + H_{1,2}H_{2,1} - sH_{1,1} + t$;
  $y := H_{2,1}(H_{1,1} + H_{2,2} - s)$;
  $z := H_{2,1}H_{3,2}$;
  for $k = 0$ to $p - 3$ **do**
    Determine the Householder reflector P with $P^T\ [x; y; z]^T = \alpha\mathbf{e}_1$;
    $r := \max\{1, k\}$;
    $H_{k+1:k+3,r:n} := P^T H_{k+1:k+3,r:n}$;
    $r := \min\{k + 4, p\}$;
    $H_{1:r,k+1:k+3} := H_{1:r,k+1:k+3}P$;
    $x := H_{k+2,k+1}$; $y := H_{k+3,k+1}$;
    if $k < p - 3$ **then**
      $z := H_{k+4,k+1}$;
    end if
  end for
  Determine the Givens rotation P with $P^T\ [x; y]^T = \alpha\mathbf{e}_1$;
  $H_{q:p,p-2:n} := P^T H_{q:p,p-2:n}$;
  $H_{1:p,p-1:p} := H_{1:p,p-1:p}P$;
  /* check for convergence */
  if $|H_{p,q}| < \varepsilon\ (|H_{q,q}| + |H_{p,p}|)$ **then**
    $H_{p,q} := 0$; $p := p - 1$; $q := p - 1$;
  else if $|H_{p-1,q-1}| < \varepsilon\ (|H_{q-1,q-1}| + |H_{q,q}|)$ **then**
    $H_{p-1,q-1} := 0$; $p := p - 2$; $q := p - 1$;
  end if
end while

```
>> PR=qr2st(H)

[it_step, p = n_true, H(p,p-1), H(p-1,p-2)]

     1     6  -1.7735e-01  -1.2807e+00
     2     6  -5.9078e-02  -1.7881e+00
     3     6  -1.6115e-04  -5.2705e+00
     4     6  -1.1358e-07  -2.5814e+00
     5     6   1.8696e-14   1.0336e+01
     6     6  -7.1182e-23  -1.6322e-01


H(6) =

    5.0000    6.0000    2.3618    5.1837  -13.4434   -2.1391
   -6.0000    5.0000    2.9918   10.0456   -8.7743  -21.0094
    0.0000   -0.0001   -0.9393    3.6939   11.7357    3.8970
```

```
    0.0000   -0.0000   -1.9412    3.0516    2.9596  -10.2714
         0    0.0000    0.0000   -0.1632    3.8876    4.1329
         0         0         0    0.0000   -0.0000    3.0000


     7     5   1.7264e-02  -7.5016e-01
     8     5   2.9578e-05  -8.0144e-01
     9     5   5.0602e-11  -4.6559e+00
    10     5  -1.3924e-20  -3.1230e+00


H(10) =

    5.0000    6.0000   -2.7603    1.3247   11.5569   -2.0920
   -6.0000    5.0000  -10.7194    0.8314   11.8952   21.0142
   -0.0000   -0.0000    3.5582    3.3765    5.9254   -8.5636
   -0.0000   -0.0000   -3.1230   -1.5582  -10.0935   -6.3406
         0         0         0    0.0000    4.0000    4.9224
         0         0         0    0.0000         0    3.0000


    11     4   1.0188e+00  -9.1705e-16


H(11) =

    5.0000    6.0000  -10.2530    4.2738  -14.9394  -19.2742
   -6.0000    5.0000   -0.1954    1.2426    7.2023   -8.6299
   -0.0000   -0.0000    2.2584   -5.4807  -10.0623    4.4380
    0.0000   -0.0000    1.0188   -0.2584   -5.9782   -9.6872
         0         0         0         0    4.0000    4.9224
         0         0         0    0.0000         0    3.0000
```

4.5.2 The complexity

We first estimate the complexity of a single step of the double step Hessenberg QR algorithm. The most expensive operations are the applications of the 3×3 Householder reflectors in steps 13 and 15 of Algorithm 4.5. Let us first count the flops for applying the Householder reflector to a 3-vector,

$$\mathbf{x} := (I - 2\mathbf{u}\mathbf{u}^T)\mathbf{x} = \mathbf{x} - \mathbf{u}(2\mathbf{u}^T\mathbf{x}).$$

The inner product $\mathbf{u}^T\mathbf{x}$ costs 5 flops, multiplying with 2 another one. The operation $\mathbf{x} := \mathbf{x} - \mathbf{u}\gamma$, $\gamma = 2\mathbf{u}^T\mathbf{x}$, cost 6 flops, altogether 12 flops.

In the k-th step of the loop there are $n-k$ of these application from the left in step 13 and $k+4$ from the right in step 15. In this step there are thus about $12n+\mathcal{O}(1)$ flops to be executed. As k is running from 1 to $p-3$. We have about $12pn$ flops for this step. Since p runs from n down to about 2 we have $6n^3$ flops. If we assume that two steps are required per eigenvalue the flop count for Francis' double step QR algorithm to compute *all* eigenvalues of a *real* Hessenberg matrix is $12n^3$. If also the eigenvector matrix is accumulated the two additional statements have to be inserted into Algorithm 4.5. After step 15 we have

---

1: $Q_{1:n,k+1:k+3} := Q_{1:n,k+1:k+3}P$;

---

and after step 23 we introduce

1: $Q_{1:n,p-1:p} := Q_{1:n,p-1:p}P$;

which costs another $12n^3$ flops.

We earlier gave the estimate of $6n^3$ flops for a Hessenberg QR step, see Algorithm 4.2. If the latter has to be spent in *complex* arithmetic then the single shift Hessenberg QR algorithm is more expensive than the double shift Hessenberg QR algorithm that is executed in real arithmetic.

Remember that the reduction to Hessenberg form costs $\frac{10}{3}n^3$ flops without forming the transformation matrix and $\frac{14}{3}n^3$ if this matrix is formed.

4.6 The symmetric tridiagonal QR algorithm

The QR algorithm can be applied straight to Hermitian or symmetric matrices. By (4.1) we see that the QR algorithm generates a sequence $\{A_k\}$ of symmetric matrices. Taking into account the symmetry, the performance of the algorithm can be improved considerably. Furthermore, from Theorem 2.14 we know that Hermitian matrices have a real spectrum. Therefore, we can restrict ourselves to single shifts.

4.6.1 Reduction to tridiagonal form

The reduction of a full Hermitian matrix to Hessenberg form produces a Hermitian Hessenberg matrix, which (up to rounding errors) is a real symmetric tridiagonal matrix. Let us consider how to take into account symmetry. To that end let us consider the first reduction step that introduces $n-2$ zeros into the first column (and the first row) of $A = A^* \in \mathbb{C}^{n\times n}$. Let

$$P_1 = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & I_{n-1} - 2\mathbf{u}_1\mathbf{u}_1^* \end{bmatrix}, \qquad \mathbf{u}_1 \in \mathbb{C}^n, \quad \|\mathbf{u}_1\| = 1.$$

Then,

$$\begin{aligned} A_1 := P_1^*AP_1 &= (I - 2\mathbf{u}_1\mathbf{u}_1^*)A(I - 2\mathbf{u}_1\mathbf{u}_1^*) \\ &= A - \mathbf{u}_1\underbrace{(2\mathbf{u}_1^*A - 2(\mathbf{u}_1^*A\mathbf{u}_1)\mathbf{u}_1^*)}_{\mathbf{v}_1^*} - \underbrace{(2A\mathbf{u}_1 - 2\mathbf{u}_1(\mathbf{u}_1^*A\mathbf{u}_1))}_{\mathbf{v}_1}\mathbf{u}_1^* \\ &= A - \mathbf{u}_1\mathbf{v}_1^* - \mathbf{v}_1\mathbf{u}_1^*. \end{aligned}$$

In the k-th step of the reduction we similarly have

$$A_k = P_k^*A_{k-1}P_k = A_{k-1} - \mathbf{u}_{k-1}\mathbf{v}_{k-1}^* - \mathbf{v}_{k-1}\mathbf{u}_{k-1}^*,$$

where the last $n-k$ elements of $\mathbf{u}_{k-1}$ and $\mathbf{v}_{k-1}$ are nonzero. Forming

$$\mathbf{v}_{k-1} = 2A_{k-1}\mathbf{u}_{k-1} - 2\mathbf{u}_{k-1}(\mathbf{u}_{k-1}^*A_{k-1}\mathbf{u}_{k-1})$$

costs $2(n-k)^2 + \mathcal{O}(n-k)$ flops. This complexity results from $A_{k-1}\mathbf{u}_{k-1}$. The rank-2 update of A_{k-1},

$$A_k = A_{k-1} - \mathbf{u}_{k-1}\mathbf{v}_{k-1}^* - \mathbf{v}_{k-1}\mathbf{u}_{k-1}^*,$$

requires another $2(n-k)^2+\mathcal{O}(n-k)$ flops, taking into account symmetry. By consequence, the transformation to tridiagonal form can be accomplished in

$$\sum_{k=1}^{n-1}\left(4(n-k)^2 + \mathcal{O}(n-k)\right) = \frac{4}{3}n^3 + \mathcal{O}(n^2)$$

floating point operations.

4.6.2 The tridiagonal QR algorithm

In the symmetric case the Hessenberg QR algorithm becomes a tridiagonal QR algorithm. This can be executed in an **explicit** or an **implicit** way. In the explicit form, a QR step is essentially

---

1: Choose a shift μ
2: Compute the QR factorization $A - \mu I = QR$
3: Update A by $A = RQ + \mu I$.

Of course, this is done by means of plane rotations and by respecting the symmetric tridiagonal structure of A.

In the more elegant implicit form of the algorithm we first compute the first Givens rotation $G_0 = G(1, 2, \vartheta)$ of the QR factorization that zeros the $(2, 1)$ element of $A - \mu I$,

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a_{11} - \mu \\ a_{21} \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}, \qquad c = \cos(\vartheta_0), \quad s = \sin(\vartheta_0). \tag{4.12}$$

Performing a similary transformation with G_0 we have $(n = 5)$

$$G_0^* A G_0 = A' = \begin{bmatrix} \times & \times & + & & \\ \times & \times & \times & & \\ + & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

Similar as with the double step Hessenberg QR algorithm we chase the bulge down the diagonal. In the 5×5 example this becomes

$$A \xrightarrow[= G(1,2,\vartheta_0)]{G_0} \begin{bmatrix} \times & \times & + & & \\ \times & \times & \times & & \\ + & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix} \xrightarrow[= G(2,3,\vartheta_1)]{G_1} \begin{bmatrix} \times & \times & 0 & & \\ \times & \times & \times & + & \\ 0 & \times & \times & \times & \\ & + & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

$$\xrightarrow[= G(3,4,\vartheta_2)]{G_2} \begin{bmatrix} \times & \times & 0 & & \\ \times & \times & \times & & \\ & \times & \times & \times & + \\ & 0 & \times & \times & \times \\ & & + & \times & \times \end{bmatrix} \xrightarrow[= G(4,5,\vartheta_3)]{G_3} \begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & 0 \\ & & \times & \times & \times \\ & & 0 & \times & \times \end{bmatrix} = \overline{A}.$$

The full step is given by

$$\overline{A} = Q^* A Q, \qquad Q = G_0 \, G_1 \, \cdots \, G_{n-2}.$$

Because $G_k \mathbf{e}_1 = \mathbf{e}_1$ for $k > 0$ we have

$$Q \, \mathbf{e}_1 = G_0 \, G_1 \cdots G_{n-2} \, \mathbf{e}_1 = G_0 \, \mathbf{e}_1.$$

Both explicit and implicit QR step form the same first plane rotation G_0. By referring to the Implicit Q Theorem 4.5 we see that explicit and implicit QR step compute *essentially* the same $\overline{A}$.

Algorithm 4.6 Symmetric tridiagonal QR algorithm with implicit Wilkinson shift

Let $T \in \mathbb{R}^{n\times n}$ be a symmetric tridiagonal matrix with diagonal entries $a_1, \ldots, a_n$ and off-diagonal entries $b_2, \ldots, b_n$.
This algorithm computes the eigenvalues $\lambda_1, \ldots, \lambda_n$ of T and corresponding eigenvectors $\mathbf{q}_1, \ldots, \mathbf{q}_n$. The eigenvalues are stored in $a_1, \ldots, a_n$. The eigenvectors are stored in the matrix Q, such that $TQ = Q\,\mathrm{diag}(a_1, \ldots, a_n)$.

$m = n$ /* Actual problem dimension. m is reduced in the convergence check. */
$Q = I_n$;
while $m > 1$ **do**
  $d := (a_{m-1} - a_m)/2$; /* Compute Wilkinson's shift */
  if $d = 0$ **then**
    $s := a_m - |b_m|$;
  else
    $s := a_m - b_m^2/(d + \mathrm{sign}(d)\sqrt{d^2 + b_m^2})$;
  end if
  $x := a(1) - s$; /* Implicit QR step begins here */
  $y := b(2)$;
  for $k = 1$ to $m - 1$ **do**
    if $m > 2$ **then**
      $[c, s] :=$ **givens**(x, y);
    else
      Determine $[c, s]$ such that $\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a_1 & b_2 \\ b_2 & a_2 \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$ is diagonal
    end if
    $w := cx - sy$;
    $d := a_k - a_{k+1}$; $z := (2cb_{k+1} + ds)s$;
    $a_k := a_k - z$; $a_{k+1} := a_{k+1} + z$;
    $b_{k+1} := dcs + (c^2 - s^2)b_{k+1}$;
    $x := b_{k+1}$;
    if $k > 1$ **then**
      $b_k := w$;
    end if
    if $k < m - 1$ **then**
      $y := -sb_{k+2}$; $b_{k+2} := cb_{k+2}$;
    end if
    $Q_{1:n;k:k+1} := Q_{1:n;k:k+1} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$;
  end for/* Implicit QR step ends here */
  if $|b_m| < \varepsilon(|a_{m-1}| + |a_m|)$ **then** /* Check for convergence */
    $m := m - 1$;
  end if
end while

Algorithm 4.6 shows the implicit symmetric tridiagonal QR algorithm. The shifts are chosen acording to Wilkinson. An issue not treated in this algorithm is **deflation**. Deflation is of big practical importance. Let us consider the following 6×6 situation

$$T = \begin{bmatrix} a_1 & b_2 & & & & \\ b_2 & a_2 & b_3 & & & \\ & b_3 & a_3 & 0 & & \\ & & 0 & a_4 & b_5 & \\ & & & b_5 & a_5 & b_6 \\ & & & & b_6 & a_6 \end{bmatrix}.$$

The shift for the next step is determined from elements a_5, a_6, and b_6. According to (4.12) the first plane rotation is determined from the shift and the elements a_1 and b_1. The implicit shift algorithm then chases the bulge down the diagonal. In this particular situation, the procedure finishes already in row/column 4 because $b_4 = 0$. Thus the shift which is an approximation to an eigenvalue of the second block (rows 4 to 6) is applied to the wrong first block (rows 1 to 3). Clearly, this shift does not improve convergence.

If the QR algorithm is applied in its explicit form, then still the first block is not treated properly, i.e. with a (probably) wrong shift, but at least the second block is diagonalized rapidly.

Deflation is done as indicated in Algorithm 4.6:

$$\textbf{if} \quad |b_k| < \varepsilon(|a_{k-1}| + |a_k|) \quad \textbf{then} \quad \text{deflate.}$$

Deflation is particularly simple in the symetric case since it just means that a tridiagonal eigenvalue problem decouples in two (or more) smaller tridiagonal eigenvalue problems. Notice, however, that the eigenvectors are still n elements long.

4.7 Research

Still today the QR algorithm computes the Schur form of a matrix and is by far the most popular approach for solving dense nonsymmetric eigenvalue problems. Multishift and aggressive early deflation techniques have led to significantly more efficient sequential implementations of the QR algorithm during the last decade. For a brief survey and a discussion of the parallelization of the QR algorithm, see [7].

The three steps of the presented symmetric QR algorithm are (1) reducion of the original matrix to tridiagonal form, (2) computation of the eigenpairs of the tridiagonal matrix, and (3) back-transformation of the eigenvectors. In the ELPA project the first step has been successfully replaced by a two-stage procedure: transformation full to banded, and banded to tridiagonal. This approach improves the utilization of memory hierarchies [8, 3].

4.8 Summary

The QR algorithm is a very powerful algorithm to stably compute the eigenvalues and (if needed) the corresponding eigenvectors or Schur vectors. All steps of the algorithm cost $\mathcal{O}(n^3)$ floating point operations, see Table 4.1. The one exception is the case where only eigenvalues are desired of a symmetric tridiagonal matrix. The linear algebra software package LAPACK [1] contains subroutines for all possible ways the QR algorithm may be employed.

	nonsymmetric case		symmetric case	
	without Schurvectors	with	without eigenvectors	with
transformation to Hessenberg/tridiagonal form	$\frac{10}{3}n^3$	$\frac{14}{3}n^3$	$\frac{4}{3}n^3$	$\frac{8}{3}n^3$
real double step Hessenberg/tridiagonal QR algorithm (2 steps per eigenvalues assumed)	$\frac{20}{3}n^3$	$\frac{50}{3}n^3$	$24n^2$	$6n^3$
total	$10n^3$	$25n^3$	$\frac{4}{3}n^3$	$9n^3$

Table 4.1: Complexity in flops to compute eigenvalues and eigenvectors/Schur vectors of a real $n \times n$ matrix

We finish by repeating, that the QR algorithm is a method for *dense* matrix problems. The reduction of a sparse matrix to tridiagonal or Hessenberg form produces **fill-in**, thus destroying the sparsity structure which one almost always tries to preserve.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide – Release 2.0*, SIAM, Philadelphia, PA, 1994. (Software and guide are available from Netlib at URL http://www.netlib.org/lapack/).

[2] P. Arbenz and G. H. Golub, *Matrix shapes invariant under the symmetric QR algorithm*, Numer. Linear Algebra Appl., 2 (1995), pp. 87–93.

[3] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, and P. Willems, *Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem*, J. Comput. Sci., 2 (2011), pp. 272–278.

[4] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[5] J. G. F. Francis, *The QR transformation – Parts 1 and 2*, Comput. J., 4 (1961-1962), pp. 265–271 and 332–345.

[6] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 2nd ed., 1989.

[7] B. Kågström, D. Kressner, and M. Shao, *On aggressive early deflation in parallel variants of the QR algorithm*, in Applied Parallel and Scientific Computing (PARA 2010), K. Jónasson, ed., Heidelberg, 2012, Springer, pp. 1–10. (Lecture Notes in Computer Science, 7133).

[8] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, and H. Lederer, *The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science*, J. Phys.: Condens. Matter, 26 (2014), p. 213201.

[9] B. N. Parlett, *The QR algorithm*, Computing Sci. Eng., 2 (2000), pp. 38–42.

[10] H. Rutishauser, *Solution of eigenvalue problems with the LR-transformation*, NBS Appl. Math. Series, 49 (1958), pp. 47–81.

[11] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.