Topic 6: QR-Iteration with Shift

In this exercise we want to enhance the QR-Iteration from the previous exercise by inclusion of a clever shift-strategy. The *QR-Iteration with Shift* shall then be applied for symmetric and non-symmetric matrices. Since this exercise requires a successful implementation of the QR iteration, it is mandatory to complete the previous exercise.

From now on let $A \in \mathbb{R}^{n\times n}$ be diagonalisable with eigenvalues λ_i. We assume that all eigenvalues are of distinct magnitude and w.l.o.g. ordered:

$$|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n|$$

The speed of convergence of the QR-iteration depends on the ratios $|\lambda_{i+1}/\lambda_i|$ of successive eigenvalues. If the ratio is close to 0 then it converges quickly, if it is close to 1 then it slows down considerably.

Similarly to the trick of the shifted inverse iteration, one can here as well introduce a Shift and thereby speed up the convergence — at least for one of the eigenvalues at a time. Let $\mu \in \mathbb{R}$ be an approximation of the eigenvalue λ_i such that

$$|\mu - \lambda_i| \ll |\mu - \lambda_j| \quad \forall j \neq i.$$

The QR-Iteration applied to the matrix $A - \mu I$ will then quickly converge for the first eigenvector corresponding to the eigenvalue λ_i:

Algorithm 1 QR-Iteration with Shift

1: $A_1 := A$, $Q_1 := I$
2: **for** $i = 1, 2, \ldots$ **do**
3: Determine μ
4: Compute a QR-decomposition $(A_i - \mu I) =: \hat{Q}_{i+1} R_{i+1}$
5: $A_{i+1} := R_{i+1}\hat{Q}_{i+1} + \mu I$
6: $Q_{i+1} := Q_i \hat{Q}_{i+1}$
7: **end for**

Algorithm 1 has the following properties:

- A_i converges to an upper triangular matrix, but in general $A_i \neq Q_i^T A Q_i =: \hat{A}_i$.
- The diagional entries of $\hat{A}_i$ are approximations of the eigenvalues of A.
- The shift parameter μ can be chosen differently in each step (without increasing the computational complexity)

In the following we assume that the matrix A is already given in upper Hessenberg form. The shift parameter μ in Algorithm 1 shall now be chosen in such a way that it approximates the same eigenvalue in every step (but we do not know the eigenvalues, how should we do this ?). After a

few steps the matrix $M := \hat{A}_i$ should be of the form

$$M = \left[\begin{array}{ccc|ccccc} m_{1,1} & \dots & \dots & \dots & \dots & \dots & m_{1,n} \\ m_{2,1} & \ddots & & & & & \vdots \\ & \ddots & m_{k,k} & & & & \vdots \\ \hline & & \mathbf{0} \approx m_{k+1,k} & m_{k+1,k+1} & & & \vdots \\ & & & m_{k+2,k+1} & \ddots & & \vdots \\ & & & & \ddots & m_{n-1,n-1} & \vdots \\ & & & & & m_{n,n-1} & m_{n,n} \end{array}\right] \approx \left[\begin{array}{c|c} M_{11} & M_{12} \\ \hline 0 & M_{22} \end{array}\right] \qquad (1)$$

for some $k \in \{1, \dots, n\}$. If the entry $m_{k+1,k}$ is close enough to 0, then the problem can be split into two subproblems for the matrices $M_{11} \in \mathbb{R}^{k \times k}$ and $M_{22} \in \mathbb{R}^{(n-k)\times(n-k)}$.

M_{11} and M_{22} are both of upper Hessenberg form. The algorithm can thus be applied recursively for M_{11} and M_{22}. From the solutions of the subproblems we obtain the solution for M. We apply Algorithm 1 until the subproblems are of size 1×1 and thus trivial. An important criterion is to decide whether or not entry $m_{k+1,k}$ in (1) is small enough. In the following the problem is subdivided into subproblems if for given $\varepsilon > 0$

$$|m_{k+1,k}| < \varepsilon. \qquad (2)$$

Practical Exercise

Copy all files from `../../vorlagen/EWP/ex6new` into your directory `EWP`

```
cp -r ../../vorlagen/EWP/ex6new .
```

Complete in the file `qr_shift.cc` the function

```
void  qr_shift_iteration( const TDenseMatrix * A, TDenseMatrix * A_i,
                          TDenseMatrix * Q, const real eps );
```

which is supposed to perform the QR-iteration with shift. Exploit the recursive structure in your program. The input matrix `A` shall not be changed by your procedure. The result A_i shall be stored in `A_i` and the orthogonal factor Q_i in `Q`. The parameter `eps` controls the splitting criterion (2).

Compile your program by `make`.
Test: Start your programm `qr_shift` and test it for the matrices A und B. Analyse the output (on the screen).