The QR-**Algorithm** is an eigenvalue algorithm. Thus, it is used to compute eigenvalues and eigenvectors of a matrix $A \in \mathbb{R}^{n\times n}$. It produces a sequence of matrices $(A_k)_{k\in\mathbb{N}}$. All A_k are similar to A and thus have the same eigenvalues. The iteration is defined as follows:

$$
\begin{aligned}
&A_0 = A \in \mathbb{R}^{n\times n}\\
&\textbf{for } k = 0, \dots, \infty :\\
&\qquad Q_{k+1}R_{k+1} := A_k \qquad (QR \text{ decomposition})\\
&\qquad A_{k+1} := R_{k+1}Q_{k+1}
\end{aligned}
$$

If the absolute values of the eigenvalues of A are distinct, one can show that $A_\infty := \lim_{k\to\infty} A_k$ is a diagonal matrix. In this case, the eigenvalues of A are the diagonal elements of A_∞.

Task:

1. Implement the QR eigenvalue algorithm as a function `eig(A,m)`. The function shall take as input a matrix $A \in \mathbb{R}^{n\times n}$ and a maximum iteration number $m \in \mathbb{N}$. It shall return the diagonal of the last iterate A_m. For the QR decomposition you can use the Gram-Schmidt algorithm from previous sheets which we have implemented as a function `QR(A)` or an appropriate Python routine.

   Hint: You can access the diagonal of a `numpy.array` by `A.diagonal()`.

2. Test your algorithm on a random matrix $A \in \mathbb{R}^{n\times n}$. In order to generate such a random matrix use the following code snippet:
   ```
   def A_gen(n):
      from numpy as np
      from scipy.linalg import qr
      A = np.random.rand(n,n)
      Q, R = qr(A)
      Lambda = np.diag(np.arange(1,n+1))
      A = Q @ (Lambda @ Q.T)
   return A
   ```

3. Find a routine in `Scipy` to compute the eigenvalues and -vectors of a matrix. Test the routine on multiple examples, especially for higher dimensions. Compare to your algorithm.

Solution:

```python
import numpy as np


def qr_factor(A):
    """
    Computes a QR-decomposition of a (mxn)-matrix with m>=n via Gram-Schmidt.

    Parameters
    ----------
    A : (mxn) matrix with m>=n

    Returns
    -------
    Q : (mxn) with orthonormal columns
    R : (nxn) upper triangular matrix
    """
    m, n = A.shape
    R = np.zeros((n, n))
```

```python
    Q = np.zeros((m, n))

    R[0, 0] = np.linalg.norm(A[:, 0])
    Q[:, 0] = A[:, 0] / R[0, 0]

    for k in range(1, n):
        for l in range(0, k):
            R[l, k] = A[:, k] @ Q[:, l]
        q = A[:, k] - Q @ R[:, k]
        R[k, k] = np.linalg.norm(q)
        Q[:, k] = q / R[k, k]

    return Q, R


def eig(A, m=50, qr="own"):
    """
    Computes the eigenvalues of a square matrix via QR eigenvalue algorithm

    Parameters
    ----------
    A : (nxn) matrix with *distinct* eigenvalues
    m : iteration number
    qr : optional parameter to switch between own qr and scipy qr

    Returns
    -------
    d : diagonal of the last QR-iterate
    """
    if qr == "own":
        qr = qr_factor
    else:


    for k in range(m):
        Q, R = qr(A)
        A = R @ Q
    return A.diagonal()


def A_gen(n):
    import numpy as np
    from scipy.linalg import qr
    A = np.random.rand(n, n)
    Q, R = qr(A)
    eigvals = - np.linspace(0, 1, n)  # np.arange(1, n+1)
    Lambda = np.diag(eigvals)
    A = Q @ Lambda @ Q.T
    return A


if __name__ == "__main__":

    # 2 test
    n = 10
    qr = ""
    A = A_gen(n)
    for m in [10, 50, 75, 150]:
        print("number of iterations:\n m =", m, "\napproximate eigenvalues:\n",
              np.sort(np.round(eig(A, m, qr=qr), 6)), "\n")
    # 3 compare to numpy.linalg
    print("--> compare to numpy.linalg:\n", np.sort(np.linalg.eigvals(A)))
```