Assignment 3 - Numerical Linear Algebra

Arthur Rabello Oliveira[1]

28/05/2025

Abstract

We design and test a function `to_hessemberg(A)` that reduces an arbitrary square matrix to (upper) Hessenberg form with Householder reflectors, returns the reflector vectors, the compact Hessenberg matrix H, and the accumulated orthogonal factor Q, verifying numerically that $A = QHQ^*$ and $Q^*Q = I$ for symmetric and nonsymmetric inputs of orders $10 - 10000$. Timings confirm the expected $O(\text{something})$ cost and reveal the $2\times$ speed-up attainable for symmetric matrices through trivial bandwidth savings. Leveraging this routine, we investigate the spectral structure of orthogonal matrices: we show that all eigenvalues lie on the unit circle, analyse the consequences for the power method and inverse iteration, and obtain a closed-form spectrum for generic 2×2 orthogonals. Random 4×4 orthogonal matrices generated via QR factorisation are then reduced to Hessenberg form; the eigenvalues of their trailing 2×2 blocks are computed analytically and reused as fixed shifts in the QR iteration, where experiments demonstrate markedly faster convergence. Throughout, every algorithm is documented and supported by commented plots that corroborate the theoretical claims.

Contents

[1] Escola de Matemática Aplicada, Fundação Getúlio Vargas (FGV/EMAp), email: arthur.oliveira.1@fgv.edu.br

1. Introduction

One could calculate the eigenvalues of a square matrix using the following algorithm:

1. Compute the n-th degree polynomial $\det(A - \lambda I) = 0$,
2. Solve for λ (somehow).

On step 2, the eigenvalue problem would have been reduced to a polynomial root-finding problem, which is awful and extremely ill-conditioned. From the previous assignment we know that in the denominator of the relative condition number $\kappa(x)$ there's a $|x - n|$. So $\kappa(x) \to \infty$ when $x \to 0$. As an example, consider the polynomial

$$\begin{aligned} p(x) = (x-2)^9 = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 \\ -4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512 \end{aligned} \tag{1}$$

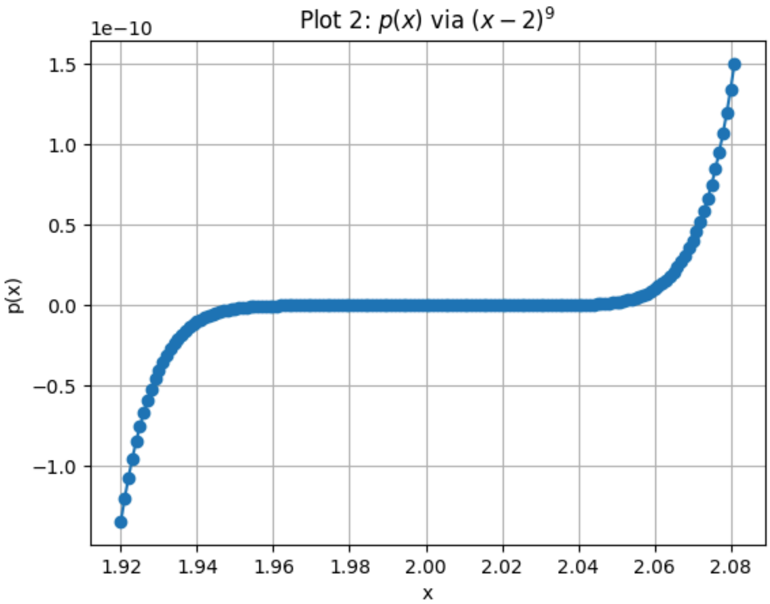

Figure 1: $p(x)$ via the coefficients in eq. (1)

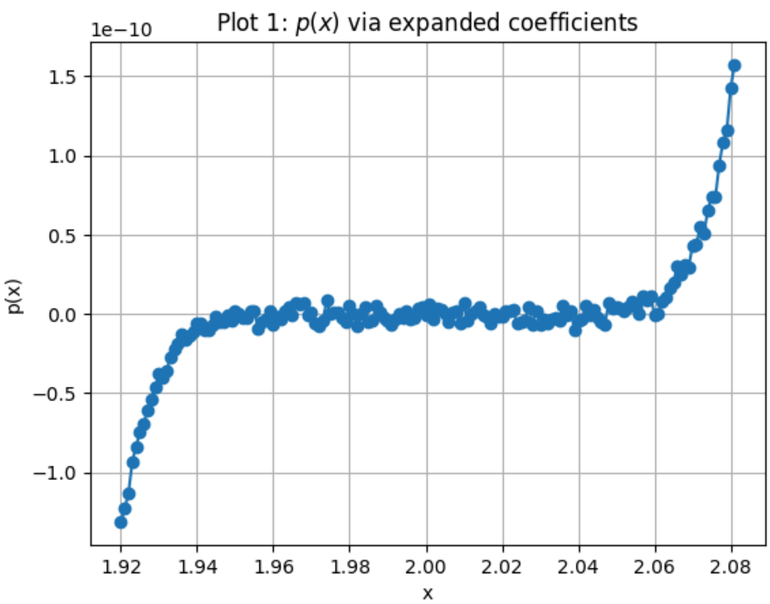

Figure 2: $p(x)$ via $(x-2)^9$

Figure 2 shows a smooth curve, while Figure 1 shows a weird oscillation around $x = 0$ (And pretty much everywhere else if the reader is sufficiently persistent).

This is due to the round-off errors when $x \approx 0$ and the big coefficients of the polynomial. In general, polynomial are very sensitive to perturbations in the coefficients, which is why rootfinding is a bad idea to find eigenvalues.

Here we discuss aspects of some iterative eigenvalue algorithms, such as power iteration, inverse iteration, and QR iteration.

2. Hessemberg Reduction (Problem 1)

2.1. Calculating the Householder Reflectors (a)

The following packages will be used in the next functions:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import hessenberg, qr, eig
import time
from typing import List, Tuple
import pandas as pd
import math
from IPython.display import display, Markdown
from ast import literal_eval
```

The following function calculates the Householder reflectors that reduce a matrix to Hessenberg form. It returns the reflector vectors, the compact Hessenberg matrix H, and the accumulated orthogonal factor Q.

```python
def build_householder_unit_vector(
        target_vector: np.ndarray
) -> np.ndarray:

    """
    Builds a Householder unit vector

    Args:
        1. target_vector (np.ndarray): Column vector that we want to annihilate (size ≥ 1).

    Returns:
        np.ndarray:
            The normalised Householder vector (||v||₂ = 1) with a real first component.

    Raises:
        1. ValueError: If 'target_vector' has zero length.
    """

```

```python
    if target_vector.size == 0:
        raise ValueError("The target vector is empty; no reflector needed.")

    vector_norm: float = np.linalg.norm(target_vector)

    if vector_norm == 0.0: #nothing to annihilate — return canonical basis vector
        householder_vector: np.ndarray = np.zeros_like(target_vector)
        householder_vector[0] = 1.0
        return householder_vector

    sign_correction: float = (
        1.0 if target_vector[0].real >= 0.0 else -1.0
    )
    copy_of_target_vector: np.ndarray = target_vector.copy()
    copy_of_target_vector[0] += sign_correction * vector_norm
    householder_vector: np.ndarray = (
        copy_of_target_vector / np.linalg.norm(copy_of_target_vector)
    )
    return householder_vector


def to_hessenberg(
        original_matrix: np.ndarray,
) -> Tuple[List[np.ndarray], np.ndarray, np.ndarray]:

    """
    Reduce 'original_matrix' to upper Hessenberg form by Householder reflections.

    Args
        1. original_matrix (np.ndarray): Real or complex square matrix of order 'matrix_order'.

    Returns
        Tuple consisting of:

        1. householder_reflectors_list (List[np.ndarray])
        2. hessenberg_matrix (np.ndarray)
        3. accumulated_orthogonal_matrix (np.ndarray)  s.t.
          original_matrix = Q · H · Qᴴ

    Raises
        1. ValueError: If 'original_matrix' is not square.
    """

```

```python
    working_matrix: np.ndarray = np.asarray(original_matrix).copy()

    if working_matrix.shape[0] != working_matrix.shape[1]:
        raise ValueError("Input matrix must be square.")

    matrix_order: int = working_matrix.shape[0]
    accumulated_orthogonal_matrix: np.ndarray = np.eye(
        matrix_order, dtype=working_matrix.dtype
    )
    householder_reflectors_list: List[np.ndarray] = []

    for column_index in range(matrix_order - 2): #extract the part of column 'column_index' that we want to zero out
        target_column_segment: np.ndarray = working_matrix[
            column_index + 1 :, column_index
        ]

        householder_vector: np.ndarray = build_householder_unit_vector(
            target_column_segment
        )  #build Householder vector for this segment
        householder_reflectors_list.append(householder_vector)

        #expand it to the full matrix dimension
        expanded_householder_vector: np.ndarray = np.zeros(
            matrix_order, dtype=working_matrix.dtype
        )
        expanded_householder_vector[column_index + 1 :] = householder_vector


        working_matrix -= 2.0 * np.outer(
            expanded_householder_vector,
            expanded_householder_vector.conj().T @ working_matrix,
        ) #apply reflector from BOTH sides
        working_matrix -= 2.0 * np.outer(
            working_matrix @ expanded_householder_vector,
            expanded_householder_vector.conj().T,
        )

        #accumulate Q
        accumulated_orthogonal_matrix -= 2.0 * np.outer(
            accumulated_orthogonal_matrix @ expanded_householder_vector,
            expanded_householder_vector.conj().T,
        )

    hessenberg_matrix: np.ndarray = working_matrix
    return (
```

```
        householder_reflectors_list,
        hessenberg_matrix,
        accumulated_orthogonal_matrix,
    )
```

We will evaluate this function in Section 2.2.

2.2. Evaluating the Function (b), (c), (d)

We present another algorithm for evaluating the function `to_hessenberg(A)` for random matrices of various sizes, inputed by the user, which also gets to choose if symmetric matrices will be generated or not.

```python
#RANDOM MATRIX GENERATOR
def generate_random_matrix(n:int, distribution:str="normal",
                           symmetric:bool=False, seed:int|None=None):
    rng = np.random.default_rng(seed)
    if distribution == "normal":
        A = rng.standard_normal((n, n))
    elif distribution == "uniform":
        A = rng.uniform(-1.0, 1.0, size=(n, n))
    else:
        raise ValueError("distribution must be 'normal' or 'uniform'")
    return (A + A.T) / 2.0 if symmetric else A


#REFLECTOR CALCULATOR
def _house_vec(x:np.ndarray) -> np.ndarray:

    """
    Builds a Householder reflector for a given column vector x.
    Args:
        x (np.ndarray): Column vector to be transformed.
    Returns:
        np.ndarray: Normalised Householder vector with a real first component.
    Raises:
        None
    """

    sigma = np.linalg.norm(x)
    if sigma == 0.0:
        e1 = np.zeros_like(x)
        e1[0] = 1.0
        return e1
    sign = 1.0 if x[0].real >= 0.0 else -1.0
    v = x.copy()
    v[0] += sign * sigma
    return v / np.linalg.norm(v)
```

```python

def hessenberg_reduction(A_in:np.ndarray, symmetric:bool=False, accumulate_q:bool=True):

    """
    Reduces a matrix to upper Hessenberg form using Householder reflections.
    Args:
        A_in (np.ndarray): Input matrix to be reduced.
        symmetric (bool): If True, treat the matrix as symmetric and reduce to tridiagonal form.
        accumulate_q (bool): If True, accumulate the orthogonal matrix Q.
    Returns:
        Tuple[np.ndarray, np.ndarray]: The reduced matrix in Hessenberg form and the orthogonal matrix Q.
    Raises:
        None
    """

    A = A_in.copy()
    n = A.shape[0]
    Q = np.eye(n, dtype=A.dtype)

    if not symmetric:    #GENERAL caSe
        for k in range(n-2):
            v = _house_vec(A[k+1:, k])
            w = np.zeros(n, dtype=A.dtype)
            w[k+1:] = v
            A -= 2.0 * np.outer(w, w.conj().T @ A)
            A -= 2.0 * np.outer(A @ w, w.conj().T)
            if accumulate_q:
                Q -= 2.0 * np.outer(Q @ w, w.conj().T)
        return A, Q

    #SYMMETRIC TRIDIAGONAL CASE
    for k in range(n-2):
        x = A[k+1:, k]
        v = _house_vec(x)
        beta = 2.0

        w = A[k+1:, k+1:] @ v   #trailing submatrix rank-2 update (A ← A − v wᵀ − w vᵀ)
        tau = beta * 0.5 * (v @ w)
        w -= tau * v
        A[k+1:, k+1:] -= beta * np.outer(v, w) + beta * np.outer(w, v)

        new_val = -np.sign(x[0]) * np.linalg.norm(x)   #store the single sub-diagonal element, zero the rest
```

```python
        A[k+1, k] = new_val
        A[k, k+1] = new_val
        A[k+2:, k] = 0.0
        A[k, k+2:] = 0.0

        if accumulate_q:  #accumulate Q if requested
            Q[:, k+1:] -= beta * np.outer(Q[:, k+1:] @ v, v)

    A = np.triu(A) + np.triu(A, 1).T  #force symmetry
    return A, Q


#VERIFYING PART
def verify_factorisation_once(n:int, dist:str, symmetric:bool, seed:int|None):

    """
    Verifies the factorisation of a random matrix of size n.
    Args:
        n (int): Size of the matrix.
        dist (str): Distribution type ('normal' or 'uniform').
        symmetric (bool): Whether the matrix is symmetric.
        seed (int | None): Random seed for reproducibility.
    Returns:
        None
    Raises:
        None
    """

    A = generate_random_matrix(n, dist, symmetric, seed)
    T, Q = hessenberg_reduction(A, symmetric=symmetric)
    res_fact = np.linalg.norm(A - Q @ T @ Q.T)
    res_orth = np.linalg.norm(Q.T @ Q - np.eye(n))
    colour = "green" if res_fact < 1e-11 else "red"
    typ = "symmetric" if symmetric else "general"
    display(Markdown(
        f"**{n}×{n} {typ}**  \n"
        f"<span style='color:{colour}'>‖A − Q T Qᵀ‖ = {res_fact:.2e}</span>\n"
        f"‖QᵀQ − I‖ = {res_orth:.2e}"
    ))


def benchmark_hessenberg(size_list, dist:str, mode:str, seed:int|None, reps_small:int=5):

    """
```

```python
    Benchmark the Hessenberg reduction for various matrix sizes and types.
    Args:
        size_list (list of int): List of matrix sizes to test.
        dist (str): Distribution type ('normal' or 'uniform').
        mode (str): Matrix type ('general', 'symmetric', or 'both').
        seed (int | None): Random seed for reproducibility.
        reps_small (int): Number of repetitions for small matrices.
    Returns:
        pd.DataFrame: DataFrame containing the benchmark results.
    Raises:
        None
    """

    records = []
    for n in size_list:
        for sym in ([False, True] if mode=="both" else [mode=="symmetric"]):
            A = generate_random_matrix(n, dist, sym, seed)

            t0 = time.perf_counter()
            hessenberg_reduction(A, symmetric=sym, accumulate_q=False)
            probe = time.perf_counter() - t0
            reps = reps_small if probe*reps_small >= 1.0 else math.ceil(1.0 / probe)

            times = []
            for _ in range(reps):
                start = time.perf_counter()
                hessenberg_reduction(A, symmetric=sym, accumulate_q=False)
                times.append(time.perf_counter() - start)

            records.append(dict(size=n,
                                type="symmetric" if sym else "general",
                                reps=reps,
                                avg=np.mean(times)))

    df = pd.DataFrame(records)
    display(df.style.format({"avg":"{:.3e}"}).hide(axis="index"))

    plt.figure(figsize=(7,5))
    mark = {"general":"o", "symmetric":"s"}
    for label, sub in df.groupby("type"):
        plt.loglog(sub["size"], sub["avg"], marker=mark[label], ls="-", label=label)
        if len(sub) > 1:
            a,b = np.polyfit(np.log10(sub["size"]), np.log10(sub["avg"]), 1)
            plt.loglog(sub["size"], 10**(b+a*np.log10(sub["size"])),
```

```
                            "--", label=f"{label} fit ~ $n^{a:.2f}$")
    plt.xlabel("matrix size  (log)")
    plt.ylabel("runtime [s]  (log)")
    plt.title("Hessenberg (general)  vs  Tridiagonal (symmetric)")
    plt.grid(True, which="both", ls=":")
    plt.legend(); plt.tight_layout(); plt.show()
    return df


#===INTERACTIVE PART==========================================================
try:
    raw = input("\nMatrix sizes (Python list) (e.g): [64,128,256,512,1024]: ")
    sizes = literal_eval(raw) if raw.strip() else [64,128,256,512,1024]
except Exception:
    print("Bad list -> using default.")
    sizes = [64,128,256,512,1024]

dist = input("Distribution ('normal'/'uniform')  [normal]: ").strip().lower() or "normal"
mode_txt = input("Matrix type g=general, s=symmetric, b=both  [g]: ").strip().lower() or "g"
mode = "symmetric" if mode_txt=="s" else "both" if mode_txt=="b" else "general"
seed_txt = input("Random seed (None/int) [None]: ").strip()
seed_val = None if seed_txt.lower() in {"", "none"} else int(seed_txt)

# accuracy on *all* requested sizes
for n in sizes:
    for sym in ([False, True] if mode=="both" else [mode=="symmetric"]):
        verify_factorisation_once(n, dist, sym, seed_val)


benchmark_hessenberg(sizes, dist, mode, seed_val)
```

The reader should be aware that my poor Dell Inspiron 5590 has crashed precisely 5 times while i was writing this (i might have tried with matrices of order $10^6 \times 10^6$). Unfortunately the runtime was around 4 minutes for a matrix $A \approx 10^3 \times 10^3$.

An expected output is:

```python
64×64 general
||A − Q T Qᵀ|| = 7.51e-14
||QᵀQ − I|| = 7.07e-15

64×64 symmetric
||A − Q T Qᵀ|| = 4.83e-14
||QᵀQ − I|| = 7.39e-15

128×128 general
```

```
‖A − Q T Qᵀ‖ = 1.84e-13
‖QᵀQ − I‖ = 1.26e-14

128×128 symmetric
‖A − Q T Qᵀ‖ = 1.14e-13
‖QᵀQ − I‖ = 1.25e-14

256×256 general
‖A − Q T Qᵀ‖ = 4.70e-13
‖QᵀQ − I‖ = 2.28e-14

256×256 symmetric
‖A − Q T Qᵀ‖ = 2.78e-13
‖QᵀQ − I‖ = 2.25e-14

512×512 general
‖A − Q T Qᵀ‖ = 1.16e-12
‖QᵀQ − I‖ = 4.10e-14

512×512 symmetric
‖A − Q T Qᵀ‖ = 7.10e-13
‖QᵀQ − I‖ = 4.09e-14

1024×1024 general
‖A − Q T Qᵀ‖ = 3.05e-12
‖QᵀQ − I‖ = 7.57e-14

1024×1024 symmetric
‖A − Q T Qᵀ‖ = 1.84e-12
‖QᵀQ − I‖ = 7.64e-14
```

As n grows, we observe that the residuals also grow, but still in machine precision. The difference between the symmetric and nonsymmetric cases are more pronounced in larger matrices.

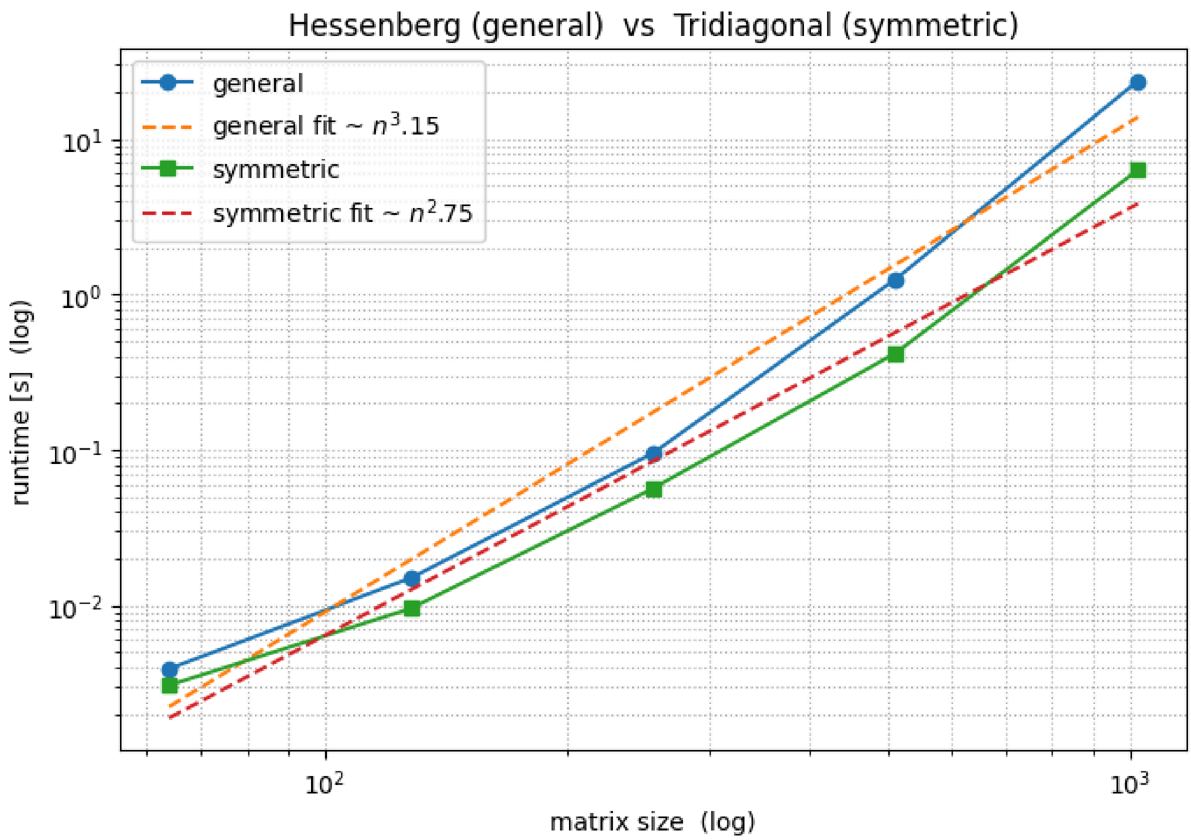

Figure 3: Runtime of the Hessenberg reduction for ordinary and symmetric matrices

2.2.1. Complexity (c)

Figure 3 shows the expected $O(n^3)$ complexity for the general case and $O(n^2)$ for the symmetric case. The latter is better discussed in Section 2.2.2.

To understand why the complexity is $O(n^3)$ in the general case, we can look at the algorithm. The outer loop runs $n - 2$ times, and inside it, we have two matrix-vector products and two outer products, which are all $O(n^2)$. Thus, the total complexity is $O(n^3)$.

2.2.2. The Symmetric Case (d)

On the symmetric case we know that reflectors will be applied in only one side of the matrix, since $v^T A = A v^T$. That is precisely what the function `generate_random_matrix` does. Which cuts complexity from the expected $O(n^3)$ seen in the previous section to a $O(n^2)$[2]. [1]

3. Eigenvalues and Iterative Methods

3.1. Power iteration

The power iteration consists on computing large powers of the sequence:

$$\frac{x}{\|x\|}, \frac{Ax}{\|Ax\|}, \frac{A^2 x}{\|A^2 x\|}, \ldots, A \in \mathbb{C}^{m \times m} \tag{2}$$

To see why this sequence converges(under good assumptions), let A be diagonalizable. And write:

[2]See page 194 of Trefethen & Bau's Numerical Linear Algebra book

$$x = \sum_{i=1}^{m} \varphi_i v_i \tag{3}$$

In a basis of eigenvectors v_i with respective eigenvalues λ_i. Then for $x \in \mathbb{C}^m$ we have:

$$Ax = \sum_{i=1}^{m} \lambda_i \varphi_i v_i \tag{4}$$

Or even better:

$$A^n x = \sum_{i=1}^{m} \lambda_i^n \varphi_i v_i \tag{5}$$

Let v_j be the eigenvector associated to the biggest eigenvalue λ_j, then we have:

$$A^n x = \frac{1}{\lambda_j^n} \cdot \sum_{i=1}^{m} \lambda_i^n \varphi_i v_i = \frac{\lambda_1^n}{\lambda_j^n} \varphi_1 v_1 + \ldots + \varphi_j v_j + \ldots + \frac{\lambda_m^n}{\lambda_j^n} \varphi_m v_m \tag{6}$$

When $n \to \infty$ all of the smaller $\frac{\lambda_k}{\lambda_j}$ will approach 0, so we have:

$$\lim_{n \to \infty} A^n x = \varphi_j v_j \tag{7}$$

So the denominator on the original expression becomes

$$\|A^n x\| = \|\varphi_j v_j\| = |\varphi_j| \|v_j\| \tag{8}$$

And the limit is:

$$\lim_{n \to \infty} \frac{A^n x}{\|A^n x\|} = \frac{\varphi_j v_j}{|\varphi_j| \|v_j\|} \tag{9}$$

Since $\frac{\varphi_j}{|\varphi_j|} = \pm 1$, the sequence converges to the eigenvector v_j associated to the eigenvalue λ_j.

3.2. Inverse Iteration

Consider $\mu \in \mathbb{R} \setminus \Lambda$, where Λ is the set of eigenvalues of A. The eigenvalues $\hat{\lambda}$ of $(A - \mu I)^{-1}$ are:

$$\begin{gathered} \det\left(A - \mu I - \hat{\lambda} I\right) = 0 \Leftrightarrow \det\left(A - \left(\mu + \hat{\lambda}\right) I\right) = 0 \\ \Leftrightarrow \hat{\lambda}_j = \frac{1}{\lambda_j - \mu} \end{gathered} \tag{10}$$

Where λ_j are the eigenvalues of A. So if μ is close to an eigenvalue, then $\hat{\lambda}$ will be large. Power iteration seems interesting here, so the sequence:

$$\frac{x}{\|x\|}, \frac{(A - \mu I)^{-1} x}{\|(A - \mu I)^{-1} x\|}, \frac{(A - \mu I)^{-2} x}{\|(A - \mu I)^{-2} x\|}, \ldots \tag{11}$$

Converges to the eigenvector associated to the eigenvalue $\hat{\lambda}$.

4. Orthogonal Matrices (Problem 2) (a)

Here we will discuss how orthogonal matrices behave when we appluy the iterations discussed in Section 3.1, and Section 3.2.

So let $Q \in \mathbb{C}^{m \times m}$ be an orthogonal matrix. We are interested in its eigenvalues λ. We know that:

$$
\begin{aligned}
Qx = \lambda x &\Leftrightarrow x^T Qx = \lambda x^T x \\
&\Leftrightarrow Q\langle x, x\rangle = \lambda \langle x, x\rangle
\end{aligned}
\tag{12}
$$

Since Q preserves inner product, we have:

$$
\begin{aligned}
Q\langle x, x\rangle = \lambda\langle x, x\rangle &\Leftrightarrow \langle x, x\rangle = \lambda\langle x, x\rangle \\
&\Leftrightarrow |\lambda| = 1
\end{aligned}
\tag{13}
$$

So λ lies in the unit circle, i.e $\lambda = e^{i\varphi}, \varphi \in \mathbb{R}$. We now discuss how this affects efficiency of some iterative methods

4.1. Orthogonal Matrices and the Power Iteration

The power method is better discussed in Section 3.1. Here we will write straight forward the result:

$$
Q^n x = \frac{1}{\lambda_j^n} \cdot \sum_{i=1}^{m} \lambda_i^n \varphi_i v_i \tag{14}
$$

Where λ_i are the eigenvalues of $Q \in \mathbb{C}^{m \times m}$, φ_i are the coefficients of the expansion of x in the basis of eigenvectors v_i. Since we have that $|\lambda_i| = 1$, we have:

The fact that $|\lambda_i| = 1 \Rightarrow |\lambda_i^n| = 1$ is sufficiently enough for one to be convinced that power iteration does not converge.

Let $\lambda_k = e^{i\psi_k}$, where $\psi_k \in \mathbb{R}$. Then expanding eq. (14):

$$
Q^n x = \frac{1}{e^{i\psi_j \cdot n}} \cdot \sum_{\tau=1}^{m} e^{i\psi_\tau n} \varphi_\tau v_\tau \tag{15}
$$

When $n \to \infty$ if $\lambda_j = 1$ then we have:

$$
Q^n x = \varphi_j v_j + \sum_{\tau \neq j} e^{i\psi_\tau n} \varphi_\tau v_\tau \tag{16}
$$

Since no eigenvalue dominates other eigenvalues in the orthogonal case, usually power iteration fails.

4.2. Orthogonal Matrices and Inverse Iteration

If we apply inverse iteration to an orthogonal matrix with a shift μ, we have:

$$
\begin{aligned}
\det\left(Q - \mu I - \hat{\lambda} I\right) = 0 &\Leftrightarrow \det\left(Q - \left(\mu + \hat{\lambda}\right) I\right) = 0 \\
&\Leftrightarrow \hat{\lambda}_j = \frac{1}{\lambda_j - \mu}
\end{aligned}
\tag{17}
$$

We know that the eigenvalues of Q are on the unit circle, so if μ is close to an eigenvalue λ_j, $\hat{\lambda}_j$ will be huge (dominant), which makes power iteration converge to the eigenvector associated to $\hat{\lambda}_{j_j}$, which is the eigenvector associated to λ_j. The fact that the eigenvalues are on the unit circle also contributes to the convergence of the method.

So we concude that inverse iteration works well for orthogonal matrices, *if μ is close to an eigenvalue of Q*.

4.3. The 2×2 Case (b)

We will calculate the eigenvalues of:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{18}$$

With $a, b, c, d \in \mathbb{R}$. The characteristic polynomial gives us:

$$\begin{gathered} \det(A - \lambda I) = 0 \Leftrightarrow \det\begin{pmatrix} a - \lambda & b \\ c & d - \lambda \end{pmatrix} = 0 \\ \Leftrightarrow (a - \lambda)(d - \lambda) - bc = 0 \Leftrightarrow \lambda^2 + \lambda(-a - d) + (ad - bc) = 0 \\ \Leftrightarrow \lambda = (a + d) \pm \frac{\sqrt{(a + d)^2 - 4(ad - bc)}}{2} \end{gathered} \tag{19}$$

So the eigenvalues are:

$$\begin{aligned} \lambda_1 &= \frac{a + d + \sqrt{(a + d)^2 - 4(ad - bc)}}{2} \\ \lambda_2 &= \frac{a + d - \sqrt{(a + d)^2 - 4(ad - bc)}}{2} \end{aligned} \tag{20}$$

4.4. Random Orthogonal Matrices (c)

This code generates orthogonal matrices of order 4×4 generated by the QR factorization of random matrices, and reduces the to Hessenberg form. The eigenvalues of the bottom-right 2×2 block are analytically calculated using Section 4.3.

```python
def generate_orthogonal_matrix_qr(n=4, seed=None):

    """
    Generates a random orthogonal matrix using QR decomposition.
    Args:
        n (int): Size of the matrix (n x n).
        seed (int | None): Random seed for reproducibility.
    Returns:
        np.ndarray: An n x n orthogonal matrix.
    Raises:
        None
    """

    if seed is not None:
        np.random.seed(seed)
    A = np.random.randn(n, n)
    Q, _ = np.linalg.qr(A)
    return Q

def analytical_eigenvalues_2x2(a, b, c, d):

    """
    Calculates the eigenvalues of a 2x2 matrix analytically.
    Args:
        a (float): Element at position (0,0).
```

```
        b (float): Element at position (0,1).
        c (float): Element at position (1,0).
        d (float): Element at position (1,1).
    Returns:
        Tuple[float, float]: The two eigenvalues of the matrix.
    Raises:
    """

    trace = a + d
    det = a * d - b * c
    discriminant = trace**2 - 4 * det

    #complex if discriminant negative
    discriminant_root = np.sqrt(discriminant) if discriminant >= 0 else
    np.sqrt(complex(discriminant))

    lambda1 = (trace + discriminant_root) / 2
    lambda2 = (trace - discriminant_root) / 2

    return lambda1, lambda2

def analyze_orthogonal_and_hessenberg(n=4, n_matrices=30):

    """
    Analyzes orthogonal matrices and their Hessenberg forms.
    Args:
        n (int): Size of the matrices (n x n).
        n_matrices (int): Number of orthogonal matrices to generate and analyze.
    Returns:
        None
    Raises:
        None
    """

    for i in range(n_matrices):
        print(f"\n--- Orthogonal Matrix Q number {i+1} ---")
        Q = generate_orthogonal_matrix_qr(n=n)
        print("Matrix Q:")
        print(np.array_str(Q, precision=4, suppress_small=True))

        householder_list, H, Q_accum = to_hessenberg(Q)

        print("\nHessenberg Form H (of Q):")
        print(np.array_str(H, precision=4, suppress_small=True))

        block = Q[2:4, 2:4]
```

```python
        a, b, c, d = block[0,0], block[0,1], block[1,0], block[1,1]
        analytical_eigenvalues = analytical_eigenvalues_2x2(a, b, c, d)

        print("\nBlock Q[3:4,3:4] (indices 2 and 3, 2x2):")

        print(np.array_str(block, precision=4, suppress_small=True))

        print("\nEigenvalues of the 2x2 block (analytically calculated):")
        for idx, val in enumerate(analytical_eigenvalues):
            print(f"  λ_{idx+1} = {val} (size = {abs(val):.4f})")

        print("-" * 40)

analyze_orthogonal_and_hessenberg()
```

We ran this code for 30 matrices, the output was:

```
--- Orthogonal Matrix Q number 1 ---
Matrix Q:
[[-0.5629  0.6801  0.4635 -0.0764]
 [-0.6703 -0.6884  0.2231  0.1645]
 [-0.1497  0.2337 -0.3793  0.8827]
 [ 0.4598 -0.0949  0.7691  0.4336]]

Hessenberg Form H (of Q):
[[-0.5629 -0.6779 -0.4534 -0.1342]
 [ 0.8265 -0.4617 -0.3088 -0.0914]
 [ 0.     -0.5721  0.7865  0.2329]
 [-0.      0.      0.2839 -0.9588]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.3793  0.8827]
 [ 0.7691  0.4336]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.9458819605346136 (size = 0.9459)
  λ_2 = -0.8915812804514585 (size = 0.8916)
----------------------------------------

--- Orthogonal Matrix Q number 2 ---
Matrix Q:
[[-0.4016  0.3605  0.8354  0.1045]
 [-0.2846 -0.3518  0.1255 -0.8829]
 [-0.3325 -0.8166  0.136   0.4519]
 [ 0.8045 -0.282   0.5176 -0.0734]]

Hessenberg Form H (of Q):
```

```
[[-0.4016 -0.3235 -0.1952  0.8343]
 [ 0.9158 -0.1418 -0.0856  0.3658]
 [ 0.     -0.9355  0.0805 -0.3439]
 [ 0.      0.     -0.9737 -0.2278]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[ 0.136   0.4519]
 [ 0.5176 -0.0734]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.5261528266779573 (size = 0.5262)
  λ_2 = -0.4635182526918817 (size = 0.4635)
----------------------------------------

--- Orthogonal Matrix Q number 3 ---
Matrix Q:
[[-0.0452 -0.9852 -0.1571  0.0523]
 [ 0.4259  0.0412 -0.0809  0.9002]
 [ 0.1843  0.1346 -0.9569 -0.1793]
 [-0.8846  0.0982 -0.2303  0.3934]]

Hessenberg Form H (of Q):
[[-0.0452  0.4954 -0.8604 -0.1112]
 [-0.999  -0.0224  0.0389  0.005 ]
 [ 0.      0.8684  0.4918  0.0636]
 [-0.      0.      0.1282 -0.9917]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.9569 -0.1793]
 [-0.2303  0.3934]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.42331102317930497 (size = 0.4233)
  λ_2 = -0.9868680130188092 (size = 0.9869)
----------------------------------------

--- Orthogonal Matrix Q number 4 ---
Matrix Q:
[[-0.2568  0.8228 -0.2287  0.4525]
 [-0.2848 -0.018   0.9011  0.3265]
 [-0.6054 -0.5408 -0.3667  0.4544]
 [ 0.6975 -0.1738 -0.0346  0.6943]]

Hessenberg Form H (of Q):
[[-0.2568  0.2274  0.1895  0.92  ]
 [ 0.9665  0.0604  0.0503  0.2444]
```

```
 [-0.     -0.9719  0.0475  0.2305]
 [ 0.      0.     -0.9794  0.2017]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.3667  0.4544]
 [-0.0346  0.6943]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.67929095950588 (size = 0.6793)
  λ_2 = -0.3516952691053273 (size = 0.3517)
----------------------------------------

--- Orthogonal Matrix Q number 5 ---
Matrix Q:
[[-0.5025 -0.7649 -0.0345  0.4015]
 [ 0.3718  0.0654  0.6626  0.6469]
 [ 0.6819 -0.6372  0.0297 -0.358 ]
 [ 0.3798  0.0679 -0.7476  0.5406]]

Hessenberg Form H (of Q):
[[-0.5025  0.1798 -0.7903  0.3011]
 [-0.8646 -0.1045  0.4593 -0.175 ]
 [-0.     -0.9781 -0.1943  0.074 ]
 [-0.      0.      0.356   0.9345]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[ 0.0297 -0.358 ]
 [-0.7476  0.5406]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.8621172090206812 (size = 0.8621)
  λ_2 = -0.2918043055687003 (size = 0.2918)
----------------------------------------

--- Orthogonal Matrix Q number 6 ---
Matrix Q:
[[-0.9092  0.1745  0.2312 -0.2992]
 [ 0.1618 -0.3496 -0.2497 -0.8884]
 [-0.1454  0.4629 -0.8736  0.0369]
 [-0.3551 -0.7956 -0.3479  0.3462]]

Hessenberg Form H (of Q):
[[-0.9092 -0.2422 -0.3011 -0.1552]
 [-0.4164  0.5288  0.6573  0.3389]
 [ 0.      0.8134 -0.517  -0.2665]
 [-0.      0.      0.4582 -0.8888]]
```

```

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.8736  0.0369]
 [-0.3479  0.3462]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.3356278613353456 (size = 0.3356)
  λ_2 = -0.8630034423078552 (size = 0.8630)
----------------------------------------

--- Orthogonal Matrix Q number 7 ---
Matrix Q:
[[-0.4296  0.6844 -0.4085  0.4245]
 [ 0.2184 -0.4244 -0.028   0.8783]
 [ 0.17   -0.3028 -0.9122 -0.2177]
 [ 0.8596  0.5097 -0.0167  0.032 ]]

Hessenberg Form H (of Q):
[[-0.4296 -0.4928  0.7535  0.0703]
 [-0.903   0.2344 -0.3584 -0.0334]
 [-0.     -0.838  -0.5433 -0.0507]
 [ 0.      0.      0.0928 -0.9957]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.9122 -0.2177]
 [-0.0167  0.032 ]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.0358294023777706 (size = 0.0358)
  λ_2 = -0.9159948591320213 (size = 0.9160)
----------------------------------------

--- Orthogonal Matrix Q number 8 ---
Matrix Q:
[[-0.3402 -0.0284 -0.009   0.9399]
 [-0.7148 -0.6207  0.1662 -0.2759]
 [-0.5681  0.5973 -0.5323 -0.1927]
 [ 0.2249 -0.5071 -0.83    0.0581]]

Hessenberg Form H (of Q):
[[-0.3402  0.2518  0.8476  0.3201]
 [ 0.9403  0.0911  0.3067  0.1158]
 [ 0.      0.9635 -0.2505 -0.0946]
 [-0.     -0.      0.3533 -0.9355]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
```

```
[[-0.5323 -0.1927]
 [-0.83    0.0581]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.25997158900622225 (size = 0.2600)
  λ_2 = -0.7341829044352151 (size = 0.7342)
----------------------------------------

--- Orthogonal Matrix Q number 9 ---
Matrix Q:
[[-0.2692  0.5878 -0.5428  0.5361]
 [-0.8282  0.0254  0.5486  0.1116]
 [-0.292   0.3638 -0.2875 -0.8365]
 [-0.3954 -0.7222 -0.5673  0.0189]]

Hessenberg Form H (of Q):
[[-0.2692 -0.561  -0.1619  0.7659]
 [ 0.9631 -0.1568 -0.0453  0.2141]
 [-0.     -0.8128  0.1205 -0.5699]
 [ 0.     -0.     -0.9784 -0.2069]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.2875 -0.8365]
 [-0.5673  0.0189]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.5713848430035342 (size = 0.5714)
  λ_2 = -0.8399942790872519 (size = 0.8400)
----------------------------------------

--- Orthogonal Matrix Q number 10 ---
Matrix Q:
[[-0.4241  0.1349 -0.1773  0.8778]
 [-0.332  -0.8664 -0.3594 -0.0999]
 [ 0.8422 -0.2846 -0.2058  0.409 ]
 [-0.0237 -0.3876  0.8928  0.2284]]

Hessenberg Form H (of Q):
[[-0.4241 -0.2373  0.8677  0.1046]
 [ 0.9056 -0.1111  0.4063  0.049 ]
 [-0.      0.9651  0.2602  0.0313]
 [ 0.      0.      0.1196 -0.9928]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.2058  0.409 ]
 [ 0.8928  0.2284]]
```

```

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.6533894082311485 (size = 0.6534)
  λ_2 = -0.6307995911677337 (size = 0.6308)
----------------------------------------

--- Orthogonal Matrix Q number 11 ---
Matrix Q:
[[-0.1675 -0.9786 -0.0461 -0.1099]
 [ 0.7871 -0.1742  0.5819  0.1072]
 [ 0.3816 -0.1052 -0.6644  0.6339]
 [-0.4547 -0.0293  0.4667  0.758 ]]

Hessenberg Form H (of Q):
[[-0.1675  0.7485 -0.1241 -0.6295]
 [-0.9859 -0.1271  0.0211  0.1069]
 [ 0.     -0.6508 -0.1468 -0.7449]
 [ 0.     -0.     -0.9811  0.1934]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.6644  0.6339]
 [ 0.4667  0.758 ]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.9421402114923986 (size = 0.9421)
  λ_2 = -0.8485799008471894 (size = 0.8486)
----------------------------------------

--- Orthogonal Matrix Q number 12 ---
Matrix Q:
[[-0.3541  0.5585 -0.0616  0.7476]
 [-0.5108 -0.6958 -0.4435  0.2414]
 [ 0.1705  0.3507 -0.885  -0.2542]
 [ 0.7646 -0.2844 -0.1274  0.5641]]

Hessenberg Form H (of Q):
[[-0.3541  0.295   0.7613  0.4561]
 [ 0.9352  0.1117  0.2883  0.1727]
 [-0.      0.949  -0.2706 -0.1621]
 [-0.     -0.      0.5139 -0.8579]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.885  -0.2542]
 [-0.1274  0.5641]]

Eigenvalues of the 2x2 block (analytically calculated):
```

```
  λ_1 = 0.5861402323977607 (size = 0.5861)
  λ_2 = -0.9070552039589896 (size = 0.9071)
----------------------------------------

--- Orthogonal Matrix Q number 13 ---
Matrix Q:
[[-0.7015  0.4915  0.3042 -0.417 ]
 [ 0.4623 -0.1626  0.1522 -0.8583]
 [ 0.2479  0.6778 -0.6824 -0.1159]
 [-0.4825 -0.5221 -0.647  -0.2757]]

Hessenberg Form H (of Q):
[[-0.7015 -0.7069 -0.0829 -0.0373]
 [-0.7127  0.6958  0.0816  0.0367]
 [ 0.      0.1276 -0.9045 -0.4069]
 [ 0.     -0.      0.4103 -0.912 ]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.6824 -0.1159]
 [-0.647  -0.2757]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = -0.1379150700589457 (size = 0.1379)
  λ_2 = -0.820164584090632 (size = 0.8202)
----------------------------------------

--- Orthogonal Matrix Q number 14 ---
Matrix Q:
[[-0.2164 -0.5735  0.7758 -0.1498]
 [-0.3581 -0.6722 -0.6278 -0.1609]
 [-0.1535  0.3054  0.0015 -0.9398]
 [-0.8952  0.3551  0.0633  0.2617]]

Hessenberg Form H (of Q):
[[-0.2164  0.2256 -0.3541  0.8814]
 [ 0.9763  0.05   -0.0785  0.1954]
 [-0.     -0.9729 -0.0862  0.2144]
 [-0.      0.     -0.9279 -0.3728]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[ 0.0015 -0.9398]
 [ 0.0633  0.2617]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = (0.1316169167334142+0.20628124638940884j) (size = 0.2447)
  λ_2 = (0.1316169167334142-0.20628124638940884j) (size = 0.2447)
```

```
----------------------------------------

--- Orthogonal Matrix Q number 15 ---
Matrix Q:
[[-0.467   0.1387  0.7316  0.4769]
 [ 0.0375 -0.9757  0.0774  0.2017]
 [-0.7384 -0.0272 -0.627   0.2467]
 [ 0.485   0.1677 -0.2562  0.8191]]

Hessenberg Form H (of Q):
[[-0.467   0.3435 -0.8022 -0.1428]
 [-0.8843 -0.1814  0.4236  0.0754]
 [-0.      0.9215  0.3824  0.068 ]
 [-0.     -0.      0.1752 -0.9845]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.627   0.2467]
 [-0.2562  0.8191]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.7740235941481243 (size = 0.7740)
  λ_2 = -0.5818796203145907 (size = 0.5819)
----------------------------------------

--- Orthogonal Matrix Q number 16 ---
Matrix Q:
[[-0.3849  0.0435 -0.7406  0.549 ]
 [ 0.2525 -0.952  -0.07    0.158 ]
 [ 0.7807  0.2969  0.0191  0.5495]
 [-0.4225 -0.06    0.668   0.6096]]

Hessenberg Form H (of Q):
[[-0.3849  0.866   0.1952  0.2527]
 [-0.923  -0.3611 -0.0814 -0.1054]
 [ 0.      0.346  -0.5736 -0.7425]
 [ 0.      0.     -0.7914  0.6114]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[0.0191 0.5495]
 [0.668  0.6096]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.9883202000131572 (size = 0.9883)
  λ_2 = -0.3596282457787697 (size = 0.3596)
----------------------------------------

```

```
--- Orthogonal Matrix Q number 17 ---
Matrix Q:
[[-0.1066  0.7212  0.0749 -0.6804]
 [-0.6764 -0.5121  0.3485 -0.3985]
 [-0.5738  0.1159 -0.8012  0.1245]
 [-0.4493  0.4519  0.4807  0.6024]]

Hessenberg Form H (of Q):
[[-0.1066 -0.2265 -0.8608 -0.4431]
 [ 0.9943 -0.0243 -0.0923 -0.0475]
 [ 0.     -0.9737  0.2025  0.1043]
 [ 0.      0.      0.4577 -0.8891]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.8012  0.1245]
 [ 0.4807  0.6024]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.6437814804055579 (size = 0.6438)
  λ_2 = -0.842579679233141 (size = 0.8426)
----------------------------------------

--- Orthogonal Matrix Q number 18 ---
Matrix Q:
[[-0.8255  0.1636  0.5347  0.0763]
 [-0.5177 -0.2699 -0.6467 -0.4908]
 [ 0.0194  0.9433 -0.2236 -0.2446]
 [-0.2238  0.103  -0.4958  0.8327]]

Hessenberg Form H (of Q):
[[-0.8255 -0.1619 -0.4367 -0.3188]
 [ 0.5644 -0.2368 -0.6388 -0.4663]
 [ 0.      0.958  -0.2317 -0.1691]
 [-0.      0.     -0.5896  0.8077]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.2236 -0.2446]
 [-0.4958  0.8327]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.9372008193289816 (size = 0.9372)
  λ_2 = -0.32809878403400955 (size = 0.3281)
----------------------------------------

--- Orthogonal Matrix Q number 19 ---
Matrix Q:
```

```
[[-0.7885  0.0127  0.4922 -0.3686]
 [ 0.5982 -0.2155  0.6311 -0.4444]
 [ 0.0328  0.1282 -0.5621 -0.8164]
 [-0.1391 -0.968  -0.2085 -0.0141]]

Hessenberg Form H (of Q):
[[-0.7885 -0.122  -0.4769  0.3687]
 [-0.615   0.1564  0.6114 -0.4727]
 [ 0.      0.9801 -0.1569  0.1213]
 [ 0.      0.     -0.6116 -0.7911]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.5621 -0.8164]
 [-0.2085 -0.0141]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.2071704757269035 (size = 0.2072)
  λ_2 = -0.7833862909149965 (size = 0.7834)
----------------------------------------

--- Orthogonal Matrix Q number 20 ---
Matrix Q:
[[-0.3031  0.0429  0.9481  0.0861]
 [ 0.1571 -0.9342  0.1195 -0.2972]
 [-0.8533 -0.0239 -0.2292 -0.4678]
 [-0.3942 -0.3534 -0.1853  0.8279]]

Hessenberg Form H (of Q):
[[-0.3031  0.8775 -0.2647  0.261 ]
 [-0.953  -0.2791  0.0842 -0.083 ]
 [ 0.      0.3901  0.6557 -0.6464]
 [ 0.     -0.     -0.702  -0.7121]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.2292 -0.4678]
 [-0.1853  0.8279]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.9043475415351031 (size = 0.9043)
  λ_2 = -0.30565758703551016 (size = 0.3057)
----------------------------------------

--- Orthogonal Matrix Q number 21 ---
Matrix Q:
[[-0.8149  0.3649 -0.4406 -0.0928]
 [ 0.0977 -0.6065 -0.7405  0.2726]
```

```
 [ 0.0387  0.372   0.0415  0.9265]
 [-0.57   -0.6005  0.5058  0.2423]]

Hessenberg Form H (of Q):
[[-0.8149 -0.1234  0.2317  0.5167]
 [-0.5796  0.1735 -0.3257 -0.7266]
 [ 0.     -0.9771 -0.0871 -0.1943]
 [-0.      0.      0.9125 -0.4091]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[0.0415 0.9265]
 [0.5058 0.2423]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.8337542266557703 (size = 0.8338)
  λ_2 = -0.5499977130761522 (size = 0.5500)
----------------------------------------

--- Orthogonal Matrix Q number 22 ---
Matrix Q:
[[-0.3663  0.4189 -0.6842 -0.4714]
 [ 0.2919 -0.672  -0.6656  0.142 ]
 [ 0.5024  0.6076 -0.2758  0.5498]
 [-0.7268 -0.061  -0.1131  0.6748]]

Hessenberg Form H (of Q):
[[-0.3663 -0.1301  0.2621  0.8833]
 [-0.9305  0.0512 -0.1032 -0.3477]
 [-0.     -0.9902 -0.0398 -0.134 ]
 [ 0.     -0.      0.9587 -0.2845]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.2758  0.5498]
 [-0.1131  0.6748]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.6040601009013424 (size = 0.6041)
  λ_2 = -0.20508048336653284 (size = 0.2051)
----------------------------------------

--- Orthogonal Matrix Q number 23 ---
Matrix Q:
[[-0.4467 -0.7825 -0.2287  0.3687]
 [ 0.4222  0.182  -0.0157  0.8879]
 [ 0.6922 -0.3543 -0.5694 -0.2666]
 [-0.3783  0.4786 -0.7895  0.0678]]
```

```

Hessenberg Form H (of Q):
[[-0.4467  0.702  -0.0795 -0.5489]
 [-0.8947 -0.3505  0.0397  0.274 ]
 [ 0.     -0.62   -0.1125 -0.7765]
 [ 0.      0.     -0.9897  0.1433]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.5694 -0.2666]
 [-0.7895  0.0678]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.3077269347022792 (size = 0.3077)
  λ_2 = -0.8093108181365211 (size = 0.8093)
----------------------------------------

--- Orthogonal Matrix Q number 24 ---
Matrix Q:
[[-0.0413 -0.9591  0.2707  0.0721]
 [ 0.08   -0.274  -0.9584  0.0005]
 [-0.6777 -0.0389 -0.0459 -0.7328]
 [-0.7298  0.0604 -0.0778  0.6766]]

Hessenberg Form H (of Q):
[[-0.0413  0.3131  0.5108 -0.7996]
 [-0.9991 -0.0129 -0.0211  0.0331]
 [-0.      0.9496 -0.1687  0.2641]
 [ 0.      0.      0.8427  0.5384]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.0459 -0.7328]
 [-0.0778  0.6766]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.7483535886136121 (size = 0.7484)
  λ_2 = -0.1176716985715956 (size = 0.1177)
----------------------------------------

--- Orthogonal Matrix Q number 25 ---
Matrix Q:
[[-0.7483  0.5153 -0.0763  0.4107]
 [-0.4762 -0.7375 -0.4779 -0.0313]
 [-0.1713  0.3719 -0.3481 -0.8433]
 [ 0.4288  0.2287 -0.8029  0.3451]]

Hessenberg Form H (of Q):
```

```
[[-0.7483 -0.0847  0.4863  0.4431]
 [ 0.6633 -0.0956  0.5486  0.4998]
 [ 0.      0.9918  0.0944  0.086 ]
 [ 0.      0.      0.6735 -0.7392]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.3481 -0.8433]
 [-0.8029  0.3451]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.8914410770554546 (size = 0.8914)
  λ_2 = -0.8943450015206609 (size = 0.8943)
----------------------------------------

--- Orthogonal Matrix Q number 26 ---
Matrix Q:
[[-0.9036  0.127  -0.0161  0.4088]
 [-0.2097 -0.5366 -0.7493 -0.3265]
 [ 0.3449  0.3256 -0.6074  0.6373]
 [ 0.1433 -0.7681  0.2634  0.5658]]

Hessenberg Form H (of Q):
[[-0.9036  0.0616  0.2624  0.3329]
 [ 0.4284  0.13    0.5536  0.7022]
 [ 0.      0.9896 -0.0891 -0.113 ]
 [-0.     -0.      0.7853 -0.6191]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.6074  0.6373]
 [ 0.2634  0.5658]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.6947481590186378 (size = 0.6947)
  λ_2 = -0.7362843808778513 (size = 0.7363)
----------------------------------------

--- Orthogonal Matrix Q number 27 ---
Matrix Q:
[[-0.3783  0.2175  0.7347 -0.5194]
 [-0.486  -0.5305 -0.4607 -0.5198]
 [-0.7609  0.4199 -0.1943  0.455 ]
 [-0.2046 -0.7036  0.4585  0.5029]]

Hessenberg Form H (of Q):
[[-0.3783 -0.6032 -0.0497 -0.7004]
 [ 0.9257 -0.2465 -0.0203 -0.2862]
```

```
 [ 0.      0.7585 -0.0461 -0.65  ]
 [ 0.     -0.     -0.9975  0.0708]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.1943  0.455 ]
 [ 0.4585  0.5029]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.7288996874702811 (size = 0.7289)
  λ_2 = -0.42026844598624513 (size = 0.4203)
----------------------------------------

--- Orthogonal Matrix Q number 28 ---
Matrix Q:
[[-0.6177 -0.2663 -0.3252  0.6646]
 [ 0.7345 -0.3401  0.0741  0.5826]
 [ 0.0015 -0.8476 -0.2571 -0.4641]
 [ 0.2811  0.308  -0.907  -0.0592]]

Hessenberg Form H (of Q):
[[-0.6177  0.0118  0.5166  0.5928]
 [-0.7864 -0.0093 -0.4058 -0.4656]
 [ 0.     -0.9999  0.0099  0.0113]
 [ 0.     -0.      0.7539 -0.657 ]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.2571 -0.4641]
 [-0.907  -0.0592]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.4981407629675436 (size = 0.4981)
  λ_2 = -0.8144343078024388 (size = 0.8144)
----------------------------------------

--- Orthogonal Matrix Q number 29 ---
Matrix Q:
[[-0.889  -0.0288  0.3962 -0.2278]
 [ 0.1961 -0.4549 -0.0899 -0.864 ]
 [-0.4017 -0.36   -0.8217  0.1838]
 [ 0.0995 -0.814   0.3997  0.4095]]

Hessenberg Form H (of Q):
[[-0.889   0.4094  0.14   -0.1501]
 [-0.458  -0.7947 -0.2718  0.2914]
 [ 0.     -0.4483  0.6097 -0.6537]
 [ 0.     -0.     -0.7313 -0.6821]]
```

```

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.8217  0.1838]
 [ 0.3997  0.4095]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.46657705244665193 (size = 0.4666)
  λ_2 = -0.8787442629640516 (size = 0.8787)
----------------------------------------

--- Orthogonal Matrix Q number 30 ---
Matrix Q:
[[-0.2493  0.2801 -0.5896 -0.7154]
 [-0.5128 -0.2897  0.6545 -0.4742]
 [-0.2303 -0.8469 -0.4616  0.1292]
 [-0.7886  0.3471 -0.1044  0.4967]]

Hessenberg Form H (of Q):
[[-0.2493  0.5744 -0.1969 -0.7544]
 [ 0.9684  0.1479 -0.0507 -0.1942]
 [ 0.     -0.8051 -0.1498 -0.5739]
 [ 0.     -0.      0.9676 -0.2526]]

Block Q[3:4,3:4] (indices 2 and 3, 2x2):
[[-0.4616  0.1292]
 [-0.1044  0.4967]]

Eigenvalues of the 2x2 block (analytically calculated):
  λ_1 = 0.4824356418279282 (size = 0.4824)
  λ_2 = -0.4473584670796439 (size = 0.4474)
----------------------------------------
```

So we observe that in the 2×2 blocks analyzed:

1. Orthogonality is not always preserved
2. The eigenvalues are usually real, with alternating sign and size around 1.

4.5. Shift With an Eigenvalue (d)

Now we use an eigenvalue of the 2×2 block as a shift:

```python
def qr_iteration_with_fixed_shift(H, mu, max_iter=100):

    """
    Applies QR iteration with fixed shift on a matrix H.

    Args:
        H (np.ndarray): initial matrix in Hessenberg form.
```

```
        mu (complex): fixed shift to be used.
        max_iter (int): maximum number of iterations.

    Returns:
        Hk (np.ndarray): matrix after iterations.
        converged (bool): whether it converged to almost upper triangular form.
    """

    Hk = H.copy()
    n = Hk.shape[0]

    for _ in range(max_iter):
        H_shifted = Hk - mu * np.eye(n)
        Q, R = np.linalg.qr(H_shifted)
        Hk = R @ Q + mu * np.eye(n)

    subdiag = np.abs(np.diag(Hk, k=-1))
    tol = 1e-5
    converged = np.all(subdiag < tol)
    return Hk, converged

def run_qr_iteration_with_shifts(n=4, n_matrices=30):

    """
    Runs QR iteration with fixed shift on randomly generated orthogonal matrices.

    Args:
        n (int): Size of the matrices (n x n).
        n_matrices (int): Number of orthogonal matrices to generate and analyze.

    Returns:
        None

    Raises:
        None
    """

    for i in range(n_matrices):
        print(f"\n--- Orthogonal Matrix Q number {i+1} ---")
        Q = generate_orthogonal_matrix_qr(n=n)
        _, H, _ = to_hessenberg(Q)

        final_block = H[-2:, -2:]
        a, b, c, d = final_block[0,0], final_block[0,1], final_block[1,0], final_block[1,1]
```

```
        shift_candidates = analytical_eigenvalues_2x2(a, b, c, d)

        mu = shift_candidates[0]  #use the first eigenvalue as fixed shift
        print(f"Fixed shift used (eigenvalue of the final block): {mu} (modulus {abs(mu):.4f})")

        Hk, converged = qr_iteration_with_fixed_shift(H, mu, max_iter=20)

        print("Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):")
        print(np.array_str(np.diag(Hk, k=-1), precision=3, suppress_small=True))

        print(f"Converged to almost upper triangular form? {'Yes' if converged else 'No'}")
        print("-" * 50)


run_qr_iteration_with_shifts()
```

We once more ran this on 30 random matrices, with 100 iterations and using a generous `tolarance = 1e-5` for convergence. The output is:

```
--- Orthogonal Matrix Q number 1 ---
Fixed shift used (eigenvalue of the final block): 0.45675480266096846 (modulus 0.4568)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.006  1.     0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 2 ---
Fixed shift used (eigenvalue of the final block): 0.47119769752489193 (modulus 0.4712)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.443  0.501 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 3 ---
Fixed shift used (eigenvalue of the final block): 0.6543379277197392 (modulus 0.6543)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.004 -1.     0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 4 ---
```

```
Fixed shift used (eigenvalue of the final block): 0.47308128373683983 (modulus 0.4731)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[ 0.47  -0.739  0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 5 ---
Fixed shift used (eigenvalue of the final block): 0.999835862374747 (modulus 0.9998)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[ 0.079 -0.95  -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 6 ---
Fixed shift used (eigenvalue of the final block): 0.8163222681328892 (modulus 0.8163)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.597 -0.815 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 7 ---
Fixed shift used (eigenvalue of the final block): 0.8964184377049635 (modulus 0.8964)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.37  -0.249 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 8 ---
Fixed shift used (eigenvalue of the final block): 0.9689586976565394 (modulus 0.9690)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.204 -0.864  0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 9 ---
Fixed shift used (eigenvalue of the final block): 0.8100766994771901 (modulus 0.8101)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[0.332 0.362 0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

```

```
--- Orthogonal Matrix Q number 10 ---
Fixed shift used (eigenvalue of the final block): (-0.37373545734503094+0.5961171763741568j) (modulus 0.7036)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.817-0.j  0.   -0.j -0.   -0.j]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 11 ---
Fixed shift used (eigenvalue of the final block): 0.9508997822578601 (modulus 0.9509)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.    -0.985 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 12 ---
Fixed shift used (eigenvalue of the final block): 0.49537543518658966 (modulus 0.4954)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[ 0.096  0.822 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 13 ---
Fixed shift used (eigenvalue of the final block): -0.5169109822890083 (modulus 0.5169)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.    0.57  0.  ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 14 ---
Fixed shift used (eigenvalue of the final block): 0.8706109171378603 (modulus 0.8706)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.239 -0.766  0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 15 ---
Fixed shift used (eigenvalue of the final block): 0.7877591057872764 (modulus 0.7878)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.259 -0.103 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------
```

```

--- Orthogonal Matrix Q number 16 ---
Fixed shift used (eigenvalue of the final block): 0.7525805488977373 (modulus 0.7526)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.537 -0.738 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 17 ---
Fixed shift used (eigenvalue of the final block): 0.9376989809342731 (modulus 0.9377)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[ 0.003 -0.996  0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 18 ---
Fixed shift used (eigenvalue of the final block): (-0.3332517830071887+0.31982101244995226j) (modulus 0.4619)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.081+0.j  0.   -0.j  0.003-0.j]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 19 ---
Fixed shift used (eigenvalue of the final block): -0.09828756789250936 (modulus 0.0983)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[0.171 0.5   0.14 ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 20 ---
Fixed shift used (eigenvalue of the final block): 0.9649949076474909 (modulus 0.9650)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[ 0.014 -1.    -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 21 ---
Fixed shift used (eigenvalue of the final block): 0.9511618027193621 (modulus 0.9512)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.435  0.573  0.   ]
Converged to almost upper triangular form? No
```

```
--------------------------------------------------

--- Orthogonal Matrix Q number 22 ---
Fixed shift used (eigenvalue of the final block): 0.8791066383784223 (modulus 0.8791)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.001 -0.998 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 23 ---
Fixed shift used (eigenvalue of the final block): 0.9726088054062819 (modulus 0.9726)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[ 0.008  0.994 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 24 ---
Fixed shift used (eigenvalue of the final block): (-0.48219741339866473+0.8310922170834656j) (modulus 0.9608)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.442+0.j -0.   +0.j  0.   +0.j]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 25 ---
Fixed shift used (eigenvalue of the final block): (-0.9055728105908702+0.3001257739418847j) (modulus 0.9540)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.+0.j -0.+0.j -0.-0.j]
Converged to almost upper triangular form? Yes
--------------------------------------------------

--- Orthogonal Matrix Q number 26 ---
Fixed shift used (eigenvalue of the final block): 0.9549576704039157 (modulus 0.9550)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[ 0.001  0.995 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 27 ---
Fixed shift used (eigenvalue of the final block): -0.3999018693674167 (modulus 0.3999)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[0.    0.693 0.001]
```

```
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 28 ---
Fixed shift used (eigenvalue of the final block): 0.4453295247750607 (modulus 0.4453)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[ 0.287  0.893 -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 29 ---
Fixed shift used (eigenvalue of the final block): 0.9851064687720523 (modulus 0.9851)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[-0.002  1.    -0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------

--- Orthogonal Matrix Q number 30 ---
Fixed shift used (eigenvalue of the final block): 0.9234616212297447 (modulus 0.9235)
Matrix after 20 QR iterations with fixed shift (values below the subdiagonal):
[0.397 0.598 0.   ]
Converged to almost upper triangular form? No
--------------------------------------------------
```

We therefore conclude that:

1. There is usually no convergence to an upper triangular form, even after 100 iterations (we have seen only one case where it converged, with a generous tolerance of 10^{-5}).
2. Some values below the subdiagonal are still big (of order 0,01 and so).
3. The chosen shift was not enough for convergence

Bibliography

[1] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. 1997.