Assignment 3 - Numerical Linear Algebra

Arthur Rabello Oliveira[1]

26/05/2025

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

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

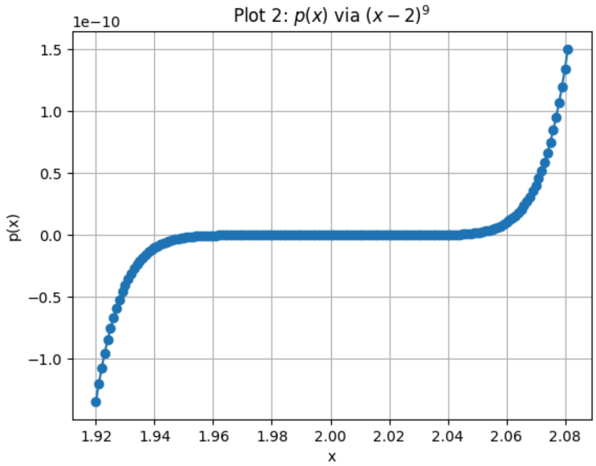

Figure 1: $p(x)$ via the coefficients in eq. (1)

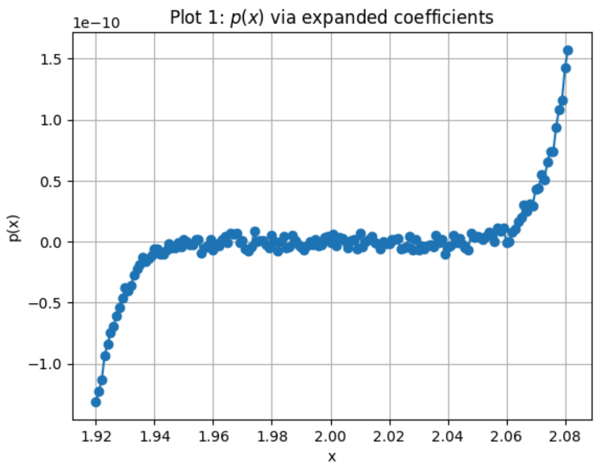

Figure 2: $p(x)$ via $(x-2)^9$

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 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
import numpy as np
import time
from typing import List, Tuple


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.
    """

    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 = (
```

```python
        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.
    """

    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)

2.3. Complexity (c)

2.4. The Symmetric Case (d)

3. Orthogonal Matrices (Problem 2)

3.1. Eigenvalues and Iterative Methods (a)

3.2. The 2×2 Case (b)

3.3. Random Orthogonal Matrices (c)

3.4. Shift With an Eigenvalue (d)

4. Conclusion

Bibliography