Numerical Linear Algebra Winter-Semester 2024

Programming-Exam Report

Heyuan Chi (4742393)

January 27, 2025

This report describes the transformation of a Hermitian matrix into a real tridiagonal form using Householder reflections, followed by a shifted QR iteration with deflation to compute all eigenvalues and eigenvectors, or compute all eigenvectors separately by inverse iteration. We present the mathematical concepts, explain the overall structure of our implementation, and provide numerical experiments that confirm both the correctness and performance of the method. All source code and additional information can be found at `https://github.com/HeyuanChi/NLA_final.git`.

Contents

1. Introduction

Hermitian matrices constitute a key class of matrices in numerical linear algebra because they have real eigenvalues and can be diagonalized by a unitary matrix. A common strategy for solving the Hermitian eigenvalue problem $Av = \lambda v$ involves the following steps:

1. **Tridiagonalization:** Reduce A to a real symmetric tridiagonal matrix T using Householder transformations so that $A = Q\,T\,Q^*$.
2. **QR Iteration:** Apply a shifted QR iteration on T (which converges more efficiently than applying QR directly on A) to compute its eigenvalues (and eigenvectors).
3. **Inverse Iteration:** After calculating all eigenvalues, calculate all eigenvectors of T through inverse iteration, and then calculate all eigenvectors of A by multiplying Q.

We implemented these methods and tested them on both random and repeated-eigenvalue scenarios to illustrate correctness and efficiency.

2. Program Structure Overview

Our project is divided into several header files, each serving a particular purpose:

- **householderTridiag.hpp**: Implements the `householderTridiag` function, which converts a complex Hermitian matrix into a real tridiagonal matrix.
- **TMatrix_base.hpp**, **TMatrix_ops.hpp**, **TMatrix_qr.hpp**: Contain the `TMatrix` class, which stores and manipulates the real tridiagonal matrix. This class also includes the `qrEigen` method for performing the shifted QR iteration.
- **eigenvectors.hpp**: Contain `thomas` for solving the tridiagonal system using the Thomas algorithm, `computeEigenVector` for calculating the eigenvector using inverse iteration and `computeAllEigenVect` for calculating all eigenvectors.
- **randHermitian.hpp**: Provides `randHermitian` for generating a random Hermitian matrix with specified eigenvalues.
- **testOne.hpp** and **testAll.hpp**: Contain routines for testing the process, including matrix generation, tridiagonalization, QR iteration, and detailed result reporting.

3. Householder Tridiagonalization

3.1. Mathematical Background

Given a Hermitian matrix $A \in \mathbb{C}^{n\times n}$, our objective is to eliminate off-diagonal entries below the first subdiagonal, yielding a real tridiagonal matrix T. This is achieved by applying a sequence of Householder transformations.

3.1.1. Householder Reflection Matrix

A Householder reflection can be written as

$$H = I - 2\frac{w\,w^*}{w^*\,w},$$

where w is the Householder vector. Such transformations are designed to zero out specific elements below the main diagonal in a column-by-column procedure.

3.1.2. Householder Vector

At the k-th step ($k = 1, \ldots, n-1$), we aim to set to zero all entries beneath the main diagonal in column k, beginning with row $k+1$. Let

$$x = \begin{bmatrix} A_{k+1,k} \\ A_{k+2,k} \\ \vdots \\ A_{n,k} \end{bmatrix} \in \mathbb{C}^{n-k}.$$

The Householder vector w can be formulated as

$$w = x - \text{phase}(x_0)\|x\|_2\, e_1,$$

where $\text{phase}(x_0)$ is determined by

$$\text{phase}(x_0) = \begin{cases} \dfrac{x_0}{|x_0|}, & |x_0| > 0, \\ 1, & \text{otherwise.} \end{cases}$$

The idea is to reflect x onto the scalar multiple of the first basis vector.

3.2. Implementation

Algorithm 1 Householder Tridiagonalization

Require: $A \in \mathbb{C}^{n\times n}$ (Hermitian), $Q \leftarrow I_{n\times n}$, tol

for $k = 1$ to $n-1$ **do**
  if $k < n-1$ **then**
    Extract $x = [A_{k+1,k}, \ldots, A_{n,k}]^T$
    Compute $\|x\|_2$
    if $\|x\|_2 <$ tol **then**
      continue
    else
      $c \leftarrow \text{phase}(x_1)$; $\alpha \leftarrow -c\|x\|_2$
      $u \leftarrow x$; $u_1 \leftarrow u_1 + \alpha$
      $w \leftarrow \dfrac{u}{\|u\|_2}$ ▷ Normalization
      $A \leftarrow A - 2\,w\,(w^*\,A)$
      $A \leftarrow A - 2\,(A\,w)\,w^*$ ▷ Update $A = H^*AH$
      $Q \leftarrow Q - 2\,(Q\,w)\,w^*$ ▷ Update Q, so $Q = QH$
    end if
  end if
  $R.\text{row}(k+1) \leftarrow R.\text{row}(k+1) \cdot \overline{c}$
  $R.\text{col}(k+1) \leftarrow R.\text{col}(k+1) \cdot c$
  $Q.\text{col}(k+1) \leftarrow Q.\text{col}(k+1) \cdot c$ ▷ Phase adjustment
end for

3.3. Implementation Improvements

To enhance efficiency and numerical robustness, the following measures are used:

1. **Immediate Extraction into T:** As soon as column k is zeroed out, we store the relevant diagonal and subdiagonal elements in `T` instead of waiting until the end.
2. **Reduced Updates:** Once $A_{i,k}$ (and $A_{k,i}$) for $i > k+1$ is set to zero, we no longer touch these elements in subsequent steps. Hence, we can restrict updates to the active sub-block $(k+1) : n$ and directly set $\texttt{T.subdiag}(k) = \|x\|_2$.
3. **Sub-block Reflections:** We only apply reflectors to the sub-block of A with row and column indices $\geq k+1$, thus avoiding unnecessary operations on elements that remain zero.
4. **Rank-2 Updates Instead of Forming H:** Instead of constructing the matrix $H = I - 2\,w\,w^*$ explicitly (which would require large-scale multiplications), we carry out transformations through rank-2 updates:
$$A \;\leftarrow\; A - 2\,w\,(w^*A), \qquad A \;\leftarrow\; A - 2\,(A\,w)\,w^*,$$
while similarly adjusting Q. This approach requires only a few vector–matrix multiplications rather than full matrix–matrix products.

4. QR Iteration

4.1. Mathematical Background

After reducing the Hermitian matrix A to a real symmetric tridiagonal matrix T, we employ the QR iteration to compute its eigenvalues and eigenvectors. For real symmetric tridiagonal matrices, QR iteration with an implicit shift (Wilkinson shift) and deflation is especially effective:

1. **Implicit Shift (Wilkinson Shift):** To accelerate convergence, we introduce a shift μ that approximates an eigenvalue of T by examining its bottom-right 2×2 block. A well-known choice is the Wilkinson shift:
$$\sigma = a_n + d - \operatorname{sign}(d)\,\sqrt{d^2 + b_{n-1}^2}, \quad d = \frac{a_{n-1} - a_n}{2}.$$
This shift effectively speeds up convergence near eigenvalues.
2. **Deflation:** Whenever the subdiagonal b_i is sufficiently small relative to the neighboring diagonal elements, i.e.,
$$|b_i| \leq \text{tol}\,(|a_i| + |a_{i+1}|),$$
we set $b_i = 0$. This splits T into two smaller blocks that can be handled independently, significantly reducing computational effort and declaring one or more eigenvalues converged.

We employ Givens rotations implicitly to maintain the tridiagonal structure during QR steps, with each rotation accumulated in a global matrix Q to keep track of the eigenvectors.

4.2. Implementation

Algorithm 2 Shifted QR Iteration on Real Symmetric Tridiagonal T

Require: Real symmetric tridiagonal matrix T, Q, tol, maxIter

```
iterCount ← 0
while True do
    (start, end) ← getSubBlock(T, tol)
    if end − start = 1 then
        solve2x2Block(T, Q, start, tol)          ▷ Handle 2×2 block directly
    else if end − start > 1 then
        qrStep(T, Q, start, end, tol)
    else
        break                                    ▷ No sub-block larger than 1; iteration ends
    end if
    iterCount ← iterCount + 1
    if iterCount ≥ maxIter then
        break
    end if
end while
```

4.2.1. getSubBlock

The function getSubBlock examines subdiagonal elements from the bottom up to identify if a portion has already converged (where the subdiagonal is negligible). It returns indices (*start*, *end*) defining the next block to process.

Algorithm 3 Compute the Start and End of Sub-Block (Part 1)

Require: T, tol

```
n ← T.size()
if n ≤ 1 then
    return (0, 0)
end if
end ← n − 1
i ← end − 1
while true do
    if |T.subdiag[i]| < (|T.diag[i]| + |T.diag[i + 1]|) · tol then
        T.subdiag[i] ← 0
        end ← end − 1
    else
        break
    end if
    if i = 0 then
        break
    end if
    i ← i − 1
end while
if end ≤ 0 then
    return (end, end)
end if
```

Algorithm 4 Compute the Start and End of Sub-Block (Part 2)

```
start ← end
i ← start − 1
while true do
    if |T.subdiag[i]| > (|T.diag[i]| + |T.diag[i + 1]|) × tol then
        start ← start − 1
    else
        T.subdiag[i] ← 0
        break
    end if
    if i = 0 then
        break
    end if
    i ← i − 1
end while
return (start, end)
```

4.2.2. solve2x2Block

When a sub-block is precisely 2×2, SOLVE2X2BLOCK solves it analytically, then updates Q accordingly.

Algorithm 5 Compute Eigenvalues of a 2×2 Block and Update Q

Require: Indices $i, i+1$ specify a 2×2 block in T, $Q \in \mathbb{C}^{n\times n}$, tol

$x \leftarrow T.\text{diag}[i]$
$y \leftarrow T.\text{subdiag}[i]$
$z \leftarrow T.\text{diag}[i+1]$
if $|y| < \text{tol}$ **then**
    $T.\text{subdiag}[i] \leftarrow 0$ ▷ Block is nearly diagonal
    return
end if
$\tau \leftarrow \dfrac{z - x}{2\,y}$
$t \leftarrow \text{sign}(\tau) \,/\, \left(|\tau| + \sqrt{1+\tau^2}\right)$
$c \leftarrow \dfrac{1}{\sqrt{1+t^2}}$
$s \leftarrow t\,c$
$x_{\text{new}} \leftarrow x\,c^2 - 2\,y\,s\,c + z\,s^2$
$z_{\text{new}} \leftarrow x\,s^2 + 2\,y\,s\,c + z\,c^2$
$T.\text{diag}[i] \leftarrow x_{\text{new}}$
$T.\text{diag}[i+1] \leftarrow z_{\text{new}}$
$T.\text{subdiag}[i] \leftarrow 0$
$Q_i \leftarrow Q.\text{col}(i)$
$Q_{i+1} \leftarrow Q.\text{col}(i+1)$
$Q.\text{col}(i) \leftarrow c\,Q_i - s\,Q_{i+1}$
$Q.\text{col}(i+1) \leftarrow s\,Q_i + c\,Q_{i+1}$

4.2.3. qrStep

QRSTEP executes a single QR step on the sub-block $[start : end]$, using Givens rotations and updating Q correspondingly.

Algorithm 6 QR Step with Implicit Shift

Require: Indices $[start : end]$ specify a sub-block in T, $Q \in \mathbb{C}^{n\times n}$, tol

$shift \leftarrow$ WILKINSONSHIFT(end)
$g \leftarrow T.\text{diag}[start] - shift$
$s \leftarrow -1$
$c \leftarrow 1$
$f \leftarrow 0$
$h \leftarrow 0$
$e \leftarrow 0$
$p \leftarrow 0$
$q \leftarrow 0$
$r \leftarrow 0$
for $i = start$ to $end - 1$ **do**
    $f \leftarrow -s \times T.\text{subdiag}[i]$
    $h \leftarrow c \times T.\text{subdiag}[i]$
    GIVENSROTATE(g, f, c, s, r)
    if $i > start$ **then**
        $T.\text{subdiag}[i-1] \leftarrow r$
    end if
    $e \leftarrow T.\text{diag}[i] - p$
    $q \leftarrow (e - T.\text{diag}[i+1]) \times s + 2\,c\,h$
    $p \leftarrow -s\,q$
    $T.\text{diag}[i] \leftarrow e + p$
    $g \leftarrow c\,q - h$
    $Q_i \leftarrow Q.\text{col}(i)$
    $Q_{i+1} \leftarrow Q.\text{col}(i+1)$
    $Q.\text{col}(i) \leftarrow c\,Q_i - s\,Q_{i+1}$
    $Q.\text{col}(i+1) \leftarrow s\,Q_i + c\,Q_{i+1}$
end for
$T.\text{diag}[end] \leftarrow T.\text{diag}[end] - p$
$T.\text{subdiag}[end-1] \leftarrow g$

4.3. Implementation Improvements

1. **Small Sub-blocks:** If a sub-block is only size 2, SOLVE2X2BLOCK is used directly; this is faster than iterative QR steps on such a small portion.
2. **Reference to LAPACK Methodology:** Instead of fully updating T by explicit matrix multiplication, we mimic the structure in LAPACK. For details, see Appendix A for a brief justification. This approach reduces unnecessary computations.

5. Inverse Iteration

5.1. Mathematical Background

Inverse iteration is a powerful method for computing eigenvectors once an approximate eigenvalue λ of a matrix H (in our case, the tridiagonal T) is known. We solve the system

$$(H - \lambda I)\,x \;=\; b$$

for a suitable choice of b, and then normalize x. Because our matrix H is tridiagonal, the system can be solved efficiently via the Thomas algorithm.

5.1.1. Thomas Algorithm

Consider a tridiagonal linear system

$$\begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_1 & b_2 & c_2 & \cdots & 0 \\ 0 & a_2 & b_3 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & c_{n-1} \\ 0 & 0 & \cdots & a_{n-1} & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{pmatrix},$$

where $\{a_i\}$ are the subdiagonal entries, $\{b_i\}$ the diagonal entries, and $\{c_i\}$ the superdiagonal entries. The Thomas algorithm proceeds in two phases:

Forward Sweep: We introduce new coefficients c'_i and d'_i, updating them row by row:

$$c'_1 \;=\; \frac{c_1}{b_1}, \quad d'_1 \;=\; \frac{d_1}{b_1},$$

and for $i = 2, 3, \ldots, n-1$ (resp. $i = 2, 3, \ldots, n$ for d'_i):

$$c'_i \;=\; \frac{c_i}{b_i - a_i\, c'_{i-1}}, \quad d'_i \;=\; \frac{d_i - a_i\, d'_{i-1}}{b_i - a_i\, c'_{i-1}}.$$

By the end of this forward pass, we have effectively eliminated the subdiagonal below row i.

Back Substitution: We obtain the solution $\{x_i\}$ by going backward from $i = n$ down to 1:

$$x_n = d'_n, \quad x_i = d'_i - c'_i\, x_{\,i+1} \quad \text{for } i = n-1, n-2, \ldots, 1.$$

5.1.2. One-Step Inverse Iteration

To compute the eigenvector associated with an approximate eigenvalue λ, we pick an random initial vector $x^{(0)}$ and do:

1. $x^{(0)} \leftarrow \dfrac{x^{(0)}}{\|x^{(0)}\|}$.
2. Solve $(H - \lambda I)\, y = x^{(0)}$ by the Thomas algorithm.
3. $x^{(1)} \leftarrow \dfrac{y}{\|y\|}$.

Even though inverse iteration can be repeated until convergence, we often find in practice (especially if λ is already a good approximation of an eigenvalue) that one iteration suffices to yield a high-quality eigenvector.

5.1.3. Repeated Eigenvalues

If two or more eigenvalues are nearly equal, one can apply Gram–Schmidt to ensure that the resulting eigenvectors remain mutually orthogonal in the shared eigenspace.

5.2. Implementation

In this section, we describe three core functions for inverse iteration:

- `thomas`: Solves the tridiagonal system $(T - \lambda I)\, y = x$ via the Thomas algorithm.
- `computeEigenVector`: Performs one step of inverse iteration to obtain an eigenvector.
- `computeAllEigenVectors`: Computes all eigenvectors for a given set of eigenvalues by repeatedly calling `computeEigenVector` and applying Gram–Schmidt for repeated eigenvalues.

5.2.1. thomas

Algorithm 7 Thomas Algorithm for symmetric tridiagonal matrix

Require:
T, an approximate eigenvalue $\lambda \in \mathbf{R}$, the right-hand side vector $x \in \mathbf{R}^n$.

$n \leftarrow T$.size()
$b \leftarrow T$.diag $- \lambda$
$c \leftarrow T$.subdiag
$s \leftarrow$ zeros vector
$x_0 \leftarrow x_0 / b_0$
$s_0 \leftarrow c_0 / b_0$
for $i = 1$ to $n - 1$ **do** ▷ Forward Sweep
    $temp \leftarrow b_i - c_{i-1} * s_{i-1}$
    $x_i \leftarrow (x_i - c_{i-1} * x_{i-1}) / temp$
    if $i < n - 1$ **then**
        $s_i \leftarrow c_i / temp$
    end if
end for
for $i = n - 2$ to 0 **do** ▷ **Back Substitution**
    $x_i \leftarrow x_i - s_i x_{i+1}$
end for

5.2.2. computeEigenVector

Algorithm 8 One-Step Inverse Iteration

Require:
T, $\lambda \in \mathbf{R}$, initial random vector $v \in \mathbf{R^n}$, $\|v\|_2 = 1$

$v \leftarrow v / \|v\|$ ▷ Normalize
`thomas`(T, λ, v) ▷ Solve $(T - \lambda I)y = v$ in-place on v
$v \leftarrow v / \|v\|$ ▷ Re-normalize
return v

5.2.3. computeAllEigenVectors

Algorithm 9 Compute all eigenvectors from sorted eigenvalues

Require: T, vector of sorted eigenvalues $e \in \mathbf{R}^n$, tol, initial $V \in \mathbb{R}^{n \times n}$ for storing eigenvectors

```
groupStart ← 0
v_0 ← computeEigenVector(T, e_0)
V.col(0) ← v_0
for i = 1 to n − 1 do
    if |e_i − e_{i−1}| > tol then
        groupStart ← i
    end if
    v_i ← computeEigenVector(T, e_i)
    for j = groupStart to i − 1 do                ▷ Inverse iteration
        v_i ← v_i − ⟨v_i, V.col(j)⟩ V.col(j)
    end for
    v_i ← v_i/‖v_i‖
    V.col(i) ← v_i
end for
return V
```

6. Testing and Results

All experiments were conducted on a MacBook Pro with an Apple M4 Max chip, running macOS. The implementation used CMake 3.10, GCC (Apple Clang version 16.0.0), and Armadillo 14.2.

6.1. Testing

The following test cases were examined:

1. **Random Eigenvalues** for $n = 20$, $n = 100$. We generate a random Hermitian matrix as $A = Q\,\Lambda\,Q^*$ to test general accuracy and stability: without eigenvectors (only eigenvalues), eigenvectors in QR-Setp, eigenvectors by Inverse Iteration.
2. **Repeated Eigenvalues** for $n = 20$, where 10 eigenvalues are 2 and the remaining 10 are 5. This verifies handling of multiple eigenvalues.
3. **Extreme Eigenvalues** for $n = 20$, where 10 eigenvalues are extremely large (1e4 level) and the remaining 10 are extremely small (1e-4 level). This verifies handling of extreme eigenvalues.

6.2. Key Metrics

We measure:

- $\|Q^*AQ - T\|_F$ to assess tridiagonalization fidelity.
- $\|\Lambda - \Lambda_{\text{true}}\|$ to compare computed and known eigenvalues.
- $\|Q^*AQ - \Lambda\|_F$ to evaluate final diagonalization accuracy.
- $\|Q^*Q - I\|_F$ to confirm orthogonality of the eigenvectors.

In all our experiments, these norms remained small. The combination of Wilkinson shifts and deflation typically yielded rapid convergence.

6.3. Results

6.3.1. Comparison of Eigenvector Computation Methods

We compare three different approaches for obtaining eigenvalues and (optionally) eigenvectors from the same random Hermitian matrix:

1. **No eigenvector computation**: compute only eigenvalues (Householder reduction + QR iteration).
2. **QR-based accumulation**: update eigenvectors within the QR step (applying each Givens/rotations to the vector matrix).
3. **Inverse Iteration**: first obtain eigenvalues, then compute eigenvectors by separately applying inverse iteration for each eigenvalue.

The next two tables summarize the total runtimes as well as typical error metrics for the small-scale case $n = 20$ and the larger case $n = 100$.

Table 1: Comparison of methods for $\mathbf{n = 20}$ random Hermitian matrix.

Method	t_{total}	t_h	t_{qr}	t_{ii}	$\|\Lambda - \Lambda_{\text{true}}\|$	$\|Q^* A\, Q - \Lambda\|_F$	$\|Q^*Q - I\|_F$
No Eigvecs	4×10^{-5}	3×10^{-5}	1×10^{-5}	—	$\sim 10^{-15}$	—	—
QR-Step	9×10^{-5}	4×10^{-5}	5×10^{-5}	—	$\sim 10^{-16}$	$\sim 10^{-15}$	$\sim 10^{-15}$
QR+InvIter	7×10^{-4}	4×10^{-4}	1×10^{-4}	2×10^{-5}	$\sim 10^{-15}$	$\sim 10^{-14}$	$\sim 10^{-13}$

Table 2: Comparison of methods for $\mathbf{n = 100}$ random Hermitian matrix.

Method	t_{total}	t_h	t_{qr}	t_{ii}	$\|\Lambda - \Lambda_{\text{true}}\|$	$\|Q^* A\, Q - \Lambda\|_F$	$\|Q^*Q - I\|_F$
No Eigvecs	4.2×10^{-3}	4×10^{-3}	2×10^{-4}	—	$\sim 10^{-15}$	—	—
QR-Step	1.1×10^{-2}	8×10^{-3}	3×10^{-3}	—	$\sim 10^{-15}$	$\sim 10^{-14}$	$\sim 10^{-14}$
QR+InvIter	8.7×10^{-3}	8×10^{-3}	2×10^{-4}	5×10^{-4}	$\sim 10^{-15}$	$\sim 10^{-10}$	$\sim 10^{-10}$

1. Inverse iteration becomes slightly faster than QR-based accumulation for larger matrices, likely due to overheads involved in continuously updating all eigenvectors during each QR step.
2. The orthogonality measure $\|Q^*Q - I\|_F$ for inverse iteration is somewhat higher (about 10^{-10}) than for QR accumulation (about 10^{-14}), but remains acceptable in practice.
3. Eigenvalue accuracy remains $\sim 10^{-15}$ for both approaches, confirming that either method provides reliable spectral information.

6.3.2. Repeated and Extreme Eigenvalues

We now consider two special scenarios, focusing on the final numerical accuracy rather than the runtime:

1. **Repeated eigenvalues:** half of the eigenvalues set to 2, and the other half set to 5.
2. **Extreme eigenvalues:** half of the eigenvalues are on the order of 10^4, and the other half on the order of 10^{-4}.

1. Despite having 10 eigenvalues exactly equal to 2 and another 10 equal to 5, the method detects and converges to the repeated eigenvalues with no noticeable difficulty.

Table 3: Summary of errors in special cases ($\mathbf{n} = \mathbf{20}$, focus on numerical accuracy).

Scenario	$\|\Lambda - \Lambda_{\text{true}}\|$	$\|Q^*AQ - \Lambda\|_F$	$\|Q^*Q - I\|_F$
Repeated Eigenvalues	$\sim 10^{-14}$	$\sim 10^{-13}$	$\sim 10^{-14}$
Extreme Eigenvalues	$\sim 10^{-12}$	$\sim 10^{-10}$	$\sim 10^{-7}$

2. With extremely large and small eigenvalues mixed, the algorithm still achieves near-unit-roundoff accuracy for the eigenvalues themselves (around 10^{-12} error).
3. The orthogonality measure increases slightly for the extreme-value case (up to 10^{-7}), but remains acceptable for most practical purposes.

6.3.3. Remarks

1. In all tested situations, the combination of Householder tridiagonalization, Wilkinson shifts, and deflation reliably converges and produces accurate eigenvalues.
2. For eigenvectors, users can choose between QR accumulation (simpler in code) or inverse iteration (faster, though with slightly higher orthogonality errors).
3. Even in the presence of repeated or extreme eigenvalues, the algorithm maintains good numerical stability with no major convergence issues.

7. Conclusion

In summary, this project provides four key contributions:

1. Converting any Hermitian matrix into a real symmetric tridiagonal form using Householder transformations.
2. Using a QR iteration with Wilkinson shifts (along with sub-block reduction and convergence checks) to efficiently compute eigenvalues and eigenvectors of that tridiagonal matrix.
3. Using Inverse iteration to compute eigenvectors from eigenvalues.
4. Verifying numerical accuracy and convergence speed through multiple test scenarios, including random eigenvalues, repeated eigenvalues, and large/small extreme eigenvalues.

Experimental results show that applying Householder reduction to first simplify the matrix, followed by a shifted QR iteration, not only produces high-accuracy eigenvalues but also allows two strategies for eigenvector computation. The first approach accumulates Givens rotations directly during the QR process, while the second approach computes all eigenvalues first and then uses inverse iteration to recover each eigenvector.

For matrices of moderate size, both approaches offer good accuracy and efficiency; for larger matrices, inverse iteration can be further optimized.

Moreover, our tests confirm that even for special cases—such as repeated eigenvalues or vastly differing magnitudes—this method remains robust and maintains strong numerical performance.

Overall, the approach is both general and efficient for solving Hermitian eigenvalue problems, laying a solid foundation for even larger-scale or parallel implementations.

Appendix A Proof of the Algorithm QR-Step Correctness

In this appendix, we provide a detailed argument showing that the QR-Step update like the algorithm yields the same result as explicitly applying the corresponding Givens transformations.

Setup

Let $T \in \mathbb{R}^{n\times n}$ be a real symmetric tridiagonal matrix of the form

$$T = \begin{pmatrix} a_1 & b_1 & z_1 & & & \\ b_1 & a_2 & b_2 & z_2 & & \\ z_1 & b_2 & \ddots & \ddots & \ddots & \\ & z_2 & \ddots & a_{n-2} & b_{n-2} & z_{n-2} \\ & & \ddots & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & z_{n-2} & b_{n-1} & a_n \end{pmatrix}, \quad z_i = 0 \text{ for all } i.$$

Let $\sigma \in \mathbb{R}$ be the chosen Wilkinson shift for this iteration. We aim to show that the iterative **QR-Step** procedure updates T exactly as a direct application of Givens rotations would do (which we refer to as "Algorithm 2.5.8" in the lecture notes).

Initialization

We begin by defining the following initial parameters:

$$g_0 = a_1 - \sigma, \quad s_0 = -1, \quad c_0 = 1, \quad p_0 = q_0 = r_0 = 0.$$

At each step k, certain local variables $(g_k, s_k, c_k, p_k, q_k, r_k)$ are updated in a way that corresponds to multiplying T by the appropriate Givens matrix from both sides.

Case $k = 1$

At the first step, we compute

$$f_1 = -\,s_0\, b_1 = b_1, \qquad h_1 = c_0\, b_1 = b_1.$$

We then form the Givens matrix

$$G_{12} = G_{12}[g_0, f_1] = G_{12}\big(a_1 - \sigma,\ b_1\big),$$

yielding the parameters (c_1, s_1, r_1). According to the QR-Step algorithm, we next update:

$$\begin{aligned} e_1 &= a_1 - p_0 = a_1, \\ q_1 &= (e_1 - a_2)\, s_1 \ + \ 2\, c_1\, h_1 = (a_1 - a_2)\, s_1 \ + \ 2\, c_1\, b_1, \\ p_1 &= -\,s_1\, q_1 = -\big[(a_1 - a_2)\, s_1^2 \ + \ 2\, c_1\, s_1\, b_1\big], \\ a_1^{(*)} &= g_1 + p_1 = \big(a_1 - \sigma\big) - (a_1 - a_2)\, s_1^2 \ - \ 2\, c_1\, s_1\, b_1 = c_1^2\, a_1 \ - \ 2\, c_1\, s_1\, b_1 \ + \ s_1^2\, a_2, \\ g_1 &= c_1\, q_1 - h_1 = (a_1 - a_2)\, c_1\, s_1 \ + \ 2\, c_1^2\, b_1 \ - \ b_1 = (c_1^2 - s_1^2)\, b_1 \ + \ c_1\, s_1\, (a_1 - a_2). \end{aligned}$$

On the other hand, if one directly applies $(G_{12})^T$ and G_{12} to T in matrix form, we obtain:

$$\begin{aligned}
a_1^{(*)} &= c_1^2\, a_1 \;-\; 2\, c_1\, s_1\, b_1 \;+\; s_1^2\, a_2,\\
a_2^{(1)} &= s_1^2\, a_1 \;+\; 2\, c_1\, s_1\, b_1 \;+\; c_1^2\, a_2 = a_2 - p_1,\\
b_1^{(1)} &= (c_1^2 - s_1^2)\, b_1 \;+\; c_1\, s_1\, (a_1 - a_2) = g_1,\\
z_1^{(1)} &= -\, s_1\, b_2,\\
b_2^{(1)} &= c_1\, b_2.
\end{aligned}$$

We thus see that $a_1^{(*)}$ matches exactly, Notice that $z_1^{(1)} = f_2$, $b_2^{(1)} = h_2$, and $a_2^{(1)} = e_2$ in the next iteration. Hence, the first pivot element a_1 is updated exactly as in a direct Givens rotation of T. The updated matrix T becomes

$$T = \begin{pmatrix} a_1^{(*)} & b_1^{(1)} & z_1^{(1)} & \\ b_1^{(1)} & a_2^{(1)} & b_2^{(1)} & \ddots \\ z_1^{(1)} & b_2^{(1)} & a_3 & \ddots \\ & \ddots & \ddots & \ddots \end{pmatrix}_{n\times n} = \begin{pmatrix} a_1^{(*)} & g_1 & f_2 & \\ g_1 & e_2 & h_2 & \ddots \\ f_2 & h_2 & a_3 & \ddots \\ & \ddots & \ddots & \ddots \end{pmatrix}_{n\times n},$$

Case $k = 2$

Next, we eliminate $z_1^{(1)}$. We define

$$G_{23} = G_{23}\big(b_1^{(1)},\, z_1^{(1)}\big) = G_{23}(g_1,\; f_2),$$

which produces (c_2, s_2, r_2). After applying G_{23}, we have

$$z_1^{(*)} = 0, \quad b_1^{(*)} = r_2.$$

Following the same algorithmic update:

$$\begin{aligned}
e_2 &= a_2^{(1)},\\
q_2 &= \big(a_2^{(1)} - a_3\big)\, s_2 \;+\; 2\, c_2\, b_2^{(1)},\\
p_2 &= -\left[\big(a_2^{(1)} - a_3\big)\, s_2^2 \;+\; 2\, c_2\, s_2\, b_2^{(1)}\right],\\
a_2^{(*)} &= c_2^2\, a_2^{(1)} \;-\; 2\, c_2\, s_2\, b_2^{(1)} \;+\; s_2^2\, a_3,\\
g_2 &= (c_2^2 - s_2^2)\, b_2^{(1)} \;+\; c_2\, s_2\, (a_2^{(1)} - a_3).
\end{aligned}$$

One can verify, by the same reasoning as in the $k = 1$ case, that the updated values for a_2 and b_1 again match the direct Givens rotation result. This consistency holds at each subsequent step, carrying us through $k = n - 2$, ensuring that a_{n-1} and b_{n-2} are properly updated.

Final Step

After all intermediate Givens transformations have been performed, the matrix T remains tridiagonal. The last iteration provides the final values:

$$b_{n-1}^{(*)} = b_{n-1}^{(1)} = g_{n-1}, \qquad a_n^{(*)} = a_n^{(1)} = a_n^{(*)} - p_{n-1}.$$