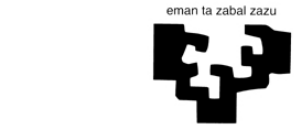

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Implementing the QR algorithm for efficiently computing matrix eigenvalues and eigenvectors

Final Degree Dissertation
Degree in Mathematics

Gorka Eraña Robles

Supervisor:
Ion Zaballa Tejada

Leioa, June 27, 2017

Contents

Introduction

The computation of eigenvalues is unavoidably iterative. This is a consequence of their definition −and how this leads to compute them as roots of the characteristic polynomial− and Abel's famous proof that there is no algebraic formula for the roots of a general polynomial of degree greater than four. The QR algorithm is one of the most important methods for computing both eigenvalues and eigenvectors and for the general, nonsymmetric eigenvalue problem it is the king. This work builds up to the ultimate algorithm, the shifted Hessenberg QR algorithm, by starting with simpler ones.

The first chapter is a brief summary of already familiar concepts that are mentioned and used throughout the work. Its goal is to make the dissertation self-contained.

The second chapter develops both the power and inverse power methods. The convergence of the power iteration is studied in terms of subspace convergence and the implementation details of both methods are taken care of.

The third chapter establishes the convergence of the QR algorithm by linking the shifted QR routine with the inverse power method and the unshifted QR algorithm with the power method. Then, in order to solve a practical problem inherent to the QR iteration, it introduces the Hessenberg form of a matrix and demonstrates how to reduce any matrix to Hessenberg form.

(This is not the only way to prove the convergence of the QR iteration. Wilkinson in [7] connects the QR method with simultaneous iteration. Then Buurema in [6] gives a geometric proof of QR's convergence with and without shifts using simultaneous iteration and Krylov subspace ideas. And Francis and Kublanovskaya had ideas of their own that will be developed later on this introduction.)

Fourth and fifth chapters implement two different versions of the QR algorithm: the explicitly shifted QR routine and the implicitly shifted QR routine. Both methods reduce any given matrix to a Schur form. While the first algorithm reduces a matrix to a complex Schur form, the second one reduces it to a real Schur form.

Finally, appendix A contains the MATLAB implementations of all the

algorithms given in the work and appendix B analyzes the precision and accuracy of both implementations of the QR algorithm.

Regarded by some as 'one of the jewels in the crown of matrix computation' the QR algorithm was a 'genuinely new contribution to the field of numerical analysis and not a refinement of ideas given by Newton, Gauss, Hadamard, or Schur.' In any case, if we 'have seen further than others, it is by standing upon the shoulders of giants'. And those giants happen to be an English man and a Russian woman. The QR iteration was simultaneously developed in the late 50's and early 60's by John Francis in England and Vera Kublanovskaya in the Soviet Union, but it is the firsts work that has had more influence simply because he dealt with the implementation and applications of the method.

John G. F. Francis was born in London in 1934 and in October 1961 −the time of the publication of his papers on the then called 'the QR transformation'− he was working for the National Research and Development Corporation (NRDC). He started to work there in 1954 and although he entered Christ's College at Cambridge University in 1955 to study math, he did not complete a degree and returned to NRDC in 1956. He left this job in 1961 and dropped all connections with numerical analysis.

Francis developed the whole QR algorithm on his own, apart from the influence of different papers, such as Rutishauser's [11]. He did not collaborate with anybody: the ideas, theorems and implementation of QR were all his. On the one hand, his knowledge of Fike's [12] and Wilkinson's work on the stability of the unitary matrix transformation helped John shift the emphasis from the Gaussian elimination-based LR algorithm of Rutishauser to the use of orthogonal Householder and Givens eliminations in QR. These make the algorithm accurate and backward stable. On the other hand, his awareness of the power of inverse interation for finding eigenvalues accurately from approximations inspired him to apply shifts to QR; and the development of the Implicit Q Theorem paves the way to obtain complex conjugate eigenvalue pairs of real matrices in real arithmetic. These ensure the fast convergence of the method.

The first of his papers, [8], starts by proving the convergence of the lower triangle entries of the unshifted explicit QR iterates A^k to zero, assuming that A is nonsingular and has eigenvalues of distinct moduli. Then proceeds to establish the invariance of normalcy, symmetry and Hermitianness of a matrix during QR iterations. He accounts for the computational saving that means first reducing the matrix A to Hessenberg form −it reduces each step from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$−, proposes elimination methods for achieving such form, and emphasizes in both the convergence of the algorithm to the Schur form of A and the rate of convergence of the iterates' subdiagonal entries a_{ij}^k as $(|\lambda_i/\lambda_j|)^k$.

The second paper [9] deals with the implementation of the algorithm.

First of all, the Implicit Q Theorem is established thus helping modify the QR algorithm from its explicit factorization to the implicit one. It is noted that complex conjugate shifts of explicit QR can be performed simultaneously in one implicit double real step. Bulge chasing is introduced: it creates a bulge that protrudes from the Hessenberg form and then chases it down the diagonal until re-establishing the Hessenberg form. The near singularity of $A - \alpha I$ when α is close to an eigenvalue helps him deflate the last row of A to save some computations. Implementation of Givens rotations and Householder eliminations are discussed for ensuring backward stability and it even details a close precursor of what nowadays is known as the Wilkinson shift.

Vera Nikolaevna Kublanovskaya, on the other hand, was born on 21 November 1920 in Vologda Oblast, Russia and passed away in February 21, 2012. In 1945, after surviving the siege of Leningrad she was accepted to study mathematics at Leningrad State University. She joined the Leningrad branch of the Steklov Mathematical Institute of the USSR Academy of Sciences after graduating in 1948 and worked there at least until 2009.

Kublanovskaya started to develop her version of the QR method inspired by [11]. She presented a matrix $A = L \cdot Q$ where L is a lower triangular matrix and Q is orthogonal −she proposed constructing the last one by using elementary rotations or reflections. Her algorithm factors $A = A_1 = L_1 \cdot Q_1$, then multiplies $A_2 = Q_1 \cdot L_1$ and factors $A_2 = Q_2 \cdot L_2$ etc. Her first paper explains the basics of this factorization and reverse-order multiplication for nonsingular matrices A with real and distinct modulus eigenvalues; and introduces the linear convergence of the diagonal entries of the matrix L_k to the eigenvalues of such matrices. The second paper proves that the convergence happens at least linearly and improves it by introducing simple shifts. Finally, in a third paper, Kublanovskaya indicates how the LQ method can be adapted to find the singular values of A, that is, the eigenvalues of the product AA^* or A^*A, without explicitly computing the matrix products.

Neither Kublanovskaya nor her colleagues implemented her method during this time or later. She was not involved in further investigations of the QR itself, but some of her work can be regarded as an extension of the underlying ideas of the QR method.

Chapter 1

Previous concepts

1.1 Notation and basic results

The goal of this dissertation is to comprehend how every available linear algebra software library computes the eigenvalues and eigenvectors of any given matrix A. Throughout the work the nature of the topic we will be treating will ask for either complex or real matrices. So at the start of every chapter we will indicate whether we work with $A \in \mathbb{C}^{n\times n}$ or $A \in \mathbb{R}^{n\times n}$, that is, $A \in \mathbb{F}^{n\times n}$ where $\mathbb{F} = \mathbb{C}$ or $\mathbb{F} = \mathbb{R}$. For this chapter, let $\mathbb{F} = \mathbb{C}$.

Scalars will be appointed with greek letters as λ or μ, vectors with lower case letters $-x$, y, etc.$-$ and matrices with upper case letters $-A$, B, etc.$-$. A^t will designate the transpose of a matrix, i.e., if

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \text{ then } A^t = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{nn} \end{bmatrix}.$$

A^* indicates the conjugate transpose of a matrix, that is,

$$A^* = \begin{bmatrix} \overline{a_{11}} & \cdots & \overline{a_{n1}} \\ \vdots & \ddots & \vdots \\ \overline{a_{1n}} & \cdots & \overline{a_{nn}} \end{bmatrix},$$

where $\overline{z}$ is the conjugate of the complex number z. Notice that if A is a real matrix, $A^t = A^*$.

A complex matrix U is unitary if

$$UU^* = U^*U = I,$$

and these kind of matrices hold another interesting equivalence:

- U is unitary.
- The rows $-$and columns$-$ of U form an orthonormal basis of $\mathbb{F}^{n\times n}$.

Unitary matrices are usually called *orthogonal* when they are real.

Now, the first thing we ought to do is define the eigenvalues and eigenvectors:

Definition 1.1.1. Let A be of order n. The pair (λ, x) is called an *eigenpair* or *right eigenpair* of A if

- $x \neq 0$, and
- $Ax = \lambda x$.

The scalar λ is called an *eigenvalue* and the vector x is called an *eigenvector*. On the other hand, the pair (λ, y) is called a *left eigenpair* if

- $y \neq 0$, and
- $y^*A = y^*\lambda$.

Real matrices may have complex eigenvalues and eigenvectors: take for example

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

Its eigenvalues are i and $-i$, and its eigenvectors $\begin{bmatrix} i \\ 1 \end{bmatrix}$ and $\begin{bmatrix} -i \\ 1 \end{bmatrix}$.

Notice that if (λ, x) is an eigenpair, then $(\lambda, \tau x)$ is also an eigenpair. Thus in order to eliminate this trivial nonuniqueness we might require, for example, $x^*x = 1$. Nevertheless, this does not eliminate the nonuniqueness of the eigenvectors, e.g., there may be many unitary vectors. That is why in some contexts −precisely in §2.1.1−, we may refer to the linear subspace spanned by an eigenvector.

Definition 1.1.2. Let V be a vector space over a field K, and S a set of vectors. Then *the subspace spanned by* S is the set of all finite linear combinations of elements of S, i.e.,

$$\langle S \rangle = \left\{ \sum_{i=1}^{k} \lambda_i v_i : k \in \mathbb{N}, v_i \in S \text{ and } \lambda_i \in K \right\}.$$

If $S = \{v_1, ..., v_n\}$, we may write $\langle S \rangle = \langle v_1, ..., v_n \rangle$. In this work, depending on the context, $K = \mathbb{R}$ or $K = \mathbb{C}$ and $V = \mathbb{R}^n$ or $V = \mathbb{C}^n$.

Secondly, as it will be proved later, the QR algorithm computes the eigenvalues of any given matrix by reducing it to its Schur form.

Theorem 1.1.1 (Schur)**.** *Let A be of order n. Then there is a unitary matrix U such that*

$$U^*AU = T$$

where T is upper triangular. By appropriate choice of U, the eigenvalues of A, which are the diagonal elements of T may be made to appear in any order.

Proof. Theorem 1.12 of [1]. □

Actually, the QR iteration can be implemented in two different ways: the first one reduces the given matrix to the complex Schur form −a complex upper triangular matrix with both the real and complex eigenvalues in the diagonal−, and the real Schur form −a real block upper triangular matrix where the blocks of order one contain the real eigenvalues and the ones of order two contain the complex conjugate eigenvalues. Chapters 4 and 5 are devoted to the complex and real Schur forms, respectively. A proof of the less known real Schur form will be provided in Chapter 5.

Thirdly, we must talk about the reliability of the algorithms. The routines given troughout the dissertation are designed so that they are either backwards stable −Definition 1.1.3−, or stable in the usual sense −Definition 1.1.4. These definitions make the algorithms accurate and their output reliable.

Definition 1.1.3. An algorithm $\tilde{f} : X \longrightarrow Y$ for a problem $f : X \longrightarrow Y$ is said to be *backwards stable* if for all $x \in X$ there exists $\tilde{x} \in X$ such that

$$\frac{\|\tilde{x} - x\|}{\|x\|} = \mathcal{O}(\epsilon_M)$$

and

$$\tilde{f}(x) = f(\tilde{x}).$$

In our case, if we are working with complex matrices, then $X = Y = \mathbb{C}^{n\times n}$, but $X = Y = \mathbb{R}^{n\times n}$ if real matrices are considered.

Definition 1.1.4. Let the sequence $A_1, A_2, ..., A_m$ be generated by the recursion

$$A_{k+1} = \text{fl}[R_k A_k S_k], \text{ for every } k = 1, ..., m-1 \tag{1.1}$$

where R_k and S_k are unitary matrices generated from A_k. fl denotes floating-point computation with rounding unit ϵ_M and includes any errors made in generating and applying R_k and S_k. Set

$$P = R_{m-1} \cdots R_1 \text{ and } Q = S_1 \cdots S_{m-1}.$$

Then we say that the sequence (1.1) is *stable in the usual sense* if there is a matrix E and a slowly growing function γ_M such that

$$\frac{\|E\|_2}{\|A\|_2} = \gamma_M \epsilon_M$$

and

$$A_m = P(A_1 + E)Q.$$

A couple of words must be said about vector and matrix norms too. We will mainly use the 2-norm, $\|\cdot\|_2$, of either a vector or a matrix and the Frobenius norm, $\|\cdot\|_F$, of a matrix. We will just give the definitions and main properties of them. Further information can be found in [17] and [18].

Definition 1.1.5. (i) For every vector $x \in \mathbb{C}^n$, the vectorial norm $\|\cdot\|_2$ is defined as

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}.$$

(ii) For every matrix $A \in \mathbb{C}^{n\times m}$, $\|\cdot\|_2$ is the matrix norm induced by the norm ℓ_2, i.e.,

$$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 .$$

(iii) For every matrix $A \in \mathbb{C}^{n\times m}$, the matrix norm $\|\cdot\|_F$ is defined as

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} |a_{ij}|^2}.$$

And a little list of properties that will be used throughout the work:

Proposition 1.1.2. *Let $A \in \mathbb{C}^{n\times m}$ and $x \in \mathbb{C}^n$. Then:*

(i) $\|x\|_2^2 = x^* x$.

(ii) *For every $x \in \mathbb{C}^n$*

$$\|x\|_2 = \|x^*\|_2 .$$

For every $A \in C^{n\times m}$

$$\|A\|_F = \|A^*\|_F \text{ and } \|A\|_2 = \|A^*\|_2 .$$

(iii) *Matrix norms $\|\cdot\|_2$ and $\|\cdot\|_F$ are unitarily invariant. That is for every $A \in \mathbb{C}^{n\times m}$ if $U \in \mathbb{C}^{r\times n}$ $-$or $\mathbb{C}^{m\times s}-$ is unitary then*

$$\|UA\|_2 = \|AU\|_2 = \|A\|_2 \text{ and } \|UA\|_F = \|AU\|_F = \|A\|_F .$$

(iv) *Matrix norms* $\|\cdot\|_2$ *and* $\|\cdot\|_F$ *are consistent, i.e., for every* $A \in \mathbb{C}^{n\times m}$ *and* $B \in \mathbb{C}^{m\times s}$

$$\|AB\|_2 \leq \|A\|_2 \|B\|_2 \text{ and } \|AB\|_F \leq \|A\|_F \|B\|_F .$$

(v) *All the norms defined in a vectorial space* V *of finite dimension are equivalent. That is, if* μ *and* ν *are two norms defined in* V*, then there exist* $a, b \in \mathbb{R}$ *such that*

$$a \leq \frac{\mu(x)}{\nu(x)} \leq b \text{ for all } x \in V.$$

In our case, $\|\cdot\|_2$ *and* $\|\cdot\|_F$ *are norms defined in the vectorial space* $\mathbb{C}^{n\times m}$*. Thus there exist* $c, d \in \mathbb{R}$*, such that*

$$c \leq \frac{\|A\|_2}{\|A\|_F} \leq d \text{ for all } A \in \mathbb{C}^{n\times m}.$$

Proof. Check [17]. □

Finally, a note regarding conditioning. In the field of numerical analysis the condition number of a function measures how much the output value of the function can change for a small change in the input argument. In other words, it measures how sensitive the output of a function is to changes on the input. A problem is said to be *well-conditioned* if its condition number is small and *ill-conditioned* if its condition number is high. Sadly, we do not have neither the time nor the space to study the conditioning of the Schur decomposition through the QR routine. A proper analysis of the QR iteration should consider and take into account the condition number. At least when studying the precision of the algorithm; to check, for example, whether those matrices with worst backwards error have the highest condition number or not.

Chapter 2

The power and inverse power methods

2.1 The power method

The power method is based on a simple idea. Let A be a complex nondefective matrix and (λ_i, x_i) a complete set of eigenpairs. All x_i are linearly independent, so if $u_0 \in \mathbb{C}^n$ then

$$u_0 = \gamma_1 x_1 + \cdots + \gamma_n x_n \tag{2.1}$$

uniquely. As $Ax_i = \lambda_i x_i$, then $A^k x_i = \lambda_i^k x_i$, thus

$$A^k u_0 = \gamma_1 \lambda_1^k x_1 + \cdots + \gamma_n \lambda_n^k x_n.$$

Assuming that both γ_1 and λ_1^k are not zero

$$\frac{1}{\gamma_1 \lambda_1^k} A^k u_0 = x_1 + \sum_{j=2}^{n} \frac{\gamma_j}{\gamma_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k x_k. \tag{2.2}$$

Now, if $|\lambda_1| > |\lambda_j| \ \ \forall j \in \{2, ..., n\}$, i.e., λ_1 is a dominant eigenvalue, the summation on the right part of (2.2) will approach to 0 as $k \to \infty$. Therefore the left part of the equality will become an increasingly accurate approximation to the dominant eigenvector x_1.

Notice that, for now, we do not know how to compute eigenvalues using this method. We can summarize this superficial knowledge of the power

iteration in Algorithm 1:

```
   Input  : matrix A, number of iterations n.
   Output: approximate eigenpair (λ, x)
1  PowerMethod (A, n)
2  |  Select starting unitary vector u_0
3  |  for k ∈ {1, ..., n} do
4  |  |  u_1 ← A u_0
5  |  |  u_0 ← u_1 / ‖u_1‖
6  |  end
7  |  return u_0
8  end
```

Algorithm 1: Initial approach to the power method.

2.1.1 The convergence of the power method: subspace convergence

Truth is, when computing eigenvectors we are not interested in the vector per se, but rather in the linear subspace spanned by that eigenvector. Take for example the following matrix and starting vector:

$$A = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \text{ and } u_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Note that u_0 is an eigenvector of A, consequently any multiple of that vector will be an eigenvector too. Now let us look at the first four iterations of the power method:

$$Au_0 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \; A^2u_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \; A^3u_0 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \text{ and } A^4u_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

The sequence of eigenvectors produced by the power method seems not to converge to u_0, however, it converges to $\langle u_0 \rangle$, the linear subspace spanned by u_0. As a result, we will have to study the convergence of the power method in terms of subspace convergence. This can be achieved by identifying the set of subspaces of dimension d with a quotient space of matrices and defining the quotient topology on it. For this purpose the Grassmanian, the set of linear subspaces of the vector space $\mathbb{C}^n$ of given dimension d, will be needed. That is,

$$\mathrm{Gr}_d(\mathbb{C}^n) = \{\mathcal{S} \leq \mathbb{C}^n : \dim \mathcal{S} = d\}.$$

First, we start by establishing a connection between linear subspaces and matrices:

Definition 2.1.1. Let $\mathcal{S}$ be a subspace of $\mathbb{C}^n$ of dimension d and $\{x_1, ..., x_d\}$ a basis for it. Then

$$X = \begin{bmatrix} x_1 & \cdots & x_d \end{bmatrix}$$

is a *base-matrix* of $\mathcal{S}$.

It is clear that if $X \in \mathbb{C}^{n\times d}$ and $\operatorname{rank}(X) = d$, then X spans a subspace of dimension d, but distinct matrices may create the same subspace:

Lemma 2.1.1. *Let $X_1, X_2 \in \mathbb{C}^{n\times d}$. X_1 and X_2 generate the same subspace if and only if* $\operatorname{rank}(X_1) = \operatorname{rank}(X_2) = d$ *and there exists $P \in \mathbb{C}^{d\times d}$ invertible such that*

$$X_1 = X_2 \cdot P.$$

Unfortunately, we lack the space to develop properly the topology of $\mathrm{Gr}_d(\mathbb{C}^n)$ and prove every result needed to establish correctly the convergence of subspaces. A decorous proof can be found in the notes to the course *Advanced Numerical Methods* imparted by Ion Zaballa at UPV/EHU, on pages 216 through 226 of [16].

The results that are fundamental to understand and establish the convergence of the power iteration are the ones that follow.

Theorem 2.1.2. *Let $\{\mathcal{S}_k\}_{k=0,1,\dots} \subset \mathrm{Gr}_d\,\mathbb{C}^n$ be a sequence of subspaces and let $\mathcal{S} \in \mathrm{Gr}_d(\mathbb{C})$. Let $Q \in \widetilde{\mathcal{M}}_{n,d}(\mathbb{C})$ be an orthonormal base-matrix of $\mathcal{S}$ and, for all $k \in \{0,1,\dots\}$, let $Q_k \in \widetilde{\mathcal{M}}_{n,d}(\mathbb{C})$ be an orthonormal base-matrix of $\mathcal{S}_k$. Then*

$$\mathcal{S}_k \to \mathcal{S} \iff \forall\, k \in \{0,1,\dots\}\ \exists Z_k \in \mathbb{C}^{d\times d} \text{ unitary such that } Q_k Z_k \to Q.$$

Proof. Theorem 1.5.2 of [15]. □

Theorem 2.1.3. *Let $X \in \mathbb{C}^{n\times d}$ and let $A \in \mathbb{C}^{n\times n}$ be a matrix such that $\operatorname{rank}(A^k X) = d$ for all $k \in \{0,1,2,\dots\}$. Then if the sequence of subspaces $\{\langle A^k X\rangle\}$ converges, it does so to a subspace $\mathcal{S} \in \mathrm{Gr}_d(\mathbb{C}^n)$ that is A-invariant.*

Proof. Theorem 10.4 of [16]. □

And the convergence of the power method to an eigenvector is a consequence of this last two results:

Corollary 2.1.4. *Let $A \in \mathbb{C}^{n\times n}$ and let $q \in \mathbb{C}^n$ be a unitary vector. If the sequence of subspaces $\left\langle \dfrac{A^k q}{\|A^k q\|_2} \right\rangle$ converges to a nonzero subspace $< y >$, then y is a unitary eigenvector of A.*

Proof. Let us assume that $\langle A^k q\rangle \longrightarrow \langle y\rangle$, then by Theorem 2.1.3, $\langle y\rangle$ is A-invariant, i.e.,

$$Ay \in \langle y\rangle \iff Ay = \alpha y;$$

which means that y is an eigenvector of A. Now,

$$\langle A^k q\rangle \longrightarrow \langle y\rangle \iff \exists \alpha_k \text{ where } \alpha_k A^k q \longrightarrow y,$$

and as $\|\cdot\|_2$ is a continuous function, then

$$\left\|\alpha_k A^k q\right\|_2 \longrightarrow \|y\|_2$$

and

$$|\alpha_k| \left\|A^k q\right\|_2 \longrightarrow \|y\|_2 .$$

We can deduct that

$$\frac{\langle |\alpha_k| A^k q\rangle}{|\alpha_k| \|A^k q\|_2} \longrightarrow \frac{\langle y\rangle}{\|y\|_2},$$

and finally,

$$\left\langle \frac{A^k q}{\|A^k q\|_2} \right\rangle \longrightarrow \left\langle \frac{y}{\|y\|_2} \right\rangle = \langle y'\rangle$$

where y' is a unitary eigenvector. □

That is, if the power method converges, it does so to a unitary eigenvector. The following theorem provides a sufficient and necessary condition for the convergence of $\left\langle \frac{A^k q}{\|A^k q\|_2} \right\rangle$ for general matrices A and vectors q.

Theorem 2.1.5. *Let $q_0 \in \mathbb{C}^n$ be a unitary vector, let $A \in \mathbb{C}^{n\times n}$ be a diagonalizable matrix and let $\lambda_1, ..., \lambda_n$ be its eigenvalues. Let $\{v_1, ..., v_n\}$ be a basis of unitary eigenvectors of A such that $Av_i = \lambda_i v_i$ for all $i \in \{1, ..., n\}$ and write q_0 in the following way*

$$q_0 = \alpha_1 v_1 + \cdots + \alpha_n v_n. \tag{2.3}$$

Let $q_k = \dfrac{A^k q_0}{\|A^k q_0\|}$ and assume also that

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n| . \tag{2.4}$$

Then, $\langle q_k\rangle \longrightarrow \langle v_1\rangle$ if and only if $\alpha_1 \neq 0$. Moreover, if $\{\langle q_k\rangle\}$ converges, then for all $k \in \{0, 1, 2, ...\}$ there exist $z_k \in \mathbb{C}$ such that $|z_k| = 1$ and

$$\|z_k q_k - v_1\|_2 = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k \right).$$

Proof. Firstly, let us prove that the convergence of the power method only happens when α_1 in (2.3) is nonzero.

$\underline{\Longrightarrow)}$ Let us assume that $\langle q_j\rangle \to \langle v_1\rangle$. Then by Theorem 2.1.2 there exist $\beta_j \in \mathbb{C}$ such that $\beta_j q_j \to v_1$. Now, let P be the projection of $\langle v_1\rangle$ along $\langle v_2, ..., v_n\rangle$. (This projection is called the spectral projection along the

eigensubspace $\langle v_1 \rangle$.) Then, $P(\beta_j q_j) \to Pv_1 = v_1$. Here, since q_j is the jth iterate through the power method, $q_j = \frac{A^j q_0}{\|A^j q_0\|_2}$ and by (2.3)

$$\frac{A^j q_0}{\|A^j q_0\|_2} = \frac{1}{\|A^j q_0\|_2}\left(\alpha_1 A^j v_1 + \cdots + \alpha_n A^j v_n\right) = \frac{1}{\|A^j q_0\|_2}\left(\alpha_1 \lambda_1^j v_1 + \cdots + \alpha_n \lambda_n^j v_n\right).$$

Thus,

$$P(\beta_j q_j) = \frac{\beta_j}{\|A^j q_0\|_2} P(A^j q_0) = \frac{\beta_j}{\|A^j q_0\|_2} P\left(\alpha_1 \lambda_1^j v_1 + \cdots + \alpha_n \lambda_n^j v_n\right) = \frac{\beta_j}{\|A^j q_0\|_2} \alpha_1 \lambda_1^j v_1.$$

That is, $\frac{\beta_j}{\|A^j q_0\|_2} \alpha_1 \lambda_1^j v_1 \to v_1$ and since $v_1 \neq 0$, then $\alpha_1 \neq 0$.

$\underline{\Longleftarrow)}$ First,

$$\langle q_j \rangle = \left\langle \frac{A^j q_0}{\|A^j q_0\|} \right\rangle$$

and by Corollary 2.1.4 if this subspace converges then it does so to a subspace generated by an eigenvalue, that is,

$$\left\langle \frac{A^j q_0}{\|A^j q_0\|_2} \right\rangle \longrightarrow \langle y \rangle \text{ where } y \text{ is an eigenvalue of A.}$$

Secondly, using the decomposition (2.3) and since λ_i is an eigenvalue of A for every $i \in \{1, ..., n\}$,

$$A^j q_0 = \alpha_1 \lambda_1^j v_1 + \cdots + \alpha_n v \lambda_n^j v_n \Longrightarrow$$

$$\frac{A^j q_0}{\|A^j q_0\|_2} = \frac{\alpha_1 \lambda_1^j}{\|A^j q_0\|_2} v_1 + \cdots + \frac{\alpha_n \lambda_n^j}{\|A^j q_0\|_2} v_n = \alpha_1' \lambda_1^j v_1 + \cdots + \alpha_n' \lambda_n^j v_j.$$

By hypothesis, $\alpha_1 \neq 0$ then $\alpha_1' \neq 0$, and $\lambda_1 \neq 0$, thus

$$\frac{A^j q_0}{\alpha_1' \lambda_1^j \|A^j q_0\|_2} = v_1 + \sum_{i=2}^{n} \frac{\alpha_i'}{\alpha_1'} \frac{\lambda_i^j}{\lambda_1^j} v_i. \tag{2.5}$$

Here, following hypothesis (2.4), $\frac{|\lambda_i|}{|\lambda_1|} < 1$ for all $i \in \{1, ..., n\}$, thus $\frac{|\lambda_i|^j}{|\lambda_1|^j} \xrightarrow[j \to \infty]{} 0$. Then as j tends to infinity,

$$\frac{A^j q_0}{\alpha_1' \lambda_1^j \|A^j q_0\|_2} \longrightarrow v_1$$

and by Theorem 2.1.2, $\langle q_j \rangle \to \langle v_1 \rangle$.

Assuming that A is nondefective and has a dominant eigenvalue, a necessary and sufficient condition for the power method to converge has been settled and now it is time to study the convergence rate of the power iteration. In (2.5) set $q_j = \dfrac{A^j q_0}{\|A^j q_0\|_2}$, $\beta_j = \dfrac{1}{\alpha_1' \lambda_1^j}$ and $x_j = \displaystyle\sum_{i=2}^{n} \frac{\alpha_i'}{\alpha_1'} \frac{\lambda_i^j}{\lambda_1^j} v_i$; then

$$\beta_j q_j = v_1 + x_j.$$

Let us now obtain a bound for the norm of x_j:

$$\|x_j\|_2 \leq \left|\frac{\alpha_2}{\alpha_1}\right| \left|\frac{\lambda_2}{\lambda_1}\right|^j \|v_2\|_2 + \left|\frac{\alpha_3}{\alpha_1}\right| \left|\frac{\lambda_3}{\lambda_1}\right|^j \|v_3\|_2 + \cdots + \left|\frac{\alpha_n}{\alpha_1}\right| \left|\frac{\lambda_n}{\lambda_1}\right|^j \|v_n\|_2 .$$

v_i is unitary by hypothesis, thus

$$\|x_j\|_2 \leq \left|\frac{\alpha_2}{\alpha_1}\right| \left|\frac{\lambda_2}{\lambda_1}\right|^j + \left|\frac{\alpha_3}{\alpha_1}\right| \left|\frac{\lambda_3}{\lambda_1}\right|^j + \cdots + \left|\frac{\alpha_n}{\alpha_1}\right| \left|\frac{\lambda_n}{\lambda_1}\right|^j =$$

$$\left|\frac{\lambda_2}{\lambda_1}\right|^j \left(\left|\frac{\alpha_2}{\alpha_1}\right| + \left|\frac{\alpha_3}{\alpha_1}\right| \left|\frac{\lambda_3}{\lambda_2}\right|^j + \cdots + \left|\frac{\alpha_n}{\alpha_1}\right| \left|\frac{\lambda_n}{\lambda_2}\right|^j \right).$$

Here,

$$\left|\frac{\alpha_2}{\alpha_1}\right| + \left|\frac{\alpha_3}{\alpha_1}\right| \left|\frac{\lambda_3}{\lambda_2}\right|^j + \cdots + \left|\frac{\alpha_n}{\alpha_1}\right| \left|\frac{\lambda_n}{\lambda_2}\right|^j$$

is a nonincreasing sequence, so it can be bounded by, for example,

$$K = \left|\frac{\alpha_2}{\alpha_1}\right| + \left|\frac{\alpha_3}{\alpha_1}\right| \left|\frac{\lambda_3}{\lambda_2}\right|^0 + \cdots + \left|\frac{\alpha_n}{\alpha_1}\right| \left|\frac{\lambda_n}{\lambda_2}\right|^0 = \left|\frac{\alpha_2}{\alpha_1}\right| + \left|\frac{\alpha_3}{\alpha_1}\right| + \cdots + \left|\frac{\alpha_n}{\alpha_1}\right| .$$

Which means that

$$\|x_j\|_2 \leq K \left|\frac{\lambda_2}{\lambda_1}\right|^j .$$

On the other hand, since $\beta_j q_j = v_1 + x_j$ and $\lim\limits_{j\to\infty} x_j = 0$

$$\lim_{j\to\infty} \beta_j q_j = v_1.$$

Due to $\|\cdot\|_2$ being a continuous function and $\|q_j\|_2 = \|v_1\|_2 = 1$,

$$\lim_{j\to\infty} |\beta_j| = 1.$$

What is more,

$$||\beta_j| - 1| \leq \left| \frac{1}{\left|\alpha_1' \lambda_1^j\right|} - 1 \right| = \left| \frac{\left\|A^j q_0\right\|_2}{\left|\alpha_1 \lambda_1^j\right|} - 1 \right| = \frac{1}{\left|\alpha_1 \lambda_1^j\right|} \left| \left\|A^j q_0\right\|_2 - \left|\alpha_1 \lambda_1^j\right| \right| =$$

$$\frac{1}{\left|\alpha_1\lambda_1^j\right|}\left|\left\|A^jq_0\right\|_2-\left\|\alpha_1\lambda_1^jv_1\right\|_2\right|\leq\frac{1}{\left|\alpha_1\lambda_1^j\right|}\left\|A^jq_0-\alpha_1\lambda_1^jv_1\right\|_2=\|x_j\|_2\,.$$

That is, for all $j\in\{0,1,2,...\}$

$$1-\|x_j\|_2\leq|\beta_j|\leq1+\|x_j\|_2\,,$$

and since $x_j\longrightarrow0$, there exists $J\in\mathbb{N}$ such that for all $j>J$, $1-\|x_j\|_2>0$. From now on we will assume that $j>J$. Let $z_j=\frac{\beta_j}{|\beta_j|}$ then $|z_j|=1$, and

$$z_jq_j=\frac{1}{|\beta_j|}(v_1+x_j)\leq\frac{1}{1-\|x_j\|_2}(v_1+x_j)=\frac{v_1}{1-\|x_j\|_2}+\frac{x_j}{1-\|x_j\|_2}=$$

$$\frac{1-\|x_j\|_2+\|x_j\|_2}{1-\|x_j\|_2}v_1+\frac{x_j}{1-\|x_j\|_2}=\left(1+\frac{\|x_j\|_2}{1-\|x_j\|_2}\right)v_1+\frac{1}{1-\|x_j\|_2}x_j.$$

Thus,

$$z_jq_j-v_1=\frac{\|x_j\|_2\,v_1+x_j}{1-\|x_j\|_2}$$

and since v_1 is unitary

$$\|z_jq_j-v_1\|_2\leq\frac{2\,\|x_j\|_2}{1-\|x_j\|_2}.$$

Now, $x_j\longrightarrow0$ so there exists $J'\in\mathbb{N}$ such that $1-\|x_j\|_2\geq\frac{1}{2}$ for all $j>J'$. Hence,

$$\|z_jq_j-v_1\|_2\leq4\,\|x_j\|_2\,.$$

Finally, for all $j>\max\{J,J',0\}$,

$$\|z_jq_j-v_1\|_2\leq4K\left|\frac{\lambda_2}{\lambda_1}\right|^j.$$

□

Conditions $|\lambda_1|>|\lambda_2|$ and $\alpha_1\neq0$ seem rather restrictive, but they actually are not. Almost every complex random matrix has a dominant eigenvalue. And although real matrices tend to have a dominant eigenvalue, if that eigenvalue is complex then the power iteration does not converge −real arithmetic can not approach complex numbers−.

The second restriction, $\alpha_1\neq0$, is discussed in §2.1.3.

2.1.2 Convergence criteria, optimal residuals and the Rayleigh quotient

Now that we know the requisites for the power method to converge, when do we stop to iterate? Computing the residual

$$r_k = Au_k - \mu_k u_k$$

is a way of measuring how far our current approximation is from the real eigenpair and although it does not give us a direct estimate of the error it does provide the following −the iteration subscripts are dropped for generality−:

Theorem 2.1.6. *Let $r = Au - \mu u$. Then there exists a matrix*

$$E = \frac{ru^*}{\|u\|_2^2} \tag{2.6}$$

such that

$$\frac{\|E\|_p}{\|A\|_p} = \frac{\|r\|_2}{\|A\|_p \|u\|_2}, \qquad p = 2,F \tag{2.7}$$

and

$$(A - E)u = \mu u \tag{2.8}$$

Proof. We just need to verify (2.7) and (2.8) using (2.6). First,

$$(A - E)u = \mu u \iff \left(A - \frac{ru^*}{\|u\|_2^2}\right) u = \mu u \iff Au - r\frac{u^*u}{\|u\|_2^2} = \mu u$$

$$Au - r\frac{\|u\|_2^2}{\|u\|_2^2} = \mu u \iff r = Au - \mu u.$$

And for (2.7):

$$\frac{\|E\|_p}{\|A\|_p} = \frac{\left\|\frac{ru^*}{\|u\|_2^2}\right\|_p}{\|A\|_p} = \frac{\|ru^*\|_p}{\|A\|_p \|u\|_2^2}. \tag{2.9}$$

At this point we need to introduce dual norms, which are defined for any norm as follows:

$$\|x\|' = \sup_{\|y\|=1} |y^*x| \,.$$

More specifically, the dual norm of the 2-norm is the 2-norm itself:

$$\|x\|_2' = \sup_{\|y\|_2=1} |y^*x| \leq \sup_{\|y\|_2=1} \|y\|_2 \|x\|_2 = \|x\|_2 \,.$$

(The Cauchy-Schwarz inequality has been used to separate the inner product y^*x.) And the reciprocal:

$$\|x\|_2' = \sup_{\|y\|_2=1} |y^*x| = \sup_{\|y\|_2=1} \left|\sum_{i=1}^n \overline{y_i} x_i\right| \geq \left|\sum_{i=1}^n \frac{x_i x_i}{\|x\|_2}\right| = \frac{1}{\|x\|_2}\left|\sum_{i=1}^n x_i x_i\right| = \frac{\|x\|_2^2}{\|x\|_2} = \|x\|_2 .$$

(A specific unitary vector $y = \frac{x}{\|x\|_2}$ has been chosen in the inequality.) Thus, we have proven that $\|x\|_2 = \|x\|_2' \ \forall x \in \mathbb{C}^n$. Now come back to (2.9). We have to study $\|ru^*\|_p$ where $p = 2, F$. So

$$\|ru^*\|_F^2 = \sum_{i=1}^n \sum_{j=1}^n |r_i \overline{u_j}|^2 = \sum_{i=1}^n \sum_{j=1}^n |r_i|^2 \, |\overline{u_j}|^2 = \sum_{i=1}^n |r_i|^2 \sum_{j=1}^n |u_j|^2 = \|r\|_2^2 \, \|u\|_2^2$$

$$\Longrightarrow \|ru^*\|_F = \|r\|_2 \, \|u\|_2 .$$

And using the dual norm:

$$\|ru^*\|_2 = \sup_{\|z\|=1} \|ru^*z\|_2 = \sup_{\|z\|=1} |u^*z| \, \|r\|_2 = \sup_{\|z\|=1} |z^*u| \, \|r\|_2 = \|r\|_2 \, \|u\|_2 .$$

Hence

$$\frac{\|E\|_p}{\|A\|_p} = \frac{\|r\|_2 \, \|u\|_2}{\|A\|_p \, \|u\|_2^2} = \frac{\|r\|_2}{\|A\|_p \, \|u\|_2}$$

□

The theorem states that if the residual is small, then the pair (μ, u) is an exact eigenpair of the nearby matrix $A - E$. It also suggests to stop when the 2-norm or Frobenus norm −it does not matter which as both of them are unitarily invariant and equivalent− of $A - E$ is relatively small comparing to the norm of A and it gives a computationally cheaper way to compute this quotient. The reason to stop the algorithm following this criteria has to do with backwards stability −see Definition 1.1.3. In our case $\tilde{x} = A - E$, $x = A$, $\tilde{f}$ will be the eigenpair the power method computes and f the exact eigenpair of A. Hence, if we stop iterating when the relative error of $A - E$ with regard to A is of the order of the machine epsilon, the power iteration algorithm will be backwards stable. We may stop when that relative error is less than the machine epsilon times a reasonable quantity.

In computing the residual $r = Au - \mu u$ we naturally take u to be the k-th iterate of the starting vector u_0. We will choose μ in such a way that minimizes the residual in some norm, 2-norm in our case, as we are mainly working with it.

Theorem 2.1.7. *Let $u \neq 0$ and for any μ set $r_\mu = Au - \mu u$. Then $\|r_\mu\|_2$ is minimized when*

$$\mu = \frac{u^*Au}{u^*u}$$

in which case $r_\mu \perp u$.

Proof. Let us assume $\|u\|_2 = 1$ −in our case the power iteration makes u to be unitary at every step−, let $[u\ U]$ be unitary and set

$$\begin{bmatrix} u^* \\ U^* \end{bmatrix} A \begin{bmatrix} u & U \end{bmatrix} = \begin{bmatrix} u^*Au & u^*AU \\ U^*Au & U^*AU \end{bmatrix} = \begin{bmatrix} \nu & h^* \\ g & B \end{bmatrix} \Longrightarrow \begin{bmatrix} u^* \\ U^* \end{bmatrix} A = \begin{bmatrix} \nu & h^* \\ g & B \end{bmatrix} \begin{bmatrix} u^* \\ U^* \end{bmatrix}.$$

Then,

$$\begin{bmatrix} u^* \\ U^* \end{bmatrix} Au - \mu \begin{bmatrix} u^* \\ U^* \end{bmatrix} u = \begin{bmatrix} \nu & h^* \\ g & B \end{bmatrix} \begin{bmatrix} u^* \\ U^* \end{bmatrix} u - \mu \begin{bmatrix} u^* \\ U^* \end{bmatrix} u \Longrightarrow$$

$$\begin{bmatrix} u^* \\ U^* \end{bmatrix} (Au - \mu u) = \begin{bmatrix} \nu & h^* \\ g & B \end{bmatrix} \begin{bmatrix} 1 \\ U^*u \end{bmatrix} - \mu \begin{bmatrix} 1 \\ U^*u \end{bmatrix}.$$

$U^*u = 0$ due to $\begin{bmatrix} u & U \end{bmatrix}$ being unitary, thus

$$\begin{bmatrix} u^* \\ U^* \end{bmatrix} r_\mu = \begin{bmatrix} \nu & h^* \\ g & B \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \mu \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \nu \\ g \end{bmatrix} - \begin{bmatrix} \mu \\ 0 \end{bmatrix} = \begin{bmatrix} \nu - \mu \\ g \end{bmatrix}.$$

Then

$$\left\| \begin{bmatrix} u^* \\ U^* \end{bmatrix} r_\mu \right\|_2^2 = \left\| \begin{matrix} \nu - \mu \\ g \end{matrix} \right\|_2^2,$$

thus, as $[u\ U]$ is unitary and $\|\cdot\|_2$ is unitarily invariant,

$$\|r_\mu\|_2^2 = |\nu - \mu|^2 + \|g\|^2,$$

which will be minimized when $\mu = \nu = u^*Au$.

Secondly, let us prove r_μ and u are orthogonal, that is, let us compute their inner product and see that it is zero. For this purpose, let us recall that the scalar product of r_μ and μ is $r_\mu^*\mu$:

$$r_\mu \cdot u = r_\mu^* u = (Au - \mu u)^* u = (Au)^* u - (\mu u)^* u = u^*A^*u - \overline{\mu} u^*u.$$

$[u\ U]$ is a unitary matrix, thus $\|u\|_2^2 = u^*u = 1$ and

$$u^*A^*u - \overline{\mu} u^*u = u^*A^*u - \overline{\mu} = u^*A^*u - \overline{(u^*Au)} = u^*A^*u - (u^*Au)^* =$$

$$u^*A^*u - u^*A^*u = 0.$$

□

The quantity u^*Au/u^*u is an example of a Rayleigh quotient and provides a highly accurate approximation to the corresponding eigenvalue of $A - E$. It is usually generalized this way:

Definition 2.1.2. Let u and v be vectors with $v^*u \neq 0$. Then the quantity

$$\frac{v^*Au}{v^*u}$$

is called a *Rayleigh quotient*.

2.1.3 Implementation details

We know both the circumstances that make the power method converge and the appropriate criteria to stop the algorithm. Therefore it is time to implement it and give some technical details.

Computation and normalization

If we were to ingenuously implement the power method we could compute, for every step k, $A^k u_0$ which runs in $\mathcal{O}(kn^3)$. Instead, calculating $u_{k+1} = Au_k$ runs in much cheaper $\mathcal{O}(n^2)$. If the matrices we are working with have structured elements and sparsity, the computational cost can be dropped even more.

The approximation of the eigenvector will be normalized in every step in order to avoid overflow and underflow.

Choice of starting vector

If we rewrite the expansion (2.1) as $u_0 = \gamma_1 x_1 + X_2 c_2$ where

$$X_2 = \begin{bmatrix} x_2 & \cdots & x_n \end{bmatrix} \text{ and } c_2 = \begin{bmatrix} \gamma_2 \\ \vdots \\ \gamma_n \end{bmatrix},$$

then the ratio $\|c_2\| / |\gamma_1|$ gives a measure of how far is the starting vector from the real eigenvector. It is important for u_0 to have the smallest posible quotient, which is minimized by the eigenvector. We may use an approximation of the eigenvector if we have it at hand. Sometimes the structure of the problem may dictate the choice of u_0. It is advisable to avoid patterned starting vectors: highly structured problems may have such eigenvectors and if that eigenvector is not the dominant one, γ_1 will be zero and so the method will not converge.

Otherwise, taking a random u_0 (usually a vector of random normal deviates) is good enough. For a random vector u of order n to have $\gamma_1 = 0$ means that x_1 and u are orthogonal. The orthogonal complement of the vector u is a space of dimension $n-1$ and thus the probability of a random vector of order n to be in a subspace of dimension $n-1$ is 0. Hence, the probability of a vector of random normal deviates to be perpendicular to x_1 is zero.

Shift and spectral enhancement

If A is a nondefective matrix and the eigenvalues of A satisfy $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$, then the power method almost always converges to the biggest eigenvalue at a rate of $|\lambda_2| / |\lambda_1|$; but if this quotient is near one convergence

will be slow. Sometimes this can be resolved by working with the matrix $A - \kappa I$, whose eigenvalues are $\lambda_1 - \kappa, ..., \lambda_n - \kappa$. For example, let A have eigenvalues 1, 0 and -0.99. In this case the ratio will be 0.99, but if we take $\kappa = -0.5$, the eigenvalues will shift to 1.5, 0.5 and -0.49 and thus the ratio will improve to 1/3.

The shifting of a matrix is an example of spectral enhancement. The next section of this chapter discusses a much more powerful one: the shift-and-invert operation.

The shifted power method

Input : matrix A, shift κ, convergence criterion ϵ, nonzero starting vector x and maximum number of iterations n.
Output: approximate eigenpair (μ, x)

```
1  PowerMethod (A, κ, ϵ, x, n)
2      Anorm ← 1/‖A‖_F
3      xnorm ← 1/‖x‖_2
4      x ← x · xnorm
5      for i ∈ {1, ..., n} do
6          y ← Ax
7          μ ← x*y
8          r ← y − μx
9          x ← y − κx
10         xnorm ← 1/‖x‖_2
11         x ← x · xnorm
12         if ‖r‖_2 · Anorm ≤ ϵ then
13             return (μ, x)
14         end
15     end
16     error
17 end
```

Algorithm 2: The shifted power method

The MATLAB implementation of Algorithm 2 is in A.1.1. A couple of remarks to the algorithm above:

- Intermediate parameters $y = Ax$, $xnorm$ and $Anorm$ are used to avoid some computations. Furthermore, floating-point divisions are expensive so both $xnorm$ and $Anorm$ are the inverse of the norm of the matrices x and A.

- The code returns an error after a maximum number of iterations is reached so that the algorithm stops if it does not converge.

- The convergence test given in Theorem 2.1.6 is based on the eigenpair (μ, x) but Algorithm 2 returns $(\mu, \widetilde{x})$ were $\widetilde{x}$ is the next iteration of x. Strictly speaking we can't assure $\widetilde{x}$ is the eigenvector of a nearby matrix $A - E$, but since it is the result of an extra iteration it will almost always be a better approximation of the real eigenvector.

2.2 The inverse power method

The main flaw of the power iteration, the fact that it only computes the dominant eigenvalue, can be overcome by finding a proper spectral enhancement. The traditional one for doing so is called the shift-and-invert enhancement and applying it to the power iteration results in the inverse power method.

2.2.1 Shift-and-invert enhancement and the Rayleigh quotient method

Let A have eigenvalues $\lambda_1, ..., \lambda_n$ and let us assume λ_1 is simple. If we have an approximation κ of the eigenvalue λ_1, then the eigenvalues of the matrix $(A - \kappa I)^{-1}$ are

$$\mu_1 = \frac{1}{\lambda_1 - \kappa}, \ ..., \ \mu_n = \frac{1}{\lambda_n - \kappa}$$

from where we can conclude that $\mu_1 \to \infty$ as $\kappa \to \lambda_1$. Therefore we can transform any eigenvalue into a dominant one and the dominance can be as large as we want. As a result, the inverse power method is nothing but the power method applied to the enhanced matrix $A - \kappa I$.

Let us see why the Rayleigh quotient defined in Definition 2.1.2 can be used to provide a better approximation to the eigenvalue than a random shift κ. Let q_j be the unitary approximation to the dominant eigenvector of A computed in the jth iteration of the power method. If q_j were to be an exact eigenvector of A then

$$Aq_j = \lambda q_j \Longrightarrow q_j^* A q_j = \lambda q_j^* q_j = \lambda,$$

and $q_j^* A q_j$ would be an exact eigenvalue. Thus if q_j is an approximate eigenvector of A then, due to the continuity of the matrix multiplication, $q_j^* A q_j$ would be an approximate eigenvalue of A. Consequently, replacing κ with μ at every iteration will improve the convergence. This method is referred to as *the Rayleigh quotient method* and provided A is hermitian, using it can enhance the convergence of the power iteration to be, at least, quadratic.

2.2.2 Implementation details

Due to the nature of the shift-and-invert enhancement, the residual can be computed almost for free. Let

$$y = (A - \kappa I)^{-1} x,\ \hat{x} = \frac{y}{\|y\|_2} \text{ and } w = \frac{x}{\|y\|_2} = (A - \kappa I)\hat{x}.$$

Then, by taking

$$\rho = \hat{x}^* (A - \kappa I) \hat{x} = \hat{x}^* w$$

the residual can be written as

$$r = (A - \kappa I)\hat{x} - \lambda\hat{x} = (A - \kappa I)\hat{x} - \hat{x}^* (A - \kappa I)\hat{x}\hat{x} = w - \rho\hat{x}.$$

The Rayleigh quotient shift con be computed as $\mu = \kappa + \rho$ because

$$\mu = \kappa + \rho = \kappa + \hat{x}^*(A - \kappa I)\hat{x} = \kappa + \hat{x}^* A\hat{x} - \kappa = \hat{x}^* A\hat{x}.$$

Hence the algorithm can be presented as:

Input : matrix A, shift κ, convergence criterion ϵ, nonzero starting vector x and maximum number of iterations n.
Output: approximate eigenpair (μ, x)

```
1  InversePowerMethod (A, κ, ε, x, n)
2      Anorm ← 1/‖A‖_F
3      for i ← 1 to n do
4          y ← (A − κI)^{-1} x (solve the system (A − κI) y = x)
5          ynorm ← 1/‖y‖_2
6          x̂ ← y · ynorm
7          w ← x · ynorm
8          ρ ← x̂* w
9          μ ← κ + ρ
10         r ← w − ρx̂
11         x ← x̂
12         κ ← μ
13         if ‖r‖_2 · Anorm ≤ ε then
14             return (μ, x)
15         end
16     end
17     error
18 end
```

Algorithm 3: The inverse power method

The major work on this algorithm consists on solving the system on row 3 and some care must be taken so as not to increase too much the running time. It is usually solved using LU decomposition with Gaussian

elimination and partial pivoting, which runs in $\mathcal{O}(n^3)$. Again, depending on the matrices we are working with solving the system could be cheaper. They can be structured, for example.

The MATLAB code is in A.1.2.

Chapter 3

First steps towards an efficient QR algorithm

3.1 Introduction

Let us start by saying that

unless otherwise indicated, in this chapter A will be a complex matrix of order n.

The QR algorithm reduces a matrix A to triangular Schur form by unitary similarity transformations. Most discussions of it start by showing the following naive approach:

Input : matrix A
Output: triangular matrix T

```
1  naiveqr (A)
2      A_0 ← A
3      k ← 0
4      while not convergence do
5          Choose shift κ_k
6          Factor A_k − κ_k I = Q_k R_k, where Q_k is unitary and R_k is upper
             triangular (QR factorization)
7          A_{k+1} = R_k Q_k + κ_k I
8          k ← k + 1
9      end
10     return A_k
11 end
```

Algorithm 4: First approach to the QR algorithm

This routine is simplistic and far removed from the versions of QR used in practice. However, it usually presented for two reasons: the sake of historical accuracy −Kublanovskaya's version works this way and so does

Rutishauser's precursory LR algorithm– and its usefulness to relate the shifted version of QR with the shifted inverse power method and the unshifted version with the power method. Before anything else, let us prove the following results:

Lemma 3.1.1. *For every step k of the QR method, matrices A_{k+1} and A_k are unitarily similar.*

Proof. In line 6 of Algorithm (4) a QR factorization of A_k is computed: $A_k - \kappa_k I = Q_k R_k$, with Q_k unitary and R_k upper triangular. Then, $R_k = Q_k^*(A_k - \kappa_k I)$ and plugging this on line 7 of Algorithm 4:

$$A_{k+1} = R_k Q_k - \kappa_k I = Q_k^*(A_k - \kappa_k I)Q_k + \kappa_k I = Q_k^* A_k Q_k - \kappa_k I + \kappa_k I = Q_k^* A_k Q_k$$

□

Noticing that we have defined $A_0 = A$ on line 2 of algorithm (4) leads to this corollary:

Corollary 3.1.2. *For every step k, the matrix A_k computed by the QR algorithm is unitarily similar to the initial matrix A.*

Lemma 3.1.3. *Hermitian matrices remain unchanged under a QR step, that is, if A is hermitian then so is A_k for every k.*

Proof. We know from Lemma 3.1.1 that $A_{k+1} = Q_k^* A_k Q_k$, from which using Corollary 3.1.2 we infer

$$A_k = Q_k^* \cdots Q_0^* A Q_0 \cdots Q_k.$$

Then

$$A_k^* = (Q_k^* \cdots Q_0^* A Q_0 \cdots Q_k)^* = Q_k^* \cdots Q_0^* A^* Q_0 \cdots Q_k$$

and if A is hermitian, $A^* = A$, so

$$A_k^* = Q_k^* \cdots Q_0^* A Q_0 \cdots Q_k = A_k.$$

□

3.2 The shifted QR algorithm and the inverse power method

The above-mentioned connection between the shifted QR and the inverse power method is in reality an accesory to prove that this method approaches a deflation to the Schur form, which is what makes this algorithm worth considering for the computation of eigenvalues and eigenvectors. To see the

intuition behind this, let (λ, q) be a left eigenpair of the matrix A and let $Q = \begin{bmatrix} Q_* & q \end{bmatrix}$ be unitary. Then

$$Q^*AQ = \begin{bmatrix} Q_*^*AQ_* & Q_*^*Aq \\ q^*AQ_* & q^*Aq \end{bmatrix} = \begin{bmatrix} \hat{B} & \hat{h} \\ \hat{g}^* & \hat{\mu} \end{bmatrix} \tag{3.1}$$

where $\hat{g}^* = q^*AQ_* = \lambda q^*Q_* = 0$ and $\hat{\mu} = q^*Aq = \lambda q^*q = \lambda$ due to Q being unitary. Thus,

$$Q^*AQ = \begin{bmatrix} \hat{B} & \hat{h} \\ \hat{g}^* & \hat{\mu} \end{bmatrix} = \begin{bmatrix} \hat{B} & \hat{h} \\ 0 & \lambda \end{bmatrix}.$$

Our search has been reduced to the matrix $\hat{B}$. (In these cases it is said that *the problem has been deflated.*)

In practice, though, this procedure does not make much sense, since eigenvalues and eigenvectors of A, $\hat{B}$, etc. are required, which are exactly the ones this project looks for. Nevertheless, by choosing q to be an approximate of an eigenvector, $\hat{g}^*$ in (3.1) will be small as $\|\hat{g}\|_2$ is the norm of the residual $q^*A - \hat{\mu}q^*$. Let us prove this. Let $r_{\hat{\mu}} = q^*A - \hat{\mu}q^*$. Following the proof of Theorem 2.1.7 −where $(\hat{\mu}, q^*)$ is a left eigenpair instead of a right one−, $\|r_{\hat{\mu}}\|^2 = |q^*Aq - \hat{\mu}|^2 + \|\hat{g}\|^2$. Thus, if we choose $\hat{\mu} = q^*Aq$ then $\|r_{\hat{\mu}}\| = \|\hat{g}\|$.

The following theorem specifies which q does the algorithm implicitly choose by linking it to the inverse power method and as a result making the procedure worth considering for the computation of both eigenvalues and eigenvectors.

Theorem 3.2.1. *The QR method chooses q to be a vector produced by the inverse power method with shift κ applied to the vector $\boldsymbol{e}_n$, where $\boldsymbol{e}_n$ is the nth canonical vector. What is more, if A is partitioned in the form*

$$A = \begin{bmatrix} B & h \\ g^* & \mu \end{bmatrix} \tag{3.2}$$

it suggests the starting shift $\kappa = \mu$.

Proof. Rewrite the QR factorization of $A - \kappa I$:

$$A - \kappa I = \begin{bmatrix} Q_* & q \end{bmatrix} \begin{bmatrix} R_* \\ r^* \end{bmatrix} \Longrightarrow \begin{bmatrix} Q_*^* \\ q^* \end{bmatrix} (A - \kappa I) = \begin{bmatrix} R_* \\ r^* \end{bmatrix}.$$

R is upper triangular so, $r^* = r_{nn}\mathbf{e}_n^*$. Therefore

$$q^*(A - \kappa I) = r_{nn}\mathbf{e}_n^* \Longrightarrow q^* = r_{nn}\mathbf{e}_n^*(A - \kappa I)^{-1},$$

which means that the last column of the Q matrix calculated by the QR algorithm is the result of the inverse power method with shift κ applied to the vector $\mathbf{e}_n$.

Now, partition A in the form (3.2). Then, following Theorem 2.1.7, in which we proved that the residual $r_\nu = Au - \nu u$ is minimized when $\nu = \frac{u^*Au}{u^*u}$ for some $u \neq 0$. The starting vector $\mathbf{e}_n$ implies that $\nu = \mathbf{e}_n^*A\mathbf{e}_n = \mu$. □

3.2.1 Convergence of the shifted QR algorithm

We now know that the shifted QR method approaches the deflation to a Schur form of a given complex matrix A. Furthermore, the reason for this is that the shifted QR routine can be related to the inverse power method, which was explained in §2.2. However, at which rate does the problem converge?

Theorem 3.2.2. *The convergence of the shifted QR algorithm is* locally quadratic.

Proof. If A is partitioned in the form (3.2), the QR algorithm will converge to a deflated matrix if $\hat{g}$ in (3.1) converges to zero. Lets obtain a bound of the norm of $\hat{g}$ using $\|g\|_2$. Partition $A_k - \kappa_k I = QR$ in the form

$$A_k - \kappa_k I = \begin{bmatrix} B_k - \kappa_k I & h_k \\ g_k^* & \mu_k - \kappa_k \end{bmatrix} = \begin{bmatrix} P & f \\ e^* & \pi \end{bmatrix} \begin{bmatrix} S & r \\ 0 & \rho \end{bmatrix} = QR$$

and $A_{k+1} - \kappa_k I = RQ$ in

$$A_{k+1} - \kappa_k I = \begin{bmatrix} B_{k+1} - \kappa_k I & h_{k+1} \\ g_{k+1}^* & \mu_{k+1} - \kappa_k \end{bmatrix} = \begin{bmatrix} S & r \\ 0 & \rho \end{bmatrix} \begin{bmatrix} P & f \\ e^* & \pi \end{bmatrix} = RQ.$$

Let us now drop the subscripts to simplify the notation:

$$A - \kappa I = \begin{bmatrix} B - \kappa I & h \\ g^* & \mu - \kappa \end{bmatrix} = \begin{bmatrix} P & f \\ e^* & \pi \end{bmatrix} \begin{bmatrix} S & r \\ 0 & \rho \end{bmatrix} = QR \tag{3.3}$$

and

$$\hat{A} - \kappa I = \begin{bmatrix} \hat{B} - \kappa I & \hat{h} \\ \hat{g}^* & \hat{\mu} - \kappa \end{bmatrix} = \begin{bmatrix} S & r \\ 0 & \rho \end{bmatrix} \begin{bmatrix} P & f \\ e^* & \pi \end{bmatrix} = RQ. \tag{3.4}$$

Notice in (3.4) that

$$\hat{g}^* = \rho e^* \Longrightarrow \|\hat{g}\|_2 \leq |\rho| \, \|e\|_2 \tag{3.5}$$

so further information on the absolut value of ρ and the norm of e is needed to bound that of $\hat{g}$. First, since Q is unitary, the norms of its rows and columns must be one, i.e., $\|e\|_2^2 + \pi^2 = 1 = \|f\|_2^2 + \pi^2$, and so

$$\|e\|_2 = \|f\|_2 \,. \tag{3.6}$$

Now, from (3.3) and assuming S is nonsingular, we have

$$g^* = e^* S \Longrightarrow g^* S^{-1} = e^* \Longrightarrow e = \left(g^* S^{-1}\right)^* \Longrightarrow$$

$$\|e\|_2 = \left\|\left(g^* S^{-1}\right)^*\right\|_2 = \left\|(S^{-1})^* g\right\|_2 \leq \left\|S^{-1}\right\|_2 \|g\|_2 = \sigma \, \|g\|_2 \,.$$

(Remember that $\|\cdot\|_2$ is consistent and that $\|A\|_2 = \|A^*\|_2$ for every $A \in \mathbb{C}^{n\times n}$.) Briefly,

$$\|e\|_2 \leq \sigma \, \|g\|_2 \,. \tag{3.7}$$

On the other hand, to obtain a bound on ρ, rewrite (3.3) using the fact that Q is unitary

$$\begin{bmatrix} P^* & e \\ f^* & \overline{\pi} \end{bmatrix} \begin{bmatrix} B - \kappa I & h \\ g^* & \mu - \kappa \end{bmatrix} = \begin{bmatrix} S & r \\ 0 & \rho \end{bmatrix}$$

and find that $\rho = f^*h + \overline{\pi}(\mu - \kappa)$. Using (3.6), (3.7), and that $|\overline{\pi}| \leq 1$ due to $|\pi| \leq 1$; we conclude

$$\|\rho\|_2 = \|f^*h + \overline{\pi}(\mu - \kappa)\|_2 \leq \|f\|_2 \|h\|_2 + |\overline{\pi}| \, |\mu - \kappa| \leq \|e\|_2 \|h\|_2 + |\mu - \kappa| \Longrightarrow$$

$$\|\rho\|_2 \leq \sigma \|g\|_2 \|h\|_2 + |\mu - \kappa| \,. \tag{3.8}$$

Finally, substituting (3.7) and (3.8) in (3.5)

$$\|\hat{g}\| \leq \sigma \|g\|_2 \left(\sigma \|g\|_2 \|h\|_2 + |\mu - \kappa|\right) = \sigma^2 \|g\|_2^2 \|h\|_2 + \sigma \|g\|_2 |\mu - \kappa| \Longrightarrow$$

$$\|\hat{g}\| \leq \sigma^2 \|g\|_2^2 \|h\|_2 + \sigma \|g\|_2 |\mu - \kappa| \,. \tag{3.9}$$

Restoring the iteration subscripts, the result is

$$\|g_{k+1}\|_2 \leq \sigma_k^2 \|g_k\|_2^2 \|h_k\|_2 + \sigma_k \|g_k\|_2 |\mu_k - \kappa_k|$$

where some addends can be simplified. First, remembering that all the iterations of QR are unitarily similar to the initial matrix A −as was proved in Corollary 3.1.2− and looking on (3.2):

$$\|h_k\|_2^2 \leq \|h_k\|_2^2 + |\mu|^2 = \|A_k \mathbf{e}_n\|_2^2 \leq \|A_k\|_2^2 \|\mathbf{e}_n\|_2^2 = \|A_k\|_2^2 = \|A\|_2^2 \,.$$

Then $\exists \ \eta > 0$ where $\|h_k\|_2 \leq \eta$ and

$$\|g_{k+1}\|_2 \leq \sigma_k^2 \eta \|g_k\|_2^2 + \sigma_k \|g_k\|_2 |\mu_k - \kappa_k| \,.$$

This expressions suggests, apart from the one seen in Theorem 3.2.1, another reason to choose $\mu_k = \kappa_k$ as the appropriate shift on every iteration: it gives an upper bound for the norm of g_{k+1} on terms of the square of $\|g_k\|_2$. Therefore,

$$\|g_{k+1}\|_2 \leq \sigma_k^2 \eta \|g_k\|^2 \,.$$

Finally, $\sigma_k = \left\|S_k^{-1}\right\|_2$ can be proved to be smaller than a constant σ for small enough values of g_k. (It is not an easy task to prove this property rigorously. Let us just say that it is based on the important fact that the factors Q and R of the QR factorization of a matrix A depend continuously on the elements of A.) So, letting $\sigma_k \leq \sigma$ for all k leads to

$$\|g_{k+1}\| \leq \sigma^2 \eta \|g_k\|^2 \,. \tag{3.10}$$

□

Remark 3.2.1. The convergence of the shifted QR method is said to be locally quadratic and not simply quadratic because the condition $\left\|S_k^{-1}\right\|_2 \leq \sigma$ only holds for small enough values of g_k. Therefore the convergence analysis is not global. What is more, it does not exist a theoretical result ensuring the global convergence of the QR algorithm for any given matrix A. Experience has shown that the method at first consumes some iterations in which the convergence is very slow or it seems to not converge, and later starts to converge very fast. Subsequent eigenvalues need fewer and fewer iterations to converge. This has to do with the relation between the QR algorithm and the power method, which is the subject of the next section.

Remark 3.2.2. We mentioned that it does not exist any result ensuring the global convergence of the QR algorithm. [13] and [14] treat this matter meticulously.

Example 3.2.1. (i) Once quadratic convergence is reached, the confluence is very fast. For example substitute $\sigma^2\eta = 1$ and $\|g_0\| = 10^{-1}$ in (3.10):

$$\|g_1\|_2 \leq 10^{-2}$$

$$\|g_2\|_2 \leq 10^{-4}$$

$$\|g_3\|_2 \leq 10^{-8}$$

$$\|g_4\|_2 \leq 10^{-16}$$

Four iterations are enough to reduce the error by a factor corresponding to the double-precision rounding unit.

(ii) If A_0 is Hermitian then so is every A_k −check Theorem 3.1.3− and thus, in (3.3), $h_k = g_k$ which means $\eta = \|g_k\|_2$ and our bound has been improved to

$$\|g_{k+1}\|_2 \leq \sigma^2 \|g_k\|_2^3 .$$

This type of convergence is even faster. Take $\sigma^2 = 1$ and $\|g_0\| = 10^{-1}$:

$$\|g_1\|_2 \leq 10^{-3}$$

$$\|g_2\|_2 \leq 10^{-9}$$

$$\|g_3\|_2 \leq 10^{-27}$$

In three iterations the error has been reduced way below the double-precision rounding unit.

3.3 The unshifted QR algorithm and the power method

The connection of the QR algorithm with the inverse power method explains why the convergence to a particular eigenvalue is quick. But the method, however more slowly, produces approximations to the other eigenvalues. This is due to the relation between the QR routine and the power method.

As it has been shown in Lemma 3.1.1, each step of the QR routine consists of a unitary similarity transformation $A_{k+1} = Q_k^* A Q_k$. At the end of the process we will need a single transformation that will move $A = A_0$ to A_{k+1}. This can be achieved by accumulating the transformations, in other words, setting

$$\breve{Q}_k = Q_0 \cdots Q_k \tag{3.11}$$

then

$$\breve{Q}_k^* A_0 \breve{Q}_k = A_{k+1}.$$

And this matrix $\breve{Q}_k$ has more applications:

Theorem 3.3.1. *Let $Q_0, ..., Q_k$ and $R_0, ..., R_k$ be the orthogonal and triangular matrices generated by the QR algorithm with shifts $\kappa_0, ..., \kappa_k$ starting with the matrix A. Let*

$$\breve{Q}_k = Q_0 \cdots Q_k \text{ and } \breve{R}_k = R_k \cdots R_0.$$

Then

$$\breve{Q}_k \breve{R}_k = (A - \kappa_k I) \cdots (A - \kappa_0 I). \tag{3.12}$$

Proof. We will proceed by induction over k.

- k=0.
  The QR routine characterizes A_1 as $A_1 = R_0 Q_0 + \kappa_0 I$ where Q_0 is unitary. So

  $$R_0 = (A_1 - \kappa_0 I) Q_0^* = Q_0^* (A - \kappa_0 I) Q_0 Q_0^* = Q_0^* (A - \kappa_0 I).$$

  That is,

  $$\breve{Q}_0 \breve{R}_0 = Q_0 R_0 = (A - \kappa_0 I).$$

- Induction hypothesis.
  Assume $\breve{Q}_{k-1} \breve{R}_{k-1} = (A - \kappa_{k-1} I) \cdots (A - \kappa_0 I)$ it is true.

- The general $-k$th$-$ case.
  By the QR routine characterization A_{k+1} is $A_{k+1} = R_k Q_k - \kappa_k I$, thus

  $$R_k = (A_{k+1} - \kappa_k I) Q_k^* = \breve{Q}_k^* (A - \kappa_k I) \breve{Q}_k Q_k^*.$$

Notice (3.11), then

$$R_k = (A - \kappa_k I)Q_k^* = \breve{Q}_k^*(A - \kappa_k I)\breve{Q}_{k-1}.$$

Postmultiply $\breve{R}_{k-1}$:

$$\breve{R}_k = R_k\breve{R}_{k-1} = \breve{Q}_k^*(A - \kappa_k I)\breve{Q}_{k-1}\breve{R}_{k-1}.$$

By the induction hypothesis:

$$\breve{R}_k = \breve{Q}_k^*(A - \kappa_k I)(A - \kappa_{k-1} I)\cdots(A - \kappa_0 I).$$

That is,

$$\breve{Q}_k\breve{R}_k = (A - \kappa_k I)\cdots(A - \kappa_0 I).$$

□

It is from this result that we derive the relationship between the unshifted QR algorithm and the power method. The unshifted QR algorithm uses shifts $\kappa_k = 0$, so by (3.12)

$$\breve{Q}_k\breve{R}_k = A^k.$$

Remembering that $\breve{R}_k$ is upper triangular, $\breve{R}_k\mathbf{e}_1 = \breve{r}_{11}^{(k)}\mathbf{e}_1$. Therefore

$$\breve{Q}_k\breve{R}_k\mathbf{e}_1 = \breve{r}_{11}^{(k)}\breve{Q}_k\mathbf{e}_1 = A^k\mathbf{e}_1,$$

which means that the first column of $\breve{Q}_k$ is the normalized result of applying k iterations of the power method to $\mathbf{e}_1$. Here, assuming that the conditions for the convergence of the power method are met −check Theorem 2.1.5−, the vectors $\breve{q}_1^{(k)}$ approach the dominant eigenvector of A. To see this, let $\breve{Q}_k = \begin{bmatrix}\breve{q}_1^{(k)} & \breve{Q}_*^{(k)}\end{bmatrix}$. Then A_k −the kth iterate of A through the unshifted QR algorithm− is

$$A_k = \breve{Q}_k^* A\breve{Q}_k = \begin{bmatrix}(\breve{q}_1^{(k)})^* \\ (\breve{Q}_*^{(k)})^*\end{bmatrix} A \begin{bmatrix}\breve{q}_1^{(k)} & \breve{Q}_*^{(k)}\end{bmatrix} = \begin{bmatrix}(\breve{q}_1^{(k)})^* A\breve{q}_1^{(k)} & (\breve{q}_1^{(k)})^* A\breve{Q}_*^{(k)} \\ (\breve{Q}_*^{(k)})^* A\breve{q}_1^{(k)} & (\breve{Q}_*^{(k)})^* A\breve{Q}_*^{(k)}\end{bmatrix},$$

that is,

$$A_k = \begin{bmatrix}(\breve{q}_1^{(k)})^* A\breve{q}_1^{(k)} & (\breve{q}_1^{(k)})^* A\breve{Q}_*^{(k)} \\ (\breve{Q}_*^{(k)})^* A\breve{q}_1^{(k)} & (\breve{Q}_*^{(k)})^* A\breve{Q}_*^{(k)}\end{bmatrix} = \begin{bmatrix}\mu_k & h_k^* \\ g_k & B_k\end{bmatrix}.$$

This is the same process as that of the beginning of §3.2, but with right eigenvectors and eigenvalues instead of left ones. We know that $\breve{q}_1^{(k)}$ approximates the dominant eigenvector of A, then $g_k \to 0$ and the Rayleigh quotient $(\breve{q}_1^{(k)})^* A\breve{q}_1^{(k)} = \mu_k \to \lambda_1$ where λ_1 is the dominant eigenvalue of A.

3.3.1 Convergence of the unshifted QR algorithm

As is often fashion in mathematics, the convergence of the unshifted QR routine can be proved as a special case of a more general result: under the right circumstances the unshifted QR algorithm actually triangularizes A.

Remark 3.3.1. From now on some special notation will be used sometimes. Let A be a matrix of order n, then $|A|$ does not refer to its determinant but to a matrix where its elements are the elements of A in absolute value. That is,

$$|A| = \begin{bmatrix} |a_{11}| & \cdots & |a_{1n}| \\ \vdots & \ddots & \vdots \\ |a_{n1}| & \cdots & |a_{nn}| \end{bmatrix}.$$

Theorem 3.3.2. *Let* $X^{-1}AX = \Lambda = \operatorname{diag}(\lambda_1, ..., \lambda_n)$ *where*

$$|\lambda_1| > \cdots > |\lambda_n|. \tag{3.13}$$

Suppose that X^{-1} *has an LU factorization* $X^{-1} = LU$*, where* L *is unit lower triangular, and let* $X = QR$ *be the QR factorization of* X*. If* A^k *has the QR factorization* $A^k = \breve{Q}_k \breve{R}_k$*, then there are diagonal matrices* D_k *with* $|D_k| = I$ *such that* $\breve{Q}_k D_k \longrightarrow Q$*.*

Proof. By hypothesis,

$$X^{-1}AX = \Lambda \Longrightarrow A^k = X\Lambda^k X^{-1} = QR\Lambda^k LU = QR(\Lambda^k L\Lambda^{-k})(\Lambda^k U).$$

Due to L being lower triangular, for all $i > j$,

$$(\Lambda^k L\Lambda^{-k})_{i,j} = \ell_{ij}\left(\frac{\lambda_i}{\lambda_j}\right)^k.$$

Thus by (3.13), $\Lambda^k L\Lambda^{-k} \longrightarrow I$. Now, write $(\Lambda^k L\Lambda^{-k}) = I + E_k$ where $E_k \longrightarrow 0$. Hence,

$$A^k = QR(\Lambda^k L\Lambda^{-k})(\Lambda^k U) = QR(I + E_k)(\Lambda^k U) = Q(I + RE_kR^{-1})(R\Lambda^k U).$$

Letting $\hat{Q}_k\hat{R}_k$ be the QR decomposition of $I + RE_kR^{-1}$,

$$A^k = (Q\hat{Q}_k)(\hat{R}_k R\Lambda^k U);$$

and since $I + RE_kR^{-1} \longrightarrow I$, by the continuity of the elements of the QR factorization, both $\hat{R}_k \longrightarrow I$ and $\hat{Q}_k \longrightarrow I$. Define $\delta_1, ..., \delta_n$ to be the diagonal elements of $\hat{R}_k R\Lambda^k U$ and

$$D_k = \operatorname{diag}\left(\frac{\overline{\delta_1}}{|\delta_1|}, ..., \frac{\overline{\delta_n}}{|\delta_n|}\right).$$

As a consequence, the triangular factor (the one on the right) in the decomposition

$$A^k = (Q\hat{Q}_k D_k^{-1})(D_k \hat{R}_k R\Lambda^k U)$$

has positive diagonal elements, and by the uniqueness of the QR decomposition $\breve{Q}_k = Q\hat{Q}_k D_k^{-1}$. Finally,

$$\breve{Q}_k D_k = Q\hat{Q}_k \longrightarrow Q$$

□

Corollary 3.3.3. *Under the conditions of Theorem 3.3.2 the unshifted QR algorithm produces a sequence of matrices converging to a triangular matrix.*

Proof. The columns of the Q-factor of X are the Schur vectors of A corresponding to the eigenvalues in the order (3.13). Therefore, with its columns suitably scaled the matrix $\breve{Q}_k$ converges to the orthogonal part of the Schur decomposition of A. Then the QR iterates $A_k = \breve{Q}_k^* A \breve{Q}_k$ must converge to the triangular factor of the Schur decomposition of A. □

3.4 Making the QR iteration practical: the Hessenberg form

Let us look back to the implementation written in Algorithm 4 and calculate its time complexity. The most expensive computation inside the iteration is the QR decomposition in line 4, which runs in $\mathcal{O}(n^3)$. Since at least one iteration per eigenvalue is needed, the operation count of finding all the eigenvalues is, at the very least, $\mathcal{O}(n^4)$.

The solution to this problem is to somehow compute the QR decomposition in a cheaper way. In §4.4 and §5.4 we will see that the QR decomposition of *upper Hessenberg matrices* can be done in $\mathcal{O}(n^2)$. These matrices have the following form (for dimension $n = 5$):

$$\begin{bmatrix} \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & 0 & 0 & \mathrm{X} & \mathrm{X} \end{bmatrix}.$$

(In this representation, called *Wilkinson diagram*, the zeros stand for a zero element and the crosses for elements that may be nonzero.) Computing the QR decomposition in $\mathcal{O}(n^2)$ brings down the total operation count of the QR method to $\mathcal{O}(n^3)$. The important fact is that any matrix can be reduced to upper Hessenberg form. Here is the way to do it.

3.4.1 Householder transformations

The mentioned reduction is accomplished by a class of transformations called *Householder transformations* or *elementary reflectors*. Such a transformation is a matrix of the form

$$H = I - uu^* \text{ where } \|u\|_2 = \sqrt{2}$$

or

$$H = I - 2uu^* \text{ with } \|u\|_2 = 1.$$

They can be cheaply applied to any matrix A:

$$HA = (I - uu^*)A = A - u(u^*A),$$

$$AH = A(I - uu^*) = A - (Au)u^*.$$

Lemma 3.4.1. *A Householder transformation is Hermitian and unitary.*

Proof. Let $H = I - uu^*$ be an elementary reflector. Then, on the one hand,

$$H^* = (I - uu^*)^* = I^* - (uu^*)^* = I - uu^* = H,$$

which makes H Hermitian. On the other hand,

$$HH^* = H^*H = H^2 = (I - uu^*)(I - uu^*) = I - uu^* - uu^* + uu^*uu^* =$$

$$I - 2uu^* + \|u\|_2^2 uu^* = I - 2uu^* + 2uu^* = I,$$

so H is unitary. □

Householder transformations will be used in this work to introduce zeros in any vector:

Theorem 3.4.2. *Let $a \neq 0$ be a vector and let*

$$u = \frac{\rho\frac{a}{\|a\|_2} + \boldsymbol{e}_1}{\sqrt{1 + \rho\frac{a_1}{\|a\|_2}}}$$

where $|\rho| = 1$, a_1 is the first element of the vector a, and $\rho a_1 > 0$. Then

$$Ha = (I - uu^*)a = -\nu\boldsymbol{e}_1,$$

where $\nu \in \mathbb{C}$.

Proof. Let $z = \rho\frac{a}{\|a\|_2}$, then $\|z\|_2 = |\rho|\frac{\|a\|_2}{\|a\|_2} = 1$. Set $z_1 = \rho\frac{a_1}{\|a\|_2}$ $-z_1$ is the first element of the vector z. Thus we have to prove that if

$$u = \frac{z + \mathbf{e}_1}{\sqrt{1 + z_1}}, \tag{3.14}$$

then

$$H\left(\frac{\|a\|_2}{\rho}z\right) = (I - uu^*)\left(\frac{\|a\|_2}{\rho}z\right) = -\nu\mathbf{e}_1 \text{ for some } \nu \in \mathbb{C}.$$

Which can be written as

$$\frac{\|a\|_2}{\rho}Hz = \frac{\|a\|_2}{\rho}(I - uu^*)z = -\nu\mathbf{e}_1.$$

Taking (3.14) we deduce that

$$u^* = \frac{z^* + \mathbf{e}_1^*}{\sqrt{1 + z_1}}.$$

Hence,

$$uu^* = \frac{1}{1 + z_1}(z + \mathbf{e}_1)(z^* + \mathbf{e}_1^*) = \frac{1}{1 + z_1}(zz^* + z\mathbf{e}_1^* + \mathbf{e}_1 z^* + \mathbf{e}_1\mathbf{e}_1^*).$$

Now,

$$(I - uu^*)z = z - \frac{1}{1 + z_1}(zz^*z + z\mathbf{e}_1^*z + \mathbf{e}_1 z^*z + \mathbf{e}_1\mathbf{e}_1^*z).$$

By setting $c = \frac{1}{1+z_1}$ and noticing that

$$z\mathbf{e}_1^*z = z_1 z \text{ and } \mathbf{e}_1\mathbf{e}_1^*z = \begin{bmatrix} z_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

then

$$(I - uu^*)z = z - c\,\|z\|_2^2\, z - cz_1 z - c\,\|z\|_2^2\,\mathbf{e}_1 - c\begin{bmatrix} z_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Here z is unitary and

$$z = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix},$$

so

$$(I - uu^*)z = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} - \begin{bmatrix} cz_1 \\ \vdots \\ cz_n \end{bmatrix} - \begin{bmatrix} cz_1^2 \\ \vdots \\ cz_1 z_n \end{bmatrix} - \begin{bmatrix} c \\ 0 \\ \vdots \\ 0 \end{bmatrix} - \begin{bmatrix} cz_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

That is,

$$(I - uu^*)z = \begin{bmatrix} z_1 - cz_1 - cz_1^2 - c - cz_1 \\ z_2 - cz_2 - cz_1 z_2 \\ \vdots \\ z_n - cz_n - cz_1 z_n \end{bmatrix}.$$

Here, for rows 2 through n,

$$z_i - cz_i - cz_1 z_i = z_i - cz_i(1 + z_1),$$

and since $c = \frac{1}{1+z_1}$,

$$z_i - cz_i - cz_1 z_i = z_i - z_i = 0.$$

And for the first row:

$$z_1 - cz_1 - cz_1^2 - c - cz_1 = z_1 - c - cz_1 - cz_1(1 + z_1) = z_1 - c - cz_1 - z_1 =$$

$$= -c - cz_1 = -c(1 + z_1) = -1$$

That is,

$$(I - uu^*)z = \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Finally, by setting $\nu = \frac{\|a\|_2}{\rho}$,

$$\frac{\|a\|_2}{\rho} Hz = \frac{\|a\|_2}{\rho}(I - uu^*)z = -\nu \mathbf{e}_1.$$

□

This result suggests the following $\mathcal{O}(m)$ −where m is the length of the vector a− running-time algorithm to generate Householder transformations.

Its MATLAB implementation can be found in A.2.1.

Input : vector a.
Output: vector u and contant ν.

housegen (a)
  $u \leftarrow a$
  $\nu \leftarrow \|a\|_2$
  if $\nu = 0$ **then**
    $u_1 \leftarrow \sqrt{2}$
    return
  end
  if $u_1 \neq 0$ **then**
    $\rho \leftarrow \frac{\overline{u_1}}{|u_1|}$
  end
  else
    $\rho \leftarrow 1$
  end
  $u_1 \leftarrow 1 + u_1$
  $u \leftarrow \frac{u}{\sqrt{u_1}}$
  $\nu \leftarrow -\overline{\rho}\nu$
end

Algorithm 5: Generation of a Householder transformation

3.4.2 Reduction to Hessenberg form

Theorem 3.4.2 shows how to use elementary reflectors to introduce zeros in a vector. By accumulating these transformations any matrix can be reduced to upper Hessenberg form.

Theorem 3.4.3. *Let A be a matrix of order n. Then there exist a unitary matrix H such that*

$$H^*AH = U$$

where U is in upper Hessenberg form.

Proof. Partition A in the form

$$A = \begin{bmatrix} \alpha_{11} & a_{12}^* \\ a_{21} & A_{22} \end{bmatrix}$$

and let $\hat{H}_1$ be a Householder transformation such that

$$\hat{H}_1 a_{21} = \nu_1 \mathbf{e}_1.$$

By setting $H_1 = \operatorname{diag}(1, \hat{H}_1)$, then

$$H_1 A H_1 = \begin{bmatrix} \alpha_{11} & a_{12}^*\hat{H}_1 \\ \hat{H}_1 a_{21} & \hat{H}_1 A_{22} \hat{H}_1 \end{bmatrix} = \begin{bmatrix} \alpha_{11} & a_{12}^*\hat{H}_1 \\ \nu_1 \mathbf{e}_1 & \hat{H}_1 A_{22} \hat{H}_1 \end{bmatrix}.$$

We have made zero the elements in positions $(k, 1)$ for $k = 3, 4, ..., n$.

For the general step, let us assume that we have generated $k-1$ reflectors such that

$$H_{k-1} \cdots H_1 A H_1 \cdots H_{k-1} = \begin{bmatrix} A_{11} & a_{1k} & A_{1,k+1}^* \\ 0 & \alpha_{kk} & a_{k,k+1}^* \\ 0 & a_{k+1,k} & A_{k+1,k+1} \end{bmatrix}$$

where A_{11} is a Hessenberg matrix of order $k-1$. Now create a transformation such that

$$\hat{H}_k a_{k+1,k} = \nu_k \mathbf{e}_1.$$

And again, by setting $H_k = \text{diag}(I_k, \hat{H}_k)$,

$$H_k H_{k-1} \cdots H_1 A H_1 \cdots H_{k-1} H_k = \begin{bmatrix} A_{11} & a_{1k} & A_{1,k+1}^* \hat{H}_k \\ 0 & \alpha_{kk} & a_{k,k+1}^* \hat{H}_k \\ 0 & \nu_k \mathbf{e}_1 & \hat{H}_k A_{k+1,k+1} \hat{H}_k \end{bmatrix}.$$

For achieving the reduction to upper Hessenberg form of a matrix of order n, $n-2$ of the Householder transformations built above are needed. Once they are generated,

$$H_{n-2} \cdots H_1 A H_1 \cdots H_{n-2} = U$$

will hold, where U is in upper Hessenberg form. Setting $H = H_1 \cdots H_{n-2}$, taking into account that every elementary reflector is Hermitian then $H^* = (H_1 \cdots H_{n-2})^* = H_{n-2}^* \cdots H_1^* = H_{n-2} \cdots H_1$ it can be concluded that

$$H^* A H = U.$$

□

This result can be packed into a neat algorithm −Algorithm 6− that reduces a matrix A to its upper Hessenberg form.

The routine performs both on the first and second loop multiplications of vectors and matrices, which run on $\mathcal{O}(n^2)$. Both loops iterate from 1 to $n-2$ so the total running time is $\mathcal{O}(n^3)$. The MATLAB implementation of Algorithm 6 can be found in A.2.1.

This algorithm is backwards stable. Let $\hat{H}_k$ be the *exact* Householder transformation generated from the computed A_k and let $\hat{Q} = \hat{H}_1 \cdots \hat{H}_{n-2}$. Then there is a matrix E satisfying

$$\frac{\|E\|_2}{\|A\|_2} = \gamma_n \epsilon_M$$

such that the computed Hessenberg matrix H satisfies

$$H = \hat{Q}^*(A + E)\hat{Q}.$$

In this case γ_n is a slowly growing function of n. Thus, the computed Hessenberg form is the exact Hessenberg form of $A + E$ where $\|E\|_2$ is of the order of the rounding unit compared to $\|A\|_2$.

Input : matrix A of order n.
Output: reduced matrix H and transformation matrix Q.

hessreduce (A)
  $H \leftarrow A$
  $Q \leftarrow I$
  for $k \in \{1, ..., n-2\}$ **do**
    /* Generate the Householder transformation that annihilates the k-th column */
    $[u, H_{k+1,k}] \leftarrow \text{housegen}(H_{\{k+1,...,n\},k})$
    $Q_{\{k+1,...,n\},k} \leftarrow u$
    /* Multiply the transformation on the left */
    $v \leftarrow u^* \cdot H_{\{k+1,...,n\},\{k+1,...,n\}}$
    $H_{\{k+1,...,n\},\{k+1,...,n\}} \leftarrow H_{\{k+1,...,n\},\{k+1,...,n\}} - u \cdot v$
    $H_{\{k+2,...,n\},k} \leftarrow 0$
    /* Multiply the transformation on the right */
    $v \leftarrow H_{\{1,...,n\},\{k+1,...,n\}} \cdot u$
    $H_{\{1,...,n\},\{k+1,...,n\}} \leftarrow H_{\{1,...,n\},\{k+1,...,n\}} - v \cdot u^*$
  end
  /* Accumulate the transformations on matrix Q */
  for $k \in \{n-2, ..., 1\}$ **do**
    $u \leftarrow Q_{\{k+1,...,n\},k}$
    $v \leftarrow u^* \cdot Q_{\{k+1,...n\},\{k+1,...,n\}}$
    $Q_{\{k+1,...n\},\{k+1,...,n\}} \leftarrow Q_{\{k+1,...n\},\{k+1,...,n\}} - u \cdot v$
    $Q_{\{1,...,n\},k} \leftarrow \mathbf{e}_k$
  end
end

Algorithm 6: Reduction to upper Hessenberg form

3.4.3 Invariance of the Hessenberg form under a QR step

It will later be implemented a $\mathcal{O}(n^3)$ running time QR routine using the Hessenberg form. But does the QR iteration preserve this form?

Theorem 3.4.4. *Hessenberg form is preserved by QR iteration.*

Proof. Let A be an invertible matrix. Hence, its QR decomposition can be computed using the Gram-Schmidt process. If $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix}$ then its

QR decomposition has the factors

$$Q = \begin{bmatrix} e_1 & \cdots & e_n \end{bmatrix} \text{ and } R = \begin{bmatrix} e_1^* a_1 & \cdots & \cdots & e_1^* a_n \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & e_n^* a_n \end{bmatrix}$$

where

$$u_1 = a_1 \Longrightarrow e_1 = \frac{u_1}{\|u_1\|}$$

$$u_2 = a_2 - \frac{u_1^* a_2}{u_1^* u_1} u_2 \Longrightarrow e_2 = \frac{u_2}{\|u_2\|}$$

$$\vdots$$

$$u_k = a_k - \sum_{j=1}^{k-1} \frac{u_j^* a_k}{u_j^* u_j} u_j \Longrightarrow e_k = \frac{u_k}{\|u_k\|}.$$

Then the decomposition of $A_i - \kappa_i I$ will lead to an upper Hessenberg Q, since the jth column of Q is a linear combination of the leading j columns of $A_i - \kappa_i I$. The RQ product will be upper Hessenberg too, as R is upper triangular. Adding $\kappa_i I$ does not cause losing the upper Hessenberg form.

□

Chapter 4

The explicitly shifted QR algorithm

4.1 Introduction

Once the foundations of an $\mathcal{O}(n^3)$ implementation of the QR algorithm have been laid, it is time to address the technicalities that make it possible. The goal of the QR routine is to compute the Schur triangulation of any given matrix A. Nevertheless, there exist two different Schur decompositions: the complex form and the real form. This chapter explains the technical details of the explicitly shifted QR algorithm, which computes the complex Schur form. Hence,

in this chapter A will be a complex matrix of order n.

4.2 Negligible elements and deflation

If the QR routine is applied to an upper Hessenberg form matrix H, then, from what has been learned in §3.2, $h_{n,n-1}$ is expected to rapidly converge to zero. Simultaneously, from §3.3 other subdiagonal elements $h_{i+1,i}$ may tend to zero and if some of those can be regarded as negligible the problem deflates and computations are saved. The criteria we will use to select negligible subdiagonals is the following:

Lemma 4.2.1. *Let $h_{i+1,i}$ be a subdiagonal element of an upper Hessenberg form matrix H. If*

$$|h_{i+1,i}| \leq \epsilon_M \|A\|_F \quad \textit{where } \epsilon_M \textit{ is the machine epsilon,} \tag{4.1}$$

then setting $h_{i+1,i} = 0$ is, normwise, equivalent to making a relative perturbation in A of the size of the rounding unit.

Proof. To simplify the discussion let us assume that the current value $\hat{A}$ of A has been calculated without error. Then there exists an orthogonal matrix Q such that $\hat{A} = Q^*AQ$. Setting $h_{i+1,i} = 0$ means replacing $\hat{A}$ by $\tilde{A} = \hat{A} - h_{i+1,i}\mathbf{e}_{i+1}\mathbf{e}_i^*$. Now,

$$\tilde{A} = \hat{A} - h_{i+1,i}\mathbf{e}_{i+1}\mathbf{e}_i^* = Q^*AQ - Q^*Qh_{i+1,i}\mathbf{e}_{i+1}\mathbf{e}_i^*Q^*Q,$$

and by taking $E = Qh_{i+1,i}\mathbf{e}_{i+1}\mathbf{e}_i^*Q^*$

$$\tilde{A} = Q^*AQ - Q^*EQ = Q^*(A - E)Q.$$

Now, as was showed in Proposition 1.1.2 the Frobenius norm is unitarily invariant, thus

$$\|Q^*(A - E)Q\|_F = \|A - E\|_F \leq \|A\|_F + \|E\|_F .$$

Here, looking at the definition of E, $E = Qh_{i+1,i}\mathbf{e}_{i+1}\mathbf{e}_i^*Q^*$, so

$$\|E\|_F = \|Qh_{i+1,i}\mathbf{e}_{i+1}\mathbf{e}_i^*Q^*\|_F = \|h_{i+1,i}\mathbf{e}_{i+1}\mathbf{e}_i^*\|_F = |h_{i+1,i}| \, \|\mathbf{e}_{i+1}\mathbf{e}_i^*\|_F = |h_{i+1,i}| .$$

Hence,

$$\|A\|_F + \|E\|_F \leq \|A\|_F + |h_{i+1,i}| ,$$

and by (4.1)

$$\|A\|_F + \|E\|_F \leq \|A\|_F + \epsilon_M \|A\|_F \Longrightarrow \|E\|_F \leq \epsilon_M \|A\|_F \Longrightarrow$$

$$\frac{\|E\|_F}{\|A\|_F} \leq \epsilon_M, \tag{4.2}$$

which means that the perturbation matrix E is of the size of the rounding unit compared to A.

□

When applying the QR algorithm to a Hessenberg matrix, subdiagonal elements tend to converge to zero starting from the element $(n, n-1)$ and proceeding upward on the subdiagonal. That is the reason why algorithms that find negligible elements implement a backsearch. In any case, subdiagonal elements in other positions may deflate too, in which case the routine is to be implemented between those rows. In the case of the following Wilkinson diagram,

$$\left[\begin{array}{ccc|ccc|ccc} \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \hline 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathrm{X} & \mathrm{X} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathrm{X} \end{array}\right],$$

the problem must be deflated twice: first, between rows four and seven and then between rows one and four.

Algorithm 7 searches, starting from the southeast and going up the subdiagonal and in $\mathcal{O}(n)$ time, deflation rows in an upper Hessenberg matrix. If it returns indices i_1 and i_2, where $i_1 < i_2$, then the matrix is to be deflated between those rows. Else, it will return $i_1 = i_2 = 1$ and the matrix is deflated and thus the complex Schur form has been computed. The MATLAB implementation can be consulted in A.3.1.

```
   Input  : upper Hessenberg matrix H, index ℓ.
   Output: indices i_1, i_2.
 1 backsearch (H, ℓ)
 2     i_1 ← ℓ
 3     i_2 ← ℓ
 4     while i_1 > 1 do
 5         if h_{i_1,i_1−1} is negligible then
 6             h_{i_1,i_1−1} ← 0
 7             if i_1 = i_2 then
 8                 i_2 ← i_1 − 1
 9                 i_1 ← i_1 − 1
10             end
11             else
12                 return
13             end
14         end
15         else
16             i_1 ← i_1 − 1
17         end
18     end
19 end
```

Algorithm 7: Algorithm that finds deflation rows in a Hessenberg matrix.

4.3 The Wilkinson shift

After selecting the range $[i_1, i_2]$ into which to perform the QR step, its time to choose a shift. Theorem 3.2.1 settles a shift for which the QR algorithm obtains quadratic convergence, but if we operate on a real matrix H with complex eigenvalues the algorithm will not converge, as complex eigenvalues cannot be approximated by real shifts. We can jump this ditch using the *Wilkinson shift*, which is nothing but the eigenvalue of the matrix

$$W = \begin{bmatrix} h_{i_2-1,i_2-1} & h_{i_2-1,i_2} \\ h_{i_2,i_2-1} & h_{i_2,i_2} \end{bmatrix}$$

nearest to h_{i_2,i_2}. Furthermore, as the method converges, element h_{i_2,i_2-1} will approach to 0. Hence, the submatrix W will be close to upper triangular, so its eigenvalues will be converging to the diagonal elements. Thus, the nearest eigenvalue to h_{i_2,i_2} will be itself. Consequently, the Wilkinson shift will approach the Rayleigh quotient shift as $h_{i_2,i_2-1} \to 0$, maintaining the quadratic convergence.

This 2×2 eigenvalue problem is solved by applying the usual root formula to the characteristic polynomial of the matrix. Let the last matrix be

$$W = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

Then by taking the shifted matrix

$$W - dI = \begin{bmatrix} a-d & b \\ c & 0 \end{bmatrix}$$

the problem has been simplified: no longer is the eigenvalue nearest to d wanted, computing the smallest eigenvalue of $W - dI$ is now the goal. To see why, let (λ, x) be an eigenpair of $W - dI$. Then,

$$(W - dI)x = \lambda x \Longrightarrow Wx - dx = \lambda x \Longrightarrow Wx = (\lambda + d)x.$$

In other words, the eigenvalues of W can be written as $\kappa = \lambda + d$, where λ is an eigenvalue of $W - dI$ and d is southeastest element of W. As a result, the eigenvalue of W closest to d is the smallest eigenvalue, in absolute value, of $W - dI$ plus d. Now, take the characteristic polynomial

$$p_{W-dI}(\lambda) = \lambda^2 - (a-d)\lambda - bc$$

and set $p = \frac{a-d}{2}$ and $q = bc$. Then,

$$p_{W-dI}(\lambda) = \lambda^2 - 2p\lambda - q.$$

Its roots are

$$\lambda = p \pm \sqrt{p^2 + q} = p \pm r.$$

Notice that r is the square root of $p^2 + q$, so if this sum happens to be very small then there is danger of underflow when applying the square root. To avoid this, the largest root is computed first and then the smallest is deduced using the relation $\lambda_{\min}\lambda_{\max} = -q$. The largest root is determined using $|\lambda|^2 = |p \pm r|^2 = |p|^2 \pm 2 \cdot \mathrm{Re}\,(p\bar{r}) + |r|^2$ −line 10 of Algorithm 8−. All of the above can be neatly resumed in the following routine −whose MATLAB code is in A.3.2−:

Input : elements a, b, c and d of the matrix B.
Output: eigenvalue of B nearest to d.

```
1  wilkshift (a, b, c, d)
2      κ ← d
3      s ← |a| + |b| + |c| + |d|
4      if s = 0 then
5          return
6      end
7      q ← (b/s) · (c/s)
8      if q ≠ 0 then
9          p ← 0.5 · ((a/s) − (d/s))
10         r ← √(p² + q)
11         if Re(p) · Re(r) + Im(p) · Im(r) < 0 then
12             r ← −r
13         end
14         κ ← κ − s · (q/(p+r))
15     end
16 end
```

Algorithm 8: Computation of the Wilkinson shift

Notice the scaling factor s in line 3 of Algorithm 8. It is cancelled in line 14 and it is introduced so that the product bc does not overflow. There may also be scaling factors cheaper to compute, e.g., $s = \mathrm{Re}(a) + \mathrm{Im}(a) + \mathrm{Re}(b) + \mathrm{Im}(b)$. The algorithm runs in constant time $\mathcal{O}(1)$.

4.4 Implicit QR factorization and RQ product

The goal of the QR routine is to introduce zeros in the subdiagonal of an upper Hessenberg matrix. Although we already know how to introduce those zeros by using Householder transformations, it is more efficient to do so in a different way.

Definition 4.4.1. A *plane rotation* −or *Givens rotation*− is a matrix of the form

$$P = \begin{bmatrix} c & s \\ -\overline{s} & \overline{c} \end{bmatrix} \text{ where } |c|^2 + |s|^2 = 1.$$

Remark 4.4.1. The transformation is called a rotation because in the real case it rotates 2-vectors clockwise through the angle $\theta = \arccos(c)$.

Lemma 4.4.1. *Plane rotations can introduce zeros in a* 2*-vector.*

Proof. Let $a \neq 0$ and let $\begin{bmatrix} a \\ b \end{bmatrix}$ be a 2-vector. Set

$$\nu = \sqrt{|a|^2 + |b|^2},\ c = \frac{|a|}{\nu},\ \text{and}\ s = \frac{a}{|a|}\frac{\overline{b}}{\nu}.$$

Then

$$\begin{bmatrix} c & s \\ -\overline{s} & \overline{c} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \frac{|a|}{\nu} & \frac{a\overline{b}}{|a|\nu} \\ -\frac{\overline{a}b}{|a|\nu} & \frac{|a|}{\nu} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \frac{a|a|}{\nu} + \frac{a|b|^2}{\nu|a|} \\ -\frac{|a|^2 b}{|a|\nu} + \frac{|a|b}{\nu} \end{bmatrix} = \begin{bmatrix} \frac{a(|a|^2+|b|^2)}{\nu|a|} \\ 0 \end{bmatrix} = \begin{bmatrix} \nu a/\,|a| \\ 0 \end{bmatrix}.$$

That is,

$$\begin{bmatrix} c & s \\ -\overline{s} & \overline{c} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \nu a/\,|a| \\ 0 \end{bmatrix}. \tag{4.3}$$

□

Zeros can be introduced anywhere in a matrix in a similar way.

Definition 4.4.2. A *plane rotation in the* (i, j)*-plane* is a matrix of the form

$$P_{ij} = \begin{array}{c} \\ \\ \\ \text{i} \\ \\ \text{j} \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & & \text{i} & & & \text{j} & & \end{array} \\ \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & c & \cdots & s & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & -\overline{s} & \cdots & \overline{c} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix} \end{array}.$$

That is, a rotation in the (i, j)-plane is an identity matrix in which a plane rotation has been introduced in the submatrix corresponding to rows and columns i and j.

Remark 4.4.2. Let us see the effect of premultiplication and postmultiplication by P_{ij}:

- Let
$$Y = P_{ij}X.$$
If X and Y are partitioned by rows
$$y_i^* = cx_i^* + sx_j^*,$$

$$y_j^* = cx_j^* - \overline{s}x_i^*,$$

$$y_k^* = x_k^* \ \forall \ k \neq i, k.$$

Expressly, premultiplication by a rotation in the (i, j)-plane combines rows i and j and leaves other intact.

- Let

$$Y = XP_{ij}^*.$$

Then if X and Y are partitioned by columns

$$y_i = \overline{c}x_i + \overline{s}x_j,$$

$$y_j = -sx_i + cx_j,$$

$$y_k = x_k \ \forall \ k \neq i, j.$$

In other words, postmultiplication by a rotation in the (i, j)-plane alters only columns i and j.

To sum up, Algorithm 9 generates a Givens rotation satisfying (4.3) in $\mathcal{O}(1)$ time. If $b = 0$, then it returns the identity rotation, and if $a = 0$, exchanges the two elements of the vector. The scalar τ is nothing but a scaling factor to avoid both overflow and underflow; and it could be replaced by, for example, $\tau = |\mathrm{Re}(a)| + |\mathrm{Im}(a)| + |\mathrm{Re}(b)| + |\mathrm{Im}(b)|$ −this version of τ saves computations. c and s, too, could be defined in a different way in Lemma 4.4.1, but the version given makes the final value of c real, which saves some work in practice. The MATLAB code of Algorithm 9 can be found in A.3.3.

On the other hand, Algorithm 10 is a simple routine combining the only two vectors that are involved in the application of a rotation generated by Algorithm 9. Its MATLAB code can be found in A.3.3. It works in $\mathcal{O}(n)$ and can be used in different ways:

$$PX \equiv rotapp(c, s, X_{i,\{1,\ldots,n\}}, X_{j,\{1,\ldots,n\}}),$$

$$P^*X \equiv rotapp(c, -s, X_{i,\{1,\ldots,n\}}, X_{j,\{1,\ldots,n\}}),$$

$$XP \equiv rotapp(c, -\overline{s}, X_{i,\{1,\ldots,n\}}, X_{j,\{1,\ldots,n\}}),$$

$$XP^* \equiv rotapp(c, \overline{s}, X_{i,\{1,\ldots,n\}}, X_{j,\{1,\ldots,n\}}).$$

Input : scalars a and b.
Output: overwritten scalars a and b, and scalars c and s defining the plane rotation.

rotgen (a, b)
  if $b = 0$ **then**
    $c \leftarrow 1$
    $s \leftarrow 0$
    return
  end
  if $a = 0$ **then**
    $c \leftarrow 0$
    $s \leftarrow 1$
    $a \leftarrow b$
    $b \leftarrow 0$
  end
  $\mu \leftarrow a/|a|$
  $\tau \leftarrow |a| + |b|$
  $\nu \leftarrow \tau \cdot \sqrt{|a/\tau|^2 + |b/\tau|^2}$
  $c \leftarrow |a|/\nu$
  $s \leftarrow \nu \cdot \overline{b}/\nu$
  $a \leftarrow \nu \cdot \mu$
  $b \leftarrow 0$
end

Algorithm 9: Generation of a plane rotation

Input : scalars c and s, vectors x and y.
Output: overwritten vectors x and y.

rotapp (c, s, x, y)
  $t \leftarrow c \cdot x + s \cdot y$
  $y \leftarrow c \cdot y - \overline{s} \cdot x$
  $x \leftarrow t$
end

Algorithm 10: Application of a plane rotation

Finally, it is time to put together all the pieces developed previously into an algorithm that reduces an upper Hessenberg matrix to its complex Schur form. If the upper triangular form of any given matrix A is what we want, then it has to be used alongside *hessreduce* −Algorithm 6. Provided a deflation strategy is used, then it is stable in the usual sense −Definition 1.1.4. This implementation returns an error when a maximum iteration count is exceeded, so that it will not run forever if convergence does not happen. Real-life implementations, on the contrary, try ad hoc shifts to push the algorithm into convergence.

There is no theoretical result assuring the global convergence of the algorithm −check Remark 3.2.2−, but tipically *hqr* takes several iterations to deflate to the first eigenvalue, then some less to deflate to the second one, and so on. Eventually the subdiagonals of H become small enough so that quadratic convergence is achieved. If at some point $i_1 > 1$ further computations are saved. Assuming at most k iterations are needed to compute each eigenvalue and that $i_1 = 1$ then we get an upper bound to the operation count of kn^3, i.e., *hqr* runs, at most, in $\mathcal{O}(n^3)$. The MATLAB code is in A.3.3.

Remark 4.4.3. As a follow up to Remark 3.2.2, we can now give an example of a matrix that fails to converge under QR iteration:

$$Y = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Feel free to try it in the implementation given above. Practical algorithms use exceptional shifts for the matrices that do not converge. That is why, if you try to compute the eigenvalues of Y with, for example, the commands MATLAB, LAPACK or Wolfram Mathematica have implemented it will converge.

Input : upper Hessenberg matrix H, upper bound for iterations $maxiter$.
Output: real Schur form overwritten in H, transformation matrix Q.

hqr $(H, maxiter)$
  $i_1 \leftarrow 1$
  $i_2 \leftarrow n$
  $iter \leftarrow 0$
  $c \leftarrow \{0, \overset{n}{...}, 0\}$
  $s \leftarrow \{0, \overset{n}{...}, 0\}$
  while *true* **do**
    $iter \leftarrow iter + 1$
    if $iter > maxiter$ **then**
      error
    end
    $oldi_2 \leftarrow i_2$
    $\{i_1, i_2\} \leftarrow \text{backsearch}(H, i_2)$
    if $i_2 = 1$ **then**
      return
    end
    if $i_2 \neq oldi_2$ **then**
      $iter \leftarrow 0$
    end
    $\kappa \leftarrow \text{wilkshift}(h_{i_2-1,i_2-1}, h_{i_2-1,i_2}, h_{i_2,i_2-1}, h_{i_2,i_2})$
    $h_{i_1,i_1} \leftarrow h_{i_1,i_1} - \kappa$
    for $i \in \{i_1, ..., i_2 - 1\}$ **do**
      $\{h_{i,i}, h_{i+1,i}, c_i, s_i\} \leftarrow \text{rotgen}(h_{i,i}, h_{i+1,i})$
      $h_{i+1,i+1} \leftarrow h_{i+1,i+1} - \kappa$
      $\{H_{i,\{i+1,...,n\}}, H_{i+1,\{i+1,...,n\}}\} \leftarrow$
       $\text{rotapp}(c_i, s_i, H_{i,\{i+1,...,n\}}, H_{i+1,\{i+1,...,n\}})$
    end
    for $i \in \{i_1, ..., i_2 - 1\}$ **do**
      $\{H_{\{1,...,i+1\},i}, H_{\{1,...,i+1\},i+1}\} \leftarrow$
       $\text{rotapp}(c_i, \overline{s}_i, H_{\{1,...,i+1\},i}, H_{\{1,...,i+1\},i+1})$
      $\{Q_{\{1,...,n\},i}, Q_{\{1,...,n\},i+1}\} \leftarrow$
       $\text{rotapp}(c_i, \overline{s}_i, Q_{\{1,...,n\},i}, Q_{\{1,...,n\},i+1})$
      $h_{i,i} \leftarrow h_{i,i} + \kappa$
    end
    $h_{i_2,i_2} \leftarrow h_{i_2,i_2} + \kappa$
  end
end

Algorithm 11: Schur form of an upper Hessenberg matrix

4.5 Eigenvectors of the complex Schur form

Now that we know how to compute the complex Schur decomposition of a general complex matrix A, and thus its eigenvalues, it is time to compute the inseparable companion of the eigenvalues: the eigenvectors. If $A = QTQ^*$ is the complex Schur decomposition of A and Y is the matrix of right eigenvectors of T, then the matrix of right eigenvectors of A is QY. Let us see why. Let (λ_i, x_i) be an eigenpair of A for all $i = 1, ..., n$. Thus, due to $Q^*A = TQ^*$,

$$Ax_i = \lambda_i x_i \Longrightarrow Q^*Ax_i = Q^*x_i\lambda_i \text{ for every } i \in \{1, ..., n\}.$$

Hence, by setting $X = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}$ and $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$,

$$Q^*AX = Q^*X\Lambda \Longrightarrow TQ^*X = Q^*X\Lambda.$$

Taking $Q^*X = Y$,

$$TY = Y\Lambda,$$

thus Y contains the right eigenvectors of T in columns. Since we set $Q^*X = Y$, then $X = QY$ which is what we wanted to prove. This means that if Y is the matrix of right eigenvectors of T then QY is the matrix of right eigenvectors of A.

Because of this, we can use the Schur decompositions eigenvectors to compute those of A. Let T be the Schur decomposition of A and partition it the following way:

$$T = \begin{bmatrix} T_{11} & t_{1k} & T_{1,k+1} \\ 0 & \tau_{kk} & t^*_{k,k+1} \\ 0 & 0 & T_{k+1,k+1} \end{bmatrix}.$$

If τ_{kk} is a simple eigenvalue of T, then

$$\begin{bmatrix} -(T_{11} - \tau_{kk}I)^{-1}t_{1k} \\ 1 \\ 0 \end{bmatrix}$$

is an eigenvector of T, that is,

$$\begin{bmatrix} T_{11} & t_{1k} & T_{1,k+1} \\ 0 & \tau_{kk} & t^*_{k,k+1} \\ 0 & 0 & T_{k+1,k+1} \end{bmatrix} \begin{bmatrix} -(T_{11} - \tau_{kk}I)^{-1}t_{1k} \\ 1 \\ 0 \end{bmatrix} = \tau_{kk} \begin{bmatrix} -(T_{11} - \tau_{kk}I)^{-1}t_{1k} \\ 1 \\ 0 \end{bmatrix}.$$

Lets prove it. First, realize that

$$\begin{bmatrix} T_{11} & t_{1k} & T_{1,k+1} \\ 0 & \tau_{kk} & t^*_{k,k+1} \\ 0 & 0 & T_{k+1,k+1} \end{bmatrix} \begin{bmatrix} -(T_{11} - \tau_{kk}I)^{-1}t_{1k} \\ 1 \\ 0 \end{bmatrix} = \tau_{kk} \begin{bmatrix} -(T_{11} - \tau_{kk}I)^{-1}t_{1k} \\ 1 \\ 0 \end{bmatrix}$$

if and only if

$$\begin{bmatrix} T_{11} & t_{1k} & T_{1,k+1} \\ 0 & \tau_{kk} & t^*_{k,k+1} \\ 0 & 0 & T_{k+1,k+1} \end{bmatrix} \begin{bmatrix} -(T_{11} - \tau_{kk}I)^{-1}t_{1k} \\ 1 \\ 0 \end{bmatrix} - \tau_{kk} \begin{bmatrix} -(T_{11} - \tau_{kk}I)^{-1}t_{1k} \\ 1 \\ 0 \end{bmatrix} = 0.$$

Now,

$$\begin{bmatrix} -T_{11}(T_{11} - \tau_{kk}I)^{-1}t_{1k} + t_{1k} \\ \tau_{kk} \\ 0 \end{bmatrix} - \begin{bmatrix} -\tau_{kk}(T_{11} - \tau_{kk}I)^{-1}t_{1k} \\ \tau_{kk} \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} -T_{11}(T_{11} - \tau_{kk}I)^{-1}t_{1k} + t_{1k} + \tau_{kk}(T_{11} - \tau_{kk}I)^{-1}t_{1k} \\ 0 \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} -(T_{11} - \tau_{kk}I)(T_{11} - \tau_{kk}I)^{-1}t_{1k} + t_{1k} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -t_{1k} + t_{1k} \\ 0 \\ 0 \end{bmatrix} = 0$$

On to the main argument, writing the kth eigenvector in the form

$$\begin{bmatrix} x_1^{(k)} \\ 1 \\ 0 \end{bmatrix}$$

the first row can be obtained by solving the upper triangular system

$$(T_{11} - \tau_{kk}I)x_1^{(k)} = -t_{1k}, \tag{4.4}$$

and by writing this system in the more general form $(T - \mu I)x = b$ an algorithm to compute $x_1^{(k)}$ can be derived more easily. Partitioning the last equality as

$$\begin{bmatrix} T_* - \mu I & t_* \\ 0 & \tau - \mu \end{bmatrix} \begin{bmatrix} x_* \\ \xi \end{bmatrix} = \begin{bmatrix} b_* \\ \beta \end{bmatrix} \tag{4.5}$$

and looking to the second row gives $(\tau - \mu)\xi = \beta$, and so

$$\xi = \frac{\beta}{\tau - \mu}.$$

From the first row of (4.5) we infer that

$$(T_* - \mu I)x_* + \xi t_* = b_* \Longrightarrow (T_* - \mu I)x_* = b_* - \xi t_*,$$

which is a triangular system of order one less than the original, which can be further reduced by recursively applying the same procedure. All this gives the following algorithm to solve original system (4.4):

```
1 x ← b
2 for j ∈ {n, n − 1, ..., 1} do
3 |   x_j ← x_j/(T_jj − μ)
4 |   x_{1,...,j−1} ← x_{1,...,j−1} − x_j · T_{1,...,j−1},j
5 end
```

This is, grosso modo, how the eigenvectors of a complex Schur form will be computed. Two details have to be taken care of though. The first one: the shift μ applied in (4.5) will be a diagonal element of the matrix T; then, if the matrix has multiple eigenvalues, some diagonal element of T_{11} may be equal to τ_{kk} and the algorithm will try to divide by zero. This is avoided by substituting values of $\tau - \mu$ smaller than $\mu \cdot \epsilon_M$ by this quantity. It seems inappropriate to substitute an essentially zero value by an arbitrary number, but as seen in Lemma 4.2.1, this replacement corresponds to a small perturbation in A.

The second problem is that the values of an eigenvector can be of widely varying size. Thus, computing them may result in overflow. Since eigenvectors are defined up to scalar factor, we will normalize them when there is danger of overflowing. This normalization, at the same time, can provoke the underflow of other elements. However, as it has been seen in Lemma 4.2.1, setting them to zero is equivalent to a small perturbation in A.

Algorithm 12 implements in $\mathcal{O}(n^3)$ the computation of the eigenvectors of a triangular matrix A. The MATLAB code can be found in A.3.4. Some comments about it:

- Numbers *smallnum* and *bignum* are considered to avoid underflow and overflow, respectively. The first one is usually taken as $\frac{n}{\epsilon_M} w$ where w is a number just above the underflow point. In a similar way, the second one is chosen as $\frac{\epsilon_M}{n} v$ where v is a number near the overflow point.

- The algorithm is stable in the sense that each eigenvector satisfies

$$(T + E_i)x_i = t_{ii}x_i \text{ where } \frac{\|E_i\|}{\|T\|} \leq \gamma \epsilon_M \tag{4.6}$$

  for some constant γ. The matrix E_i is different for every eigenvector thus the eigenvalue-eigenvector decomposition may not be stable in the usual sense.

Input : upper triangular matrix T
Output: matrix X containing, in columns, the normalized right eigenvectors of T.

righteigvec (T)
  $n \leftarrow \text{size}(T)$
  $smallnum \leftarrow$ a small number above the underflow point
  $bignum \leftarrow$ a number near the overflow point
  for $k \in \{n, n-1, ..., 1\}$ **do**
    $X_{\{1,...,k-1\},k} \leftarrow -T_{\{1,...,k-1\},k}$
    $x_{k,k} \leftarrow 1$
    $X_{\{k+1,...,n\},k} \leftarrow 0$
    $dmin \leftarrow \max\{\epsilon_M \cdot |t_{k,k}|, smallnum\}$
    for $j \in \{k-1, ..., 1\}$ **do**
      $d \leftarrow t_{j,j} - t_{k,k}$
      if $|d| \leq dmin$ **then**
        $d \leftarrow dmin$
      end
      if $|x_{j,k}| / bignum \geq |d|$ **then**
        $s \leftarrow |d| / |x_{j,k}|$
        $X_{\{1,..,k\},k} \leftarrow s \cdot X_{\{1,...,k\},k}$
      end
      $x_{j,k} \leftarrow x_{j,k}/d$
      $X_{\{1,...,j-1\},k} \leftarrow X_{\{1,...,j-1\},k} - x_{j,k} \cdot T_{\{1,...,j-1\},j}$
    end
    $X_{\{1,...,k\},k} \leftarrow X_{\{1,...,k\},k} / \left\| X_{\{1,...,k\},k} \right\|_2$
  end
end

Algorithm 12: Right eigenvectors of an upper triangular matrix

Chapter 5

The implicitly shifted QR algorithm

5.1 Introduction

The previous chapter describes a variant of the QR algorithm that computes the complex Schur form of any given complex matrix. All the same, complex arithmetic is much more expensive than real arithmetic and thus should be avoided when possible. The Hessenberg reduction of a real matrix happens in real arithmetic, therefore only the reduction to Schur form has to be altered to avoid complex arithmetic. The implicitly shifted QR routine is an adaptation of the previous algorithm that computes the real Schur form of any matrix as such decomposition always exists. But first, a note:

From now on A will be a real matrix of order n.

Now, on to the main point.

Theorem 5.1.1 (Real Schur form)**.** *Let A be of order n. Then there is an orthogonal matrix U such that $T = U^t AU$ is block upper triangular of the form*

$$T = U^t AU = \begin{bmatrix} T_{11} & T_{12} & T_{13} & \cdots & T_{1k} \\ 0 & T_{22} & T_{23} & \cdots & T_{2k} \\ 0 & 0 & T_{33} & \cdots & T_{3k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & T_{kk} \end{bmatrix}. \tag{5.1}$$

The diagonal blocks of T are of order one or two. The block of order one contain the real eigenvalues of A. The blocks of order two contain the pairs of complex conjugate eigenvalues of A. The blocks can be made to appear in any order.

Proof. Let (λ, x) be a complex eigenpair where

$$\lambda = \mu + i\nu \text{ and } x = y + iz.$$

Firstly, note the following two equalities:

$$x + \overline{x} = y + iz + y - iz = 2y \Longrightarrow y = \frac{x + \overline{x}}{2}$$

$$x - \overline{x} = y + iz - y + iz = 2iz \Longrightarrow z = \frac{i(\overline{x} - x)}{2}.$$

Then by using these identities and the fact that if $Ax = \lambda x$ then $A\overline{x} = \overline{\lambda x}$,

$$Ay = A\left(\frac{x + \overline{x}}{2}\right) = \frac{Ax}{2} + \frac{A\overline{x}}{2} = \frac{\lambda x}{2} + \frac{\overline{\lambda x}}{2} = \frac{x(\mu + i\nu)}{2} + \frac{\overline{x}(\mu - i\nu)}{2} =$$

$$\frac{\mu x}{2} + \frac{\mu\overline{x}}{2} + \frac{i\nu x}{2} - \frac{i\nu\overline{x}}{2} = \mu\left(\frac{x + \overline{x}}{2}\right) - \nu\left(\frac{i(\overline{x} - x)}{2}\right) = \mu y - \nu z$$

and

$$Az = A\left(\frac{i(\overline{x} - x)}{2}\right) = \frac{iA\overline{x}}{2} - \frac{iAx}{2} = \frac{i\overline{\lambda x}}{2} - \frac{i\lambda x}{2} = \frac{i\overline{x}(\mu - i\nu)}{2} - \frac{ix(\mu + i\nu)}{2} =$$

$$\mu\left(\frac{i(\overline{x} - x)}{2}\right) + \nu\left(\frac{\overline{x} + x}{2}\right) = \nu y + \mu z.$$

That is, $Ay = \mu y - \nu z$ and $Az = \nu y + \mu z$; or more compactly

$$A\begin{bmatrix} y & z \end{bmatrix} = \begin{bmatrix} y & z \end{bmatrix}\begin{bmatrix} \mu & \nu \\ -\nu & \mu \end{bmatrix} = \begin{bmatrix} y & z \end{bmatrix} L.$$

(Note that the eigenvalues of L are λ and $\overline{\lambda}$:

$$\begin{vmatrix} \mu - \upsilon & \nu \\ -\nu & \mu - \upsilon \end{vmatrix} = (\upsilon - \mu)^2 + \nu^2 = \upsilon^2 - 2\mu\upsilon + \mu^2 + \nu^2 \Longrightarrow$$

$$\upsilon = \frac{2\mu \pm \sqrt{4\mu^2 - 4\mu^2 - 4\nu^2}}{2} = \frac{2\mu \pm 2i\nu}{2} = \mu \pm i\nu \Longrightarrow \upsilon = \lambda \text{ or } \upsilon = \overline{\lambda}.)$$

Now let $\begin{bmatrix} y & z \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} R \\ 0 \end{bmatrix}$ be the QR decomposition of $\begin{bmatrix} y & z \end{bmatrix}$, then $\begin{bmatrix} y & z \end{bmatrix} = Q_1 R$. It can be proved −Theorem 1.3 of [1]− that if a complex vector is an eigenvector of a matrix then its imaginary and real parts are independent; thus y and z are independent, R is nonsingular and so $Q_1 = \begin{bmatrix} y & z \end{bmatrix} R^{-1}$. Hence,

$$AQ_1 = A\begin{bmatrix} y & z \end{bmatrix} R^{-1} = \begin{bmatrix} y & z \end{bmatrix} LR^{-1} = Q_1 RLR^{-1}.$$

Now,

$$\begin{bmatrix} Q_1^t \\ Q_2^t \end{bmatrix} A \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} Q_1^t A Q_1 & Q_1^t Q A_2 \\ Q_2^t A Q_1 & Q_2^t A Q_2 \end{bmatrix} = \begin{bmatrix} Q_1^t Q_1 R L R^{-1} & Q_1^t A Q_2 \\ Q_2^t Q_1 R L R^{-1} & Q_2^t A Q_2 \end{bmatrix}$$

and remembering that the columns and rows of the Q-factor of a QR decomposition form an orthonormal basis

$$\begin{bmatrix} Q_1^t \\ Q_2^t \end{bmatrix} A \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} R L R^{-1} & Q_1^t A Q_2 \\ 0 & Q_2^t A Q_2 \end{bmatrix},$$

and thus the deflation is completed. RLR^{-1} is similar to L so its eigenvalues are λ and $\overline{\lambda}$. Following the notation given in (5.1) we have proved that

$$U_1^t A U_1 = \begin{bmatrix} T_{11} & Q_1^t A Q_2 \\ 0 & Q_2^t A Q_1 \end{bmatrix}.$$

Therefore, to complete the proof and reach the matrix given in (5.1), the matrix $Q_2^t A Q_1$ has to be deflated by blocks. This is achieved by repeating the process above for the complex eigenpairs, and the process seen at the beginning of §3.2 −but with right eigenpairs instead of left eigenpairs− for the real eigenpairs.

□

A single step of the implicitly shifted QR algorithm does three things: look for almost zero −or negligible− elements and 2×2 blocks and deflate the problem if any are found, compute the double shift and chase the bulge. The next three sections explain and implement each of those parts.

5.2 Negligible 2×2 blocks and deflation

The convergence of the shifted QR algorithm, once it is asymptotic, depends on the eigenvalues of the matrix that define the shift, i.e.,

$$\begin{bmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{nn} \end{bmatrix}.$$

If those eigenvalues are complex and nondefective $h_{n-1,n-2}$ converges quadratically to zero, and if they are real and nondefective, both $h_{n-1,n-2}$ and $h_{n,n-1}$ converge quadratically to zero. Else, elements $h_{n-1,n-2}$ or $h_{n,n-1}$ may slowly converge to zero and so deflation of both simple elements and 2×2 blocks must be allowed. Algorithm 13 is a generalization of Algorithm 7 that allows the deflation of those blocks and simple elements. Negligible elements are detected with the same criteria as in (4.1). The MATLAB implementation is in A.4.1.

Input : Hessenberg matrix H, index ℓ.
Output: indices i_1 and i_2 such that one of following condition holds:

(i) $1 \le i_1 < i_2 \le \ell$ and the matrix deflates at rows i_1 and i_2.

(ii) $1 = i_1 = i_2$ and the matrix is deflated.

```
1  backsearch2 (H, ℓ)
2      i1 ← ℓ
3      i2 ← ℓ
4      while i1 > 1 do
5          if H_{i1,i1−1} is negligible or i2 = 2 then
6              if i2 ≠ 2 then
7                  H_{i1,i1−1} ← 0
8              end
9              if i1 = i2 − 1 or i2 = 2 then
10                 /* Process the 2 × 2 block                    */
11                 if i2 ≠ 2 then
12                     i1 ← i1 − 1
13                     i2 ← i1 − 1
14                 end
15                 else
16                     i1 ← 1
17                     i2 ← 1
18                 end
19             end
20             else if i1 = i2 then
21                 i1 ← i1 − 1
22                 i2 ← i1 − 1
23             end
24             else
25                 return
26             end
27         end
28         else
29             i1 ← i1 − 1
30         end
31     end
32 end
```

Algorithm 13: Finding deflation rows in a real upper Hessenberg matrix.

5.3 The double shift

The key to perform the QR routine in real arithmetic is a happy little idea. Let κ be a complex Wilkinson shift, then its conjugate $\overline{\kappa}$ is also a candidate for a shift. Assume two steps of the QR algorithm are applied, one with shift κ and the second with shift $\overline{\kappa}$, resulting in the matrix $\hat{H}$. By Theorem 3.3.1 if

$$\breve{Q}\breve{R} = (H - \kappa I)(H - \overline{\kappa} I)$$

is the QR factorization of $(H - \kappa I)(H - \overline{\kappa} I)$, then $\hat{H} = \breve{Q} H \breve{Q}$. But

$$(H - \kappa I)(H - \overline{\kappa} I) = H^2 - (\kappa + \overline{\kappa})H + \kappa\overline{\kappa} I = H^2 - 2\,\mathrm{Re}(\kappa) H + |\kappa|\, I,$$

which is real; and as H is real then so must be $\breve{Q}$ and $\hat{H}$. This method is referred to as the *Francis double shift* strategy.

So now the plan is to compute $H^2 - 2\,\mathrm{Re}(\kappa)H + |\kappa|\, I$, then calculate its Q-factor $\breve{Q}$ and produce $\hat{H} = \breve{Q} H \breve{Q}$. But this is not practical at all, as the first two operations run in $\mathcal{O}(n^3)$. Fortunately, Francis itself provided a sling for this broken arm by demonstrating a property of the upper Hessenberg matrices.

5.3.1 Implicit Q theorem

The reduction of a matrix A to upper Hessenberg form is not unique −check Lemma 5.3.1. Nonetheless, there is some limit to this nonuniqueness and that is what the implicit Q theorem proves. But first, some hypothesis must be provided in order to prove the result.

Lemma 5.3.1. *Let $H = Q^t A Q$ be a unitary reduction of A to Hessenberg form. Then there exists another reduction to Hessenberg form $\hat{H} = \hat{Q}^t A \hat{Q}$ that only differs on the rescaling of the elements of $\hat{H}$ and the scaling of the columns of $\hat{Q}$ by a modulus of factor one. Thus the rescalling does not make essential difference and the reduction is said to be* determined up to column scaling of Q.

Proof. Let D be a matrix of order n with $|D| = I$ −remember Remark 3.3.1. Defining $\hat{Q} = QD$, the columns of $\hat{Q}$ are nothing but the columns of Q rescaled by a factor of modulus one. Now

$$\hat{H} = \hat{Q}^t A \hat{Q} = D Q^t A Q D = D H D,$$

which means that $\hat{h}_{ij} = d_{ii} h_{ij} d_{jj}$. Therefore $\hat{H}$ is also upper Hessenberg and its elements are the elements of H rescaled. □

Definition 5.3.1. Let H be upper Hessenberg of order n. Then H is *unreduced* if $h_{i+1,i} \neq 0\ \forall i \in \{1, ..., n-1\}$.

Theorem 5.3.2 (Implicit Q theorem)**.** *Let A be of order n and let $H = Q^tAQ$ be a unitary reduction of A to upper Hessenberg form. If H is unreduced then up to column scaling of Q the matrices Q and H are uniquely determined by the first column of Q.*

Proof. By hypothesis,

$$H = Q^tAQ \Longrightarrow QH = AQ,$$

then by partitioning $Q = \begin{bmatrix} q_1 & \dots & q_n \end{bmatrix}$ and taking into account that H is upper Hessenberg

$$Aq_1 = h_{11}q_1 + h_{21}q_2. \tag{5.2}$$

So by premultiplying (5.2) by q_1^t and remembering that the columns of the Q-factor form a orthonormal basis, then

$$h_{11} = q_1^tAq_1.$$

Now that the north westest element of H is known,

$$h_{21}q_2 = Aq_1 - h_{11}q_1,$$

where the right part of the equality must be nonzero since it was hypothesized that $h_{21} \neq 0$. $\|q_1\|_2 = 1$ thus by taking $h_{21} = \|Aq_1 - h_{11}q_1\|_2$ then

$$q_2 = \frac{Aq_1 - h_{11}q_1}{\|Aq_1 - h_{11}q_1\|_2}.$$

For the general case, assuming that $q_1, \dots, q_k$ have been defined and with them the elements of the respective columns of H

$$Aq_k = h_{1k}q_1 + \cdots + h_{kk}q_k + h_{k+1,k}q_{k+1}.$$

By orthonormality of the columns of Q,

$$h_{ik} = q_i^tAq_k, \text{ for all } i \in \{1, \dots, k\}$$

and

$$h_{k+1,k}q_{k+1} = Aq_k - h_{1k}q_1 - \cdots - h_{kk}q_k.$$

Since H is unreduced then $h_{k+1,k} \neq 0$ and so must be $Aq_k - h_{1k}q_1 - \cdots - h_{kk}q_k$, hence by taking

$$h_{k+1,k} = \|Aq_k - h_{1k}q_1 - \cdots - h_{kk}q_k\|_2$$

q_{k+1} is determined up to a modulus of factor one.

Lastly, from the last column of (5.2)

$$h_{in} = q_i^tAq_n, \text{ for all } i \in \{1, \dots, n\}.$$

□

5.3.2 Implementation of the double shift

Let us go back to the plan defined at the beginning of this section: compute $C = H^2 - 2\,\mathrm{Re}(\kappa)H + |\kappa|\,I$, calculate its Q-factor $\breve{Q}$ and produce $\hat{H} = \breve{Q}H\breve{Q}$. The implicit Q theorem, provided the first column of the matrix C, paves the path to an efficient implementation of the Francis double shift. Remembering the proof of Theorem 3.4.4, the first column of the Q-factor $\breve{Q}$ is nothing but the first column of C divided by its norm. Therefore to compute $\breve{Q}$ and produce $\hat{H} = \breve{Q}H\breve{Q}$ we only need the first column of C. Let us see how to calculate it without explicitly computing C itself.

We do not need to compute κ itself to know $\mathrm{Re}\,(\kappa)$ and $|\kappa|$, because κ and $\overline{\kappa}$ are the eigenvalues of the matrix

$$\begin{bmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{bmatrix},$$

therefore the roots of the characteristic polynomial

$$x^2 - (h_{n-1,n-1} + h_{n,n})x + (h_{n-1,n-1}h_{n,n} - h_{n-1,n}h_{n,n-1});$$

from where

$$x^2 - (h_{n-1,n-1} + h_{n,n})x + (h_{n-1,n-1}h_{n,n} - h_{n-1,n}h_{n,n-1}) = (x-\kappa)(x-\overline{\kappa}),$$

and

$$x^2 - (h_{n-1,n-1} + h_{n,n})x + (h_{n-1,n-1}h_{n,n} - h_{n-1,n}h_{n,n-1}) = x^2 - (\kappa+\overline{\kappa})x - \kappa\overline{\kappa}.$$

Hence,

$$t = 2\,\mathrm{Re}\,(\kappa) = \kappa + \overline{\kappa} = h_{n-1,n-1} + h_{n,n} = \mathrm{tr}\begin{bmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{bmatrix} \tag{5.3}$$

and

$$d = |\kappa|^2 = \kappa\overline{\kappa} = h_{n-1,n-1}h_{n,n} - h_{n-1,n}h_{n,n-1} = \det\begin{bmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{bmatrix}. \tag{5.4}$$

Now, as H is upper Hessenberg only the first three components of the first column of H^2 are non zero:

$$\begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \\ 0 & h_{32} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{21} \end{bmatrix} = \begin{bmatrix} h_{11}^2 + h_{12}h_{21} \\ h_{21}(h_{11} + h_{22}) \\ h_{21}h_{32} \end{bmatrix}.$$

Then the first three components of the first column of C are

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} h_{11}^2 + h_{12}h_{21} - th_{11} + d \\ h_{21}(h_{11} + h_{22}) - th_{21} \\ h_{21}h_{32} \end{bmatrix}.$$

Insert (5.3) and (5.4) and develop the sums:

$$c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} h_{11}^2 + h_{12}h_{21} - (h_{n-1,n-1} + h_{nn})h_{11} + (h_{n-1,n-1}h_{nn} - h_{n-1,n}h_{n,n-1}) \\ h_{21}(h_{11} + h_{22}) - (h_{n-1,n-1} + h_{nn})h_{21} \\ h_{21}h_{32} \end{bmatrix} =$$

$$\begin{bmatrix} h_{11}^2 + h_{12}h_{21} - h_{n-1,n-1}h_{11} - h_{nn}h_{11} + h_{n-1,n-1}h_{nn} - h_{n-1,n}h_{n,n-2} \\ h_{21}(h_{11} + h_{22} - h_{n-2,n-1} + h_{nn} + h_{11} - h_{11}) \\ h_{21}h_{32} \end{bmatrix} =$$

$$\begin{bmatrix} h_{21}\Big((h_{11}^2 - h_{n-1,n-1}h_{11} - h_{nn}h_{11} + h_{n-1,n-1}h_{nn} - h_{n-1,n}h_{n,n-2}) + h_{12}\Big) \\ h_{21}[(h_{22} - h_{11}) - (h_{nn} - h_{11}) - (h_{n-1,n-1} - h_{11})] \\ h_{21}h_{32} \end{bmatrix} =$$

$$h_{21}\begin{bmatrix} [(h_{nn} - h_{11})(h_{n-1,n-1} - h_{11}) - h_{n-1,n}h_{n,n-2}] + h_{12} \\ (h_{22} - h_{11}) - (h_{nn} - h_{11}) - (h_{n-1,n-1} - h_{11}) \\ h_{32} \end{bmatrix}.$$

That is,

$$c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = h_{21}\begin{bmatrix} [(h_{nn} - h_{11})(h_{n-1,n-1} - h_{11}) - h_{n-1,n}h_{n,n-2}] + h_{12} \\ (h_{22} - h_{11}) - (h_{nn} - h_{11}) - (h_{n-1,n-1} - h_{11}) \\ h_{32} \end{bmatrix}.$$

Two proportional vectors determine the same Householder transformation so h_{21} can be ignored in the definition of c, hence the working Algorithm (14) ignores it. (The same goes for the norm of the first column of C that must have been dividing c.) Input elements are scaled to avoid underflow or overflow on the computation of the element c_1. The last row computes the Householder transformation as that is what we set out for. The MATLAB implementation is in A.4.2.

Input : elements h_{11}, h_{12}, h_{21}, h_{22}, h_{32}, $h_{n_1n_1}$, h_{n_1n}, h_{nn_1}, h_{nn}.
Output: vector u.

startqr2step $(h_{11}, h_{12}, h_{21}, h_{22}, h_{32}, h_{n_1n_1}, h_{n_1n}, h_{nn_1}, h_{nn})$
  $s \leftarrow {}^{1}/\max\{|h_{11}|,|h_{12}|,|h_{21}|,|h_{22}|,|h_{32}|,|h_{n_1n_1}|,|h_{n_1n}|,|h_{nn_1}|,|h_{nn}|\}$
  $h_{11} \leftarrow sh_{11}$
  $h_{12} \leftarrow sh_{12}$
  $h_{21} \leftarrow sh_{21}$
  $h_{22} \leftarrow sh_{22}$
  $h_{32} \leftarrow sh_{32}$
  $h_{n_1n_1} \leftarrow sh_{n_1n_1}$
  $h_{n_1n} \leftarrow sh_{n_1n}$
  $h_{nn_1} \leftarrow sh_{nn_1}$
  $h_{nn} \leftarrow sh_{nn}$
  $p \leftarrow h_{nn} - h_{11}$
  $q \leftarrow h_{n_1n_1} - h_11$
  $r \leftarrow h_{22} - h_{11}$
  $c \leftarrow \begin{bmatrix} {}^{(pq-h_{nn_1}h_{n_1n})}/_{h_{21}} + h_{12} \\ r - p - q \\ h_{32} \end{bmatrix}$
  $\{u, \nu\} \leftarrow \text{housegen}(c)$
end

Algorithm 14: The start of an implicit QR double shift

5.4 Bulge chasing

The use of the implicit double shift creates a protrusion in the matrix. Let R_0 denote the Householder transformation corresponding to the vector returned by Algorithm 14. Premultiplication by R_0 acts only on the first three rows and postmultiplication on the first three columns. Figure 5.1 illustrates this transformation for a matrix of dimension $n = 5$.

$$H = \begin{bmatrix} \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & 0 & 0 & \mathrm{X} & \mathrm{X} \end{bmatrix} \xrightarrow{R_0 H R_0} \begin{bmatrix} \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ 0 & 0 & 0 & \mathrm{X} & \mathrm{X} \end{bmatrix} = H_0$$

Figure 5.1: Effect of the Householder transformation corresponding to the vector u returned by Algorithm 14.

This matrix has to be converted back to Hessenberg form and that is achieved by *chasing the bulge* down the diagonal. Following with the ex-

ample started in Figure 5.1, first an elementary reflector R_1 is applied. This Householder transformation affects rows and columns two through four. Check Figure 5.2.

$$H_0 = \begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \\ X & X & X & X & X \\ 0 & 0 & 0 & X & X \end{bmatrix} \xrightarrow{R_1 H_0 R_1} \begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \end{bmatrix} = H_1$$

Figure 5.2: First step of the bulge chasing.

Now, in Figure 5.3, an elementary reflector that eliminates the unwanted zeros on the second column is applied.

$$H_1 = \begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \end{bmatrix} \xrightarrow{R_2 H_1 R_2} \begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & 0 & X & X & X \\ 0 & 0 & X & X & X \end{bmatrix} = H_2$$

Figure 5.3

This last element can be annihilated by either a Householder transformation or a plane rotation R_3. This is because plane rotations can only introduce one zero at a time. In the previous steps we wanted to introduce more than one zero so the application of Givens rotations was out of the picture. Note that plane rotations are not simetric matrices, then if a Givens rotation is applied instead of a Householder transformation, the matrix that it is postmultiplied has to be the transpose of the one premultiplied.

$$H_2 = \begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & 0 & X & X & X \\ 0 & 0 & X & X & X \end{bmatrix} \xrightarrow{R_3 H_2 R_3^t} \begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & 0 & X & X & X \\ 0 & 0 & 0 & X & X \end{bmatrix} = H_3$$

Algorithm 15 implements this process. It takes a Hessenberg matrix H, the vector u resulting from Algorithm 14, and indices i_1 and i_2 −between which the routine is applied in the matrix H. The result is overwritten in H and transformations are accumulated in Q. The algorithm runs in $\mathcal{O}(n^2(i_2 - i_1))$ and the MATLAB code is in A.4.3.

Input : Hessenberg matrix H, vector u, indices i_1 and i_2.
Output: matrix H and matrix Q.

qr2step (H, u, i_1, i_2)
  $Q \leftarrow I$
  for $i \in \{i_1, ..., i_2 - 1\}$ **do**
    $j \leftarrow \max\{i_1 - 1, i_1\}$
    $v \leftarrow u^t \cdot H_{\{i,...,i+2\},\{j,...,n\}}$
    $H_{\{i,...,i+2\},\{j,...,n\}} \leftarrow H_{\{i,...,i+2\},\{j,...,n\}} - uv$
    $iu \leftarrow \min\{i + 3, i2\}$
    $v \leftarrow H_{\{1,...,iu\},\{i,...,i+2\}} \cdot u$
    $H_{\{1,...,iu\},\{i,...,i+2\}} \leftarrow H_{\{1,...,iu\},\{i,...,i+2\}} - vu^t$
    $v \leftarrow Q_{\{1,...,n\},\{i,...,i+2\}} - vu^t$
    $Q_{\{1,...,n\},\{i,...,i+2\}} \leftarrow Q_{\{1,...,n\},\{i,...,i+2\}} - vu^t$
    if $i \neq i_2 - 2$ **then**
      $\{u, \nu\} \leftarrow \text{housegen}(H_{\{i+1,...,i+3\},\{i\}})$
    end
    if $i \neq i_1$ **then**
      $H_{i+1,j} \leftarrow 0$
      $H_{i+2,j} \leftarrow 0$
    end
  end
  $\{H_{i_2-1,i_2-2}, H_{i_2,i_2-2}, c, s\} \leftarrow \text{rotgen}(H_{i_2-1,i_2-2}, H_{i_2,i_2-2})$
  $\{H_{\{i_2-1\},\{i_2-1,...,n\}}, H_{\{i_2\},\{i_2-1,...,n\}}\} \leftarrow$
   $\text{rotapp}(c, s, H_{\{i_2-1\},\{i_2-1,...,n\}}, H_{\{i_2\},\{i_2-1,...,n\}})$
  $\{H_{\{1,...,i_2\},\{i_2-1\}}, H_{\{1,...,i_2\},\{i_2\}}\} \leftarrow$
   $\text{rotapp}(c, s, H_{\{1,...,i_2\},\{i_2-1\}}, H_{\{1,...,i_2\},\{i_2\}})$
  $\{Q_{\{1,...,n\},\{i_2-1\}}, Q_{\{1,...,n\},\{i_2\}}\} \leftarrow$
   $\text{rotapp}(c, s, Q_{\{1,...,n\},\{i_2-1\}}, Q_{\{1,...,n\},\{i_2\}})$
end

Algorithm 15: The doubly shifted QR step.

5.5 A working implementation of the implicitly shifted QR algorithm

Algorithm 16 computes the real Schur form of any given Hessenberg matrix. The MATLAB implementation can be found in A.4.4.

Remark 5.5.1. To follow up on Remark 3.2.2 and Remark 4.4.3, the implicit QR algorithm does not converge for the matrix

$$Y = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Feel free to try it in the present implementation of the implicitly shifted QR iteration. Again, real-life implementations use ad-hoc shifts every few iterations if the routine fails to converge.

Input : real upper Hessenberg matrix H, maximum number of iterations $maxiter$.
Output: real Schur form T, transformation matrix Q

hqr2
  $i_1 \leftarrow 1$
  $i_2 \leftarrow n$
  $iter \leftarrow 0$
  while *true* **do**
    if $iter > itermax$ **then**
      error
    end
    $oldi_2 \leftarrow i_2$
    $\{i_1, i_2\} \leftarrow$ backsearch2(H, i_2)
    if $i_2 = 1$ **then**
      return
    end
    if $i_2 = oldi_2$ **then**
      $iter \leftarrow iter + 1$
    end
    else
      $iter \leftarrow 0$
    end
    $u \leftarrow$ startqr2step$(h_{i_1,i_1}, h_{i_1,i_1+1}, h_{i_1+1,i_1}, h_{i_1+1,i_1+1}, h_{i_1+2,i_1+1}, h_{i_2-1,i_2-1}, h_{i_2-1,i_2}, h_{i_2,i_2-1}, h_{i_2,i_2},)$
    $\{H, Q\} \leftarrow$ qr2step(H, u, i_1, i_2)
  end
end

Algorithm 16: Real Schur form of a real upper Hessenberg matrix.

Some observations about it:

- Assuming the real Schur forms needs k iterations to compute an eigenvalue and that $i_1 = 1$ throughout the reduction, then it consumes $2kn^3$ floating point multiplications and $2kn^3$ floating point additions to compute all of them. On the other hand, assuming that calculating each eigenvalue takes k' iterations, the complex Schur form will need $12k'n^3$ floating point multiplications plus $8k'n^3$ floating point additions. The Real Schur form is cheaper to compute.

- The double shift algorithm is numerically −and backwards− stable in the usual sense, very much like the single shift algorithm.

- Although both routines are backwards stable they do not esentially calculate the same matrices. While the eigenvalues of the real Schur form occur in conjugate pairs, those of the complex form can drift away from conjugacy. What is more, if the eigenvalue is ill conditioned, the conjugate eigenvalues can drift far away from conjugacy.

5.6 Eigenvectors of the real Schur form

As in §4.5, if $A = QTQ^*$ is the real Schur decomposition of A and Y is the matrix of right eigenvectors of T, then the matrix of right eigenvectors of A is QY. We might want to use Algorithm 12 to compute the right eigenvectors of the real Schur form T, but this form, as opposed to the complex Schur one, has 2×2 blocks on the diagonal. Therefore Algorithm 12 has to be modified. This section contains the theoretical background we developed to obtain a generalization of Algorithm 12.

In order to do this, we have to consider two distinct cases: the case of a real eigenvalue and the case of complex conjugate eigenvalues. Let us start with the first one. There are two cases to consider if the eigenvalue we are going to compute is simple: the eigenvalue located just above in the diagonal is also simple or the eigenvalues above are complex conjugate. Let us begin with the first case. Suppose T has the form

$$\begin{bmatrix} T_{11} & t_{12} & t_{13} \\ 0 & \tau_{22} & \tau_{23} \\ 0 & 0 & \tau_{33} \end{bmatrix},$$

where neither τ_{22} nor τ_{33} is an eigenvalue of T_{11}. Then we seek the eigenvector corresponding to $\lambda = \tau_{33}$ in the form

$$\begin{bmatrix} x_1 \\ \xi_2 \\ 1 \end{bmatrix}.$$

Thus we have the equation

$$\begin{bmatrix} T_{11} & t_{12} & t_{13} \\ 0 & \tau_{22} & \tau_{23} \\ 0 & 0 & \tau_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ \xi_2 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 \\ \xi_2 \\ 1 \end{bmatrix} \lambda. \tag{5.5}$$

(The reason for putting λ on the right of the eigenvector will become clear on the case of complex conjugate eigenvalues.) The second row of (5.5) yields

$$\tau_{22}\xi_2 + \tau_{23} = \xi_2\lambda \Longrightarrow \xi_2 = \frac{\tau_{23}}{\lambda - \tau_{22}}.$$

And the first row

$$T_{11}x_1 + t_{12}\xi_2 + t_{13} = x_1\lambda \Longrightarrow (T_{11} - \lambda I)x_1 = -t_{12}\xi_2 - t_{13},$$

hence we may obtain x_1 by solving the 2×2 system $(T_{11}-\lambda I)x_1 = -t_{12}\xi_2 - t_{13}$ and ξ_2 by the usual back-substitution.

Let us now assume that T has the form

$$T = \begin{bmatrix} T_{11} & T_{12} & t_{13} \\ 0 & T_{22} & t_{23} \\ 0 & 0 & \tau_{33} \end{bmatrix},$$

where τ_{33} is not an eigenvalue of neither T_{11} or T_{22}, and T_{11} and T_{22} do not share any eigenvalue. Thus, we seek eigenvectors of the form

$$\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix},$$

that is,

$$\begin{bmatrix} T_{11} & T_{12} & t_{13} \\ 0 & T_{22} & t_{23} \\ 0 & 0 & \tau_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} \lambda, \tag{5.6}$$

where $\lambda = \tau_{33}$. From the second row we deduce that

$$T_{22}x_2 + t_{23} = \lambda x_2 \Longrightarrow (T_{22} - \lambda I)x_2 = -t_{23}.$$

Notice that $(T_{22} - \lambda I)$ is not singular due to τ_{33} not being an eigenvalue of T_{22}. Now, from the first row of (5.6),

$$T_{11}x_1 + T_{12}x_2 + t_{13} = \lambda x_1 \Longrightarrow (T_{11} - \lambda I)x_1 = -T_{12}x_2 - t_{13},$$

where $(T_{11} - \lambda I)$ is not singular because τ_{33} is not an eigenvalue of T_{11}. Therefore, we may obtain both x_1 and x_2 solving linear systems.

For the case of complex conjugate eigenvalues, we split the study into two cases: the eigenvalue above is simple or the eigenvalues above are complex conjugate. For the first one, let us assume that T has the form

$$T = \begin{bmatrix} T_{11} & t_{12} & T_{13} \\ 0 & \tau_{22} & t_{23}^* \\ 0 & 0 & T_{33} \end{bmatrix},$$

where τ_{22} is not an eigenvalue of neither T_{11} and T_{33}, and T_{33} and T_{11} do not share any eigenvalue. Thus, instead of looking for an eigenvector, we look for an *eigenbasis*: a matrix of two columns of the form

$$\begin{bmatrix} X_1 \\ x_2^* \\ X_3 \end{bmatrix}.$$

The linear subspace spanned by the columns of this matrix contains the real and complex parts of the eigenvectors of T_{33} −remember that the 2×2 blocks located in the diagonal of a real Schur form contain complex conjugate eigenvalues. Now, in the same way that we have generalized an eigenvector we have to generalize the eigenvalue. This will be achieved by taking a matrix L of order two. Thus, the eigenvalue problem has been generalized to

$$\begin{bmatrix} T_{11} & t_{12} & T_{13} \\ 0 & \tau_{22} & t_{23}^* \\ 0 & 0 & T_{33} \end{bmatrix} \begin{bmatrix} X_1 \\ x_2^* \\ X_3 \end{bmatrix} = \begin{bmatrix} X_1 \\ x_2^* \\ X_3 \end{bmatrix} L. \tag{5.7}$$

From the third row of (5.7) we deduce

$$T_{33}X_3 = X_3L \Longrightarrow L = X_3^{-1}T_{33}X_3,$$

i.e., L and T_{33} are similar and thus they have the same eigenvalues. The second row yields

$$\tau_{22}x_2^* + T_{33}X_3 = x_2^*L \Longrightarrow x_2^*(L - \tau_{22}I) = t_{23}^*X_3,$$

where $(L - \tau_{22}I)$ is not singular because, by hypothesis, τ_{22} is not an eigenvalue of T_{33} and thus it is not an eigenvalue of L. The first row of (5.7) yields

$$T_{11}X_1 + t_{12}x_2^* + T_{13}X_3 = X_1L \Longrightarrow T_{11}X_1 - X_1L = -T_{13}X_3 - t_{12}x_2^*,$$

which is a solvable Sylvester's equation due to T_{11} and L having no common eigenvalues −check Theorem 1.16 on [1]. (Algorithm 1.1 on [1] gives an implementation that solves Sylvester's equations, but we will use the command `sylvester` available on MATLAB.) Therefore we may obtain X_3 by solving a 2×2 system and X_1 by solving a Sylvester's equation.

On the other hand, if the eigenvalues just above happen to be complex conjugate then T has the form

$$T = \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ 0 & T_{22} & T_{23} \\ 0 & 0 & T_{33} \end{bmatrix},$$

where T_{11} and T_{33} do not share any eigenvalues and neither do T_{22} and T_{33}. So, we seek an eigenbasis of the form

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix},$$

where X_3 is nonsingular, i.e.,

$$\begin{bmatrix} T_{11} & T_{12} & T_{13} \\ 0 & T_{22} & T_{23} \\ 0 & 0 & T_{33} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} L. \tag{5.8}$$

Take X_3 arbitrary but nonsingular. The third row yields

$$T_{33}X_3 = X_3L \Longrightarrow L = X_3^{-1}T_{33}X_3.$$

(Notice that L and T_{33} are similar and thus they share eigenvalues.) From the second row of (5.8),

$$T_{22}X_2 + T_{33}X_3 = X_2L \Longrightarrow T_{22}X_2 - X_2L = -T_{33}X_3,$$

is a solvable Sylvester's equation for X_2 because T_{22} and T_{33} do not share eigenvalues. And from the first row,

$$T_{11}X_1 + T_{12}X_2 + T_{13}X_3 = X_1L \Longrightarrow T_{11}X_1 - X_1L = -T_{12}X_2 - T_{13}X_3,$$

another solvable Sylvester's equation for X_1, due to T_{11} and T_{33} not having common eigenvalues.

Now, looking back at (5.7) and (5.8), what about X_3? How may we chose it? A natural choice will be $X_3 = I$, which is a generalization of $\xi_3 = 1$ and allows to form L without any computations. But chosing $X_3 = \begin{bmatrix} y_3 & z_3 \end{bmatrix}$, where $x_3 = y_3 + iz_3$ is the right eigenvector of T_{33}, turns out to be more convenient. Arguing as in Theorem 5.1.1, let

$$L = \begin{bmatrix} \mu & \nu \\ -\nu & \mu \end{bmatrix},$$

where $\mu \pm i\nu$ are the eigenvalues of T_{33}. Now, by partitioning $X = \begin{bmatrix} y & z \end{bmatrix}$, then

$$TX = XL \text{ or } T\begin{bmatrix} y & z \end{bmatrix} = \begin{bmatrix} y & z \end{bmatrix}\begin{bmatrix} \mu & \nu \\ -\nu & \mu \end{bmatrix}.$$

This means that $y \pm iz$ are the right eigenvectors of T corresponding to the eigenvalues $\mu \pm i\nu$. Thus this choice generates the real and imaginary parts of the desired eigenvector.

In conclussion, the algorithm that computes the right eigenvectors of a real Schur form T has to walk the diagonal starting from the southeast and finishing in the northeastest element. For each element of the diagonal it must be checked whether it is a simple eigenvalue or a 2×2 block and compute accordingly. We are not done though. When implementing the algorithm to compute the eigenbases of a real Schur matrix T some special cases have to be considered. They are discussed below.

The first one: all the eigenvectors and eigenbases have been calculated and the only remaining eigenvector corresponds to a simple eigenvalue. Partition T the following way,

$$\begin{bmatrix} \tau_{11} & t_{12}^* \\ 0 & T_{22} \end{bmatrix},$$

for which we seek an eigenvector of the form

$$\begin{bmatrix} \xi_1 \\ x_2 \end{bmatrix}.$$

That is,

$$\begin{bmatrix} \tau_{11} & t_{12}^* \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} \xi_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \xi_1 \\ x_2 \end{bmatrix} \lambda.$$

The second row yields

$$T_{22}x_2 = x_2\lambda \Longrightarrow T_{22}x_2 - x_2\lambda = 0$$

which is a Sylvester's equation with the solution $x_2 = 0$. Following with the first row

$$\tau_{11}\xi_1 + t_{12}^* x_2 = \xi_1\lambda \Longrightarrow \tau_{11}\xi_1 = \xi_1\lambda,$$

thus we may take any $\xi_1 \neq 0$. Then $\mathbf{e}_1$ is an eigenvector of T corresponding to the eigenvalue τ_{11}.

The second one: all the eigenvectors and eigenbases have been computed and the only remaining eigenbasis corresponds to two complex conjugate eigenvalues. Partitioning T in the form

$$\begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix},$$

where T_{11} is the 2×2 block containing the remaining eigenvalues. We seek an eigenbasis of the form

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix},$$

thus

$$\begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} L.$$

The second row yields

$$T_{22}X_2 = X_2L \Longrightarrow T_{22}X_2 - X_2L = 0,$$

which is a Sylvester's equation with the solution $X_2 = 0$. Therefore the first row gives

$$T_{11}X_1 + T_{22}X_2 = X_1L \Longrightarrow L = X_1^{-1}T_{11}X_1.$$

We will select X_1 following the result developed earlier in this section: it will contain, in columns, the real and imaginary parts of the eigenvectors of T_{11}.

The third and last one: all the eigenvectors and eigenbases have been computed, except for an eigenvector corresponding to a simple eigenvalue

and an eigenbasis corresponding to a 2×2 block. Now, partitioning T in the form

$$\begin{bmatrix} \tau_{11} & t_{12}^* & t_{13}^* \\ 0 & T_{22} & T_{23} \\ 0 & 0 & T_{33} \end{bmatrix}$$

we seek an eigenbasis of the form

$$\begin{bmatrix} \xi_1 \\ X_2 \\ X_3 \end{bmatrix},$$

i.e.,

$$\begin{bmatrix} \tau_{11} & t_{12}^* & t_{13}^* \\ 0 & T_{22} & T_{23} \\ 0 & 0 & T_{33} \end{bmatrix} \begin{bmatrix} \xi_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} \xi_1 \\ X_2 \\ X_3 \end{bmatrix} L.$$

The third row yields

$$T_{33}X_3 = X_3L \Longrightarrow T_{33}X_3 - X_3L = 0,$$

which is a Sylvester's equation with solution $X_3 = 0$. Thus the second row yields

$$T_{22}X_2 = X_2L \Longrightarrow L = X_2^{-1}T_{22}X_2,$$

and we will choose X_2 as usual: it will contain, in columns, the real and imaginary parts of the eigenvectors of T_{22}. Now, on the first row

$$\tau_{11}\xi_1 + t_{12}^*X_2 = \xi_1 L \Longrightarrow \xi_1(L - \tau_{11}I) = T_{12}^*X_2,$$

which is a linear system.

We now have the knowledge to implement a working algorithm that computes the eigenbases of a real Schur form. The routine we created does not fit in a single page, so it has been divided in Algorithm 17, Algorithm 18 and Algorithm 19. The MATLAB implementation is far from being perfect. It does take into account most of the computational aspects of the problem and thus computes correctly the eigenbases of most matrices −as it will be shown in Appendix B. Anyway, it overlooks both possible underflow and overflows, so for some specific matrices the imaginary part of some complex eigenvalues may underflow. An example of such a matrix is $A = PDP^{-1}$ where P is any nonsingular matrix of order six and

$$D = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 100\epsilon_M & 0 & 0 \\ 0 & 0 & -100\epsilon_M & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}.$$

Calculating the relative error of the eigenbases results in an error, as MATLAB tries to divide 0/0.

This algorithm is stable in the same sense as (4.6).

Input : complex Schur form matrix T of order n.
Output: matrix Y containing, in columns, the normalized right eigenvectors of T.

hqr2 (T)
  $j \leftarrow n$
  while $j \neq 0$ **do**
    if $j = 1$ **then**
      $y_{11} \leftarrow 1$
      $j \leftarrow j - 1$
    end
    else if $t_{21} \neq 0$ *and* $j = 2$ **then**
      $y_{\{1,2\},\{1,2\}} \leftarrow I$
      $j \leftarrow j - 2$
    end
    if $t_{j,j-1} = 0$ *and* $j > 1$ **then**
      if $j = 2$ **then**
        $\lambda \leftarrow t_{j,j}$
        $y_{jj} \leftarrow 1$
        $y_{j-1,j} \leftarrow t_{j-1,j}/(\lambda - t_{j-1,j-1})$
        $y_{\{1,\ldots,j\},j} \leftarrow y_{\{1,\ldots,j\},j} / \left\| y_{\{1,\ldots,j\},j} \right\|$
        $j \leftarrow j - 1$
      end
      else if $t_{j-1,j-2} = 0$ *and* $j > 2$ **then**
        $\lambda \leftarrow t_{jj}$
        $y_{jj} \leftarrow 1$
        $y_{j-1,j} \leftarrow t_{j-1,j}/(\lambda - t_{j-1,j-1})$
        Solve the system $(t_{\{1,\ldots,j-2\},\{1,\ldots,j-2\}} - \lambda I)x_1 = -t_{\{1,\ldots,j-2\},j-1} \cdot y_{j-1,j} - t_{\{1,\ldots,j-2\},j}$
        $y_{\{1,\ldots,j-2\},j} \leftarrow x_1$
        $y_{\{1,\ldots,j\},j} \leftarrow y_{\{1,\ldots,j\},j} / \left\| y_{\{1,\ldots,j\},j} \right\|$
        $j \leftarrow j - 1$
      end
      else if $j \geq 3$ **then**
        $\lambda \leftarrow t_{jj}$
        $y_{jj} \leftarrow 1$
        Solve the system $(t_{\{j-2,j-1\},\{j-2,j-1\}} - \lambda I)x_2 = -t_{\{j-2,j-1\},j}$
        $y_{\{j-2,j-1\},j} \leftarrow x_2$
        if $j > 3$ **then**
          Solve the system $(t_{\{1,\ldots,j-3\},\{1,\ldots,j-3\}} - \lambda I)x_1 = -t_{\{1,\ldots,j-3\},\{j-2,j-1\}} y_{\{j-2,j-1\},j} - t_{\{1,\ldots,j-3\},j}$
          $y_{\{1,\ldots,j-3\},j} \leftarrow x_1$
        end
        $y_{\{1,\ldots,j\},j} \leftarrow y_{\{1,\ldots,j\},j} / \left\| y_{\{1,\ldots,j\},j} \right\|$
        $j \leftarrow j - 1$
      end
    end
  end
end

Algorithm 17: Eigenbases of a real Schur form.

Input : complex Schur form matrix T of order n.
Output: matrix Y containing, in columns, the normalized right eigenvectors of T.

hqr2 (T)
  while $j \neq 0$ **do**
    /* The current eigenvalues are complex conjugates */
    else
      if $j = 3$ **then**
        $y_{11} \leftarrow 1$
        $Z \leftarrow$matrix of right eigenvectors of $t_{\{j-1,j\},\{j-1,j\}}$
        $L \leftarrow Z^{-1} \cdot t_{\{j-1,j\},\{j-1,j\}} \cdot Z$
        $y_{\{2,\ldots,j\},\{2,\ldots,j\}} \leftarrow \left[\mathrm{Re}(Z_{\{1,\ldots,n\},1}) \quad \mathrm{Im}(Z_{\{1,\ldots,n\},1})\right]$
        $y_{\{1,\ldots,n\},\{j-1,j\}} \leftarrow y_{\{1,\ldots,n\},\{j-1,j\}} / \left\| y_{\{1,\ldots,n\},\{j-1,j\}} \right\|$
        $j \leftarrow j - 3$
      end
      else if $t_{j-2,j-3} = 0$ *and* $j > 3$ **then**
        $y_{\{j-1,j\},\{j-1,j\}} \leftarrow I$
        $L \leftarrow t_{\{j-1,j\},\{j-1,j\}}$
        Solve the system $x_2^*(L - t_{j-2,j-2}I) = (t_{j-2,\{j-1,j\}})^* I$
        $y_{j-2,\{j-1,j\}} \leftarrow x_2$
        Solve Sylvester's equation
        $t_{\{1,\ldots,j-3\},\{1,\ldots,j-3\}}X_1 - X_1 L =$
        $-t_{\{1,\ldots,j-3\},\{j-1,j\}}I - t_{\{1,\ldots,j-3\},j-2}x_2^*$
        $y_{\{1,\ldots,j-3\},\{j-1,j\}} \leftarrow X_1$
        $y_{\{1,\ldots,j\},\{j-1,j\}} \leftarrow y_{\{1,\ldots,j\},\{j-1,j\}} / \left\| y_{\{1,\ldots,j\},\{j-1,j\}} \right\|$
        $j \leftarrow j - 2$
      end
    end
  end
end

Algorithm 18: Eigenbases of a real Schur form, part 2.

Input : complex Schur form matrix T of order n.
Output: matrix Y containing, in columns, the normalized right eigenvectors of T.

hqr2 (T)
  while $j \neq 0$ **do**
    else
      else if $j \geq 4$ **then**
        if $j > 4$ **then**
          $y_{\{j-1,j\},\{j-1,j\}} \leftarrow I$
          $L \leftarrow t_{\{j-1,j\},\{j-1,j\}}$
          Solve Sylvester's equation $t_{\{j-3,j-2\},\{j-3,j-2\}}X_2 - X_2 L = -t_{\{j-3,j-2\},\{j-1,j\}}I$
          $y_{\{j-3,j-2\},\{j-1,j\}} \leftarrow X_2$
          Solve Sylvester's equation $t_{\{1,\ldots,j-4\},\{1,\ldots,j-4\}}X_3 - X_3 L = -t_{\{1,\ldots,j-4\},\{j-3,j-2\}}X_2 - t_{\{1,\ldots,j-4\},\{j-1,j\}}$
          $y_{\{1,\ldots,j-4\},\{j-1,j\}} \leftarrow X_3$
        end
        else
          $y_{\{3,4\},\{3,4\}} \leftarrow I$
          Solve Sylvester's equation $t_{\{1,2\},\{1,2\}}X_4 - X_4 t_{\{3,4\},\{3,4\}} = -t_{\{1,2\},\{3,4\}}$
          $y_{\{1,2\},\{3.4\}} \leftarrow X_4$
        end
        $y_{\{1,\ldots,j\},\{j-1,j\}} \leftarrow y_{\{1,\ldots,j\},\{j-1,j\}} / \left\| y_{\{1,\ldots,j\},\{j-1,j\}} \right\|$
        $j \leftarrow j - 2$
      end
    end
  end
end

Algorithm 19: Eigenbases of a real Schur form, part 3.

Appendix A

Implementation in MATLAB

As the title says, this appendix contains all MATLAB implementations of the algorithms described on the work. The code has been ordered in the same structure of chapters, sections and subsections as above. It has also been uploaded to the following GitHub repository: https://github.com/gorkaerana/Bachelors-degree-dissertation.

A.1 The power and inverse power methods

A.1.1 The power method

PowerMethod function

```
1  function [ lambda, z ] = PowerMethod ( A, kappa,
       epsilon, x, MaxIter )
2  %GORKA ERAA ROBLES - This function is an
       implementation of the shifted
3  %power method.
4  %   It follows the ideas developed in 2.1.
5
6  Anorm = 1/norm(A,'fro');
7  xnorm = 1/norm(x);
8  x = x*xnorm;
9
10 for i = 1:MaxIter
11     y = A*x;
12     mu = (x')*y;
13     r = y - mu*x;
14     x = y - kappa*x;
15     xnorm = 1/norm(x);
16     x = x*xnorm;
17     if (norm(r)*Anorm < epsilon)
```

```
        lambda = mu;
        z = x;
        return
    end
end

error('Maximum number of iterations exceeded; increase
    options MaxIter.');

end
```

A.1.2 The inverse power method

InversePowerMethod function

```
function [ lambda, z ] = InversePowerMethod ( A, kappa
   , epsilon, x, MaxIter )
%GORKA ERAA ROBLES - This function is an
   implementation of the inverse
%power iteration using the Rayleigh quotient method.
%   It follows the ideas developed in 2.2.

[m,~] = size(A);

Anorm = 1/norm(A,'fro');

for i = 1:MaxIter
    y = (A - kappa*eye(m))\x;
    ynorm = 1/norm(y);
    x1 = y*ynorm;
    w = x*ynorm;
    ro = (x1')*w;
    mu = kappa + ro;
    r = w - ro*x1;
    x = x1;
    kappa = mu;
    if (norm(r)*Anorm <= epsilon)
        lambda = mu;
        z = x;
        return
    end
end

```

```
27 error('Maximum number of iterations exceeded; increase
       option MaxIter.')
28
29 end
```

A.2 First steps towards an efficient QR algorithm

A.2.1 Making the QR iteration practical: the Hessenberg form

Housegen function

```
1  function [ u,nu ] = housegen( a )
2  %GORKA ERAA ROBLES - This function computes a vector
       u that generates a
3  %Householder reflection H = I - uu* satisfying Ha =
       nue1.
4  %   It follows the ideas developed in 3.4.1.
5  %   Accumulating this transformations any matrix can
       be reduced to upper
6  %   Hessenberg form.
7
8  u = a;
9  nu = norm(a);
10 if ( nu==0 )
11     u(1) = sqrt(2);
12     return
13 end
14 if ( u(1)~=0 )
15     rho = (u(1)')/norm(u(1));
16 else
17     rho = 1;
18 end
19
20 u = (rho/nu)*u;
21 u(1) = 1 + u(1);
22 u = u/sqrt(u(1));
23 nu = -(rho')*nu;
24
25 end
```

Hessreduce function

```
function [ H,Q ] = hessreduce( A )
%GORKA ERAA ROBLES - This algorithm reduces a matrix
   A of order n to upper
%Hessenberg form by Householder transformations.
%   It follows the ideas developed in 3.4.2.

[n,~] = size(A);
H = A;
Q = eye(n);

for k = 1:n-2
    % Householder transformation "for each column/row"
    [u,H(k+1,k)] = housegen(H(k+1:n,k));
    Q(k+1:n,k) = u;

    % Multiply transformations on left
    v = (u')*H(k+1:n,k+1:n);
    H(k+1:n,k+1:n) = H(k+1:n,k+1:n) - u*v;
    H(k+2:n,k) = 0;

    % Multiply transformations on right
    v = H(1:n,k+1:n)*u;
    H(1:n,k+1:n) = H(1:n,k+1:n) - v*(u');

end

% Accumulate transformations on matrix Q
I = eye(n);
for k = n-2:-1:1
    u = Q(k+1:n,k);
    v = (u')*Q(k+1:n,k+1:n);
    Q(k+1:n,k+1:n) = Q(k+1:n,k+1:n) - u*v;
    Q(:,k) = I(:,k);
end
```

A.3 The explicitly shifted QR algorithm

A.3.1 Negligible elements and deflation

Backsearch function

```
function [ i1,i2 ] = backsearch( H,z )
```

```
2  %GORKA ERAA ROBLES - This function finds deflating
       rows on a complex Schur
3  %form matrix.
4  %    It is based on the ideas developed in 4.2.
5  %    Input: Hessenberg matrix H of order n; index z (1
       < z <= n)
6  %    Output: indices i1 and i2 (i1, i2 <= z) holding
       one of the following
7  %    conditions:
8  %          1) 1 <= i1 < i2 <= z, deflate at rows i1 and
       i2.
9  %          2) 1 = i2 = i2, matrix is completely deflated
10
11 i1 = z;
12 i2 = z;
13 normH = norm(H,'fro');
14
15 while (i1 > 1)
16
17     if (abs(H(i1,i1-1)) < eps*normH)
18         H(i1,i1-1) = 0;
19         if (i1 == i2)
20             i2 = i1 - 1;
21             i1 = i1 - 1;
22         else
23             return
24         end
25     else
26         i1 = i1 - 1;
27     end
28
29 end
30
31 end
```

A.3.2 The Wilkinson shift

Wilkshift function

```
1  function [ kappa ] = wilkshift( a,b,c,d )
2  %GORKA ERAA ROBLES - This function computes the
       Wilkinson shift of a
3  %submatrix of order 2.
```

```
4  %   It is based on the ideas of 4.3.
5  %   Input: matrix B = ( a b )
6  %                     ( c d )
7  %   Output: shift kappa, nearest eigenvalue to d
8
9  kappa = d;
10 s = abs(a) + abs(b) + abs(c) + abs(d);
11
12 if (s == 0)
13     return
14 end
15
16 q = (b/s)*(c/s);
17
18 if (q ~= 0)
19     p = 0.5*((a/s) - (d/s));
20     r = sqrt(p*p + q);
21     if ( (real(p)*real(r) + imag(p)*imag(r)) < 0 )
22         r = -r;
23     end
24     kappa = kappa - s*(q/(p+r));
25 end
26
27 end
```

A.3.3 Implicit QR factorization and RQ product

Rotgen function

```
1  function [ a,b,c,s ] = rotgen( a,b )
2  %GORKA ERAA ROBLES - This function generates a Givens
       rotation from
3  %elements a and b. It is implemented in complex
       arithmetic.
4  %   It follows the ideas developed in 4.4.
5  %   Input: quantities a, b where (c   s ) (a) = (nu*a/
       abs(a))
6  %                                (-s' c') (b)   (
       0     )
7  %   Output: constants c and s; overwrites a with its
       final version and b with
8  %   zero
9
```

```
10 if ( b==0 )
11     c = 1;
12     s = 0;
13     return
14 end
15 if ( a==0 )
16     c = 0;
17     s = 1;
18     a = b;
19     b = 0;
20     return
21 end
22
23 mu = a/abs(a);
24 tau = abs(real(a)) + abs(imag(a)) + abs(real(b)) + abs
       (imag(b));
25 nu = tau*sqrt(abs(a/tau)^2 + abs(b/tau)^2);
26 c = abs(a)/nu;
27 s = mu*(b')/nu;
28 a = nu*mu;
29 b = 0;
30
31 end
```

Rotapp function

```
1  function [ x,y ] = rotapp( c,s,x,y )
2  %GORKA ERAA ROBLES - This function takes a plane
       rotation defined by c and
3  %s (the scalars returned by rotgen) and applies it to
       the vectors x and y.
4  %P (x^t). It is implemented in complex arithmetic.
5  %  (y^t)
6  %   It follows the ideas developed in 4.4.
7  %   Input: rotation matrix P = (c   s ); vectors x and
        y
8  %                              (-s' c')
9  %   Output: x and y overwritten with  P (x^t)
10 %                                      (y^t)
11
12 t = c*x + s*y;
13 y = c*y - (s')*x;
14 x = t;
15
```

```
16 end
```

hqr function

```
1  function [ H,Q ] = hqr( H,Q,maxiter )
2  %GORKA ERAA ROBLES - This routine overwrites H with a
       unitary similar
3  %triangular matrix whose diagonals are the eigenvalues
       of H (the real Schur
4  %form). Transformations are stored in Q.
5  %   It follows the ideas developed in 4.4.
6  %   Input: upper Hessenberg matrix H; number of
      maximum iterations maxiter
7  %   Output: Schur form overwritten in H; similarity
      transformation Q
8
9  [n,~] = size(H);
10 i2 = n;
11 iter = 0;
12 c = zeros(1,n);
13 s = zeros(1,n);
14
15 while 1
16     iter = iter + 1;
17
18     if (iter > maxiter) % Throws an error if maxiter
          is exceeded
19         error('Maximum number of iterations exceeded;
              increase option maxiter.')
20     end
21
22     oldi2 = i2;
23     [i1,i2] = backsearch(H,i2); % Check subdiagonal
          for near ceros, deflating points
24
25     if ( i2==1 ) % End the function if H is upper
          triangular
26         return
27     end
28     if ( i2~=oldi2 ) % Set iteration number to zero if
          there is another deflating row
29         iter = 0;
30     end
31
```

```
32      % Compute Wilkinson shift
33      kappa = wilkshift( H(i2-1,i2-1),H(i2-1,i2),H(i2,i2
           -1),H(i2,i2) );
34
35      H(i1,i1) = H(i1,i1) - kappa; % Apply shift to the
           element of the diagonal that is left out of the
            loop
36      for j = i1:i2-1 % Loop reducing the matrix to
           triangular form
37          [ H(j,j),H(j+1,j),c(j),s(j) ] = rotgen( H(j,j)
              ,H(j+1,j) ); % Apply rotation so that the
              subdiagonal is set to zero
38          H(j+1,j+1) = H(j+1,j+1) - kappa; % Apply shift
               to diagonal
39          [ H(j,j+1:n),H(j+1,j+1:n) ] = rotapp( c(j),s(j
              ),H(j,j+1:n),H(j+1,j+1:n) ); % Modify the
              involved rows
40      end
41
42      for k = i1:i2-1 % Loop applying the back
           multiplication
43          [ H(1:k+1,k),H(1:k+1,k+1) ] = rotapp( c(k),
              conj(s(k)),H(1:k+1,k),H(1:k+1,k+1) );
44          [ Q(1:n,k),Q(1:n,k+1) ] = rotapp( c(k),conj(s(
              k)),Q(1:n,k),Q(1:n,k+1) ); % Accumulate
              transformations
45          H(k,k) = H(k,k) + kappa;
46      end
47      H(i2,i2) = H(i2,i2) + kappa; %
48
49  end
```

A.3.4 Eigenvectors of the complex Schur form

Righteigvec function

```
1   function [ X ] = righteigvec( T )
2   %GORKA ERAA ROBLES - This routine computes, given an
       upper triangular
3   %matrix T, its right eigenvectors. They are stored in
       the matrix X by
4   %columns. They are normalized to have Frobenius norm
       one.
```

```
5  %   It follows the ideas developed in 4.5.
6  %   Input: upper triangular matrix T
7  %   Output: upper triangular matrix X of the right
       eigenvectors of T
8
9  [n,~] = size(T);
10 smallnum = (n/eps)*realmin;
11 bignum = (eps/n)*realmax;
12 X = zeros(n);
13
14 for k =n:-1:1
15     X(1:k-1,k) = -T(1:k-1,k);
16     X(k,k) = 1;
17     X(k+1:n,k) = 0;
18     dmin = max(eps*abs(T(k,k)),smallnum);
19     for j = k-1:-1:1
20         d = T(j,j) - T(k,k);
21         if ( abs(d) <= dmin )
22             d = dmin;
23         end
24         if ( abs(X(j,k))/bignum >= abs(d) )
25             s = abs(d)/abs(X(j,k));
26             X(1:k,k) = s*X(1:k,k);
27         end
28         X(j,k) = X(j,k)/d;
29         X(1:j-1,k) = X(1:j-1,k) - X(j,k)*T(1:j-1,j);
30     end
31     X(1:k,k) = X(1:k,k)/norm(X(1:k,k),'fro');
32 end
```

A.4 The implicitly shifted QR algorithm

A.4.1 Negligible 2×2 blocks and deflation

Backsearch2 function

```
1  function [ H,Q,i1,i2 ] = backsearch2( H,Q,z )
2  %GORKA ERAA ROBLES - This function finds for
       deflating rows on a real
3  %Schur form matrix.
4  %   It is based on the ideas developed in 5.2.
5
```

```
6  normH = norm(H,'fro');
7  i1 = z;
8  i2 = z;
9
10 while i1>1
11
12     if ( abs(H(i1,i1-1)) > eps*normH && i2 ~= 2 )
13         i1 = i1 - 1; % Reduce i1
14     else
15         if i2 ~= 2
16             H(i1,i1-1) = 0; % Deflate
17         end
18
19         % Check if it is a 1x1 or 2x2 block
20         if ( i1 == i2 - 1 || i2 == 2 )
21
22             % If it is a 2x2 block process it
23             [H,Q] = blockprocess(H,Q,i2);
24
25             if i2 ~= 2 % If it is a complex block go
                   to row i1-1
26                 i2 = i1 - 1;
27                 i1 = i1 - 1;
28
29             else % If i2==2 then we have reached the
                   firs 2x2 block
30                  i1 = 1;
31                  i2 = 1;
32             end
33
34         % If not, it is not a 2x2 block, it is a 1x1.
               Go to row i1-1 or
35         % break the loop.
36         elseif i1 == i2
37             i2 = i1 - 1;
38             i1 = i1 - 1;
39             else % Break the loop and finish the
                   function
40                  break
41         end
42     end
43 end
44
45 end
```

A.4.2 The double shift

Startqr2step function

```
function [ u ] = startqr2step( h11, h12, h21, h22, h32, hn1n1, hn1n, hnn1, hnn )
%GORKA ERAA ROBLES - This function returns the elements of the first
%column of H^2 - 2*Re(kappa)*H + |kappa|*I.
%   It is based on the ideas developed in 5.3.2.
%   Once it has computed the first three elements, it creates a vector u
%   that generates a Householder reflection so that the bulge chasing can
%   be applied to the rest of the matrix.

elements = [h11, h12, h21, h22, h32, hn1n1, hn1n, hnn1, hnn];
s = 1/max(abs(elements));
elements = s*elements;
p = elements(9) - elements(1);
q = elements(6) - elements(1);
r = elements(4) - elements(1);
c = [(p*q - elements(8)*elements(7))/elements(3) + elements(2); (r-p-q); elements(5)];
[u,~] = housegen(c);

end
```

A.4.3 Bulge chasing

Qr2step function

```
function [ H,Q ] = qr2step( H,Q,u,i1,i2 )
%GORKA ERAA ROBLES - This function, apllies the bulge chasing to a matrix
%H. It takes the vector u generated by startqr2step and the deflation rows
%given by backsearch2.
%   It follows the ideas developed in 5.4.
```

```
6
7  [n,~] = size(H);
8
9  for i = i1:i2-2
10
11     j = max([i-1,i1]);
12     v = transpose(u)*H(i:i+2,j:n);
13     H(i:i+2,j:n) = H(i:i+2,j:n) - u*v;
14     iu = min([i+3,i2]);
15     v = H(1:iu,i:i+2)*u;
16     H(1:iu,i:i+2) = H(1:iu,i:i+2) - v*transpose(u);
17     v = Q(1:n,i:i+2)*u;
18     Q(1:n,i:i+2) = Q(1:n,i:i+2) - v*transpose(u);
19
20     if ( i ~= (i2-2) )
21         [u,~] = housegen(H(i+1:i+3,i));
22     end
23     if ( i ~= i1 )
24         H(i+1,j) = 0;
25         H(i+2,j) = 0;
26     end
27
28 end
29
30 if ( i2 > 2 )
31     [H(i2-1,i2-2),H(i2,i2-2),c,s] = rotgen(H(i2-1,i2
          -2),H(i2,i2-2));
32     [H(i2-1,i2-1:n),H(i2,i2-1:n)] = rotapp(c,s,H(i2-1,
          i2-1:n),H(i2,i2-1:n));
33     [H(1:i2,i2-1),H(1:i2,i2)] = rotapp(c,s,H(1:i2,i2
          -1),H(1:i2,i2));
34     [Q(1:n,i2-1),Q(1:n,i2)] = rotapp(c,s,Q(1:n,i2-1),Q
          (1:n,i2));
35 end
36
37 end
```

A.4.4 A working implementation of the implicitly shifted QR algorithm

hqr2 function

```
1  function [ H,Q ] = hqr2( H,Q,maxiter )
```

```
%GORKA ERAA ROBLES - This function applies the implicitly shifted QR
%iteration to an upper Hessenberg form matrix H and applies the
%transformations to the matrix Q.
%   It follows the ideas developed in 5.5.

[n,~] = size(H);
i2 = n;
iter = 0;

while i2 > 1

    if ( iter > maxiter ) % Return an error if the routine does not converge.
        error('Maximum number of iterations exceeded; increase option maxiter.')
    end

    oldi2 = i2;
    [H,Q,i1,i2] = backsearch2(H,Q,i2); % Find deflation rows.


    if ( i2 == oldi2 ) % If it does not converge, sum one to the counter.
        iter = iter + 1;
    else % If it does converge set the counter to 0.
        iter = 0;
    end

    % Apply the bulge chasing if the matrix hast not been completely
    % deflated.
    if ( i2 > 1 )
        u = startqr2step(H(i1,i1),H(i1,i1+1),H(i1+1,i1),H(i1+1,i1+1),H(i1+2,i1+1),H(i2-1,i2-1),H(i2-1,i2),H(i2,i2-1),H(i2,i2));
        [H,Q]= qr2step(H,Q,u,i1,i2);
    end


end

```

```
37 end
```

A.4.5 Eigenvectors of the real Schur form

eigenbasis function

```
1  function [ Y ] = eigenbasis( T )
2  %GORKA ERAA ROBLES - This function computes the
       eigenbases of a real Schur
3  %form matrix T.
4  %   The implementation follows the computations
       developed in section 5.6 of
5  %   the dissertation.
6
7  [n,~] = size(T);
8  j = n;
9  Y = zeros(n);
10
11 while ( j ~= 0 )
12
13     if ( j == 1 ) % Base case number 1: the first
          eigenvalues is simple
14         Y(1,1) = 1;
15         j = j - 1;
16
17     elseif ( T(2,1) ~= 0 && j == 2 ) % Base case
          number 2: the first two eigenvalues are complex
           conjugate
18         [L,~] = complexschur(T(1:2,1:2));
19         L = [real(L(1,1)),imag(L(1,1));-imag(L(1,1)),
              real(L(1,1))];
20         Y(1:2,1:2) = L;
21 %         Y(1:2,1:2) = eye(2);
22         j = j - 2;
23
24     elseif ( T(j,j-1) == 0 && j > 1 ) % The current
          eigenvalue is simple
25
26         if ( j == 2 )
27             lambda = T(j,j);
28             Y(j,j) = 1;
29             Y(j-1,j) = T(j-1,j)/(lambda - T(j-1,j-1));
30             Y(1:j,j) = Y(1:j,j)/norm(Y(1:j,j),'fro');
```

```
31              j = j - 1;
32
33          elseif ( T(j-1,j-2) == 0 && j > 2 ) % The
                eigenvalue above is simple
34              lambda = T(j,j);
35              Y(j,j) = 1;
36              Y(j-1,j) = T(j-1,j)/(lambda - T(j-1,j-1));
37              Y(1:j-2,j) = (T(1:j-2,1:j-2)-lambda*eye(j
                    -2))\(-T(1:j-2,j-1)*Y(j-1,j)-T(1:j-2,j)
                    );
38              Y(1:j,j) = Y(1:j,j)/norm(Y(1:j,j),'fro');
39              j = j - 1;
40
41          elseif ( j >= 3 ) % The eigenvalues above are
                complex conjugate
42              lambda = T(j,j);
43              Y(j,j) = 1;
44              Y(j-2:j-1,j) = (T(j-2:j-1,j-2:j-1)-lambda*
                    eye(2))\(-T(j-2:j-1,j));
45              if ( j > 3 )
46                  Y(1:j-3,j) = (T(1:j-3,1:j-3)-lambda*
                        eye(j-3))\(-T(1:j-3,j-2:j-1)*Y(j-2:
                        j-1,j)-T(1:j-3,j));
47              end
48              Y(1:j,j) = Y(1:j,j)/norm(Y(1:j,j),'fro');
49              j = j - 1;
50
51          end
52
53      else % The current eigenvalues are complex
            conjugates
54
55          if ( j == 3 ) % Base case: the one above is
                the last simple eigenvalue
56              Y(1,1) = 1;
57              [L,Z] = complexschur(T(j-1:j,j-1:j));
58              X3 = righteigvec(L);
59              X3 = Z*X3;
60              L = [real(L(1,1)),imag(L(1,1));-imag(L
                    (1,1)),real(L(1,1))];
61              Y(2:j,2:j) = [real(X3(:,1)),imag(X3(:,1))
                    ];
62              Y(1,2:j) = T(1,2:j)*Y(2:j,2:j)/(L-T(1,1)*
                    eye(2));
```

```
63              Y(:,j-1:j) = Y(:,j-1:j)/norm(Y(:,j-1:j),'
                    fro');
64              j = j - 3;
65
66          elseif ( T(j-2,j-3) == 0 && j > 3 ) % The
                eigenvalue above is simple
67              [L,Q1] = complexschur(T(j-1:j,j-1:j));
68              X3 = righteigvec(L);
69              X3 = Q1*X3;
70              X3 = [real(X3(:,1)),imag(X3(:,1))]; % The
                    definitive X3
71              Y(j-1:j,j-1:j) = X3;
72              L = [real(L(1,1)),imag(L(1,1));-imag(L
                    (1,1)),real(L(1,1))]; % Instead of
                    computing X3^(-1)*T33*X3, we follow
                    theoretical results
73              Y(j-2,j-1:j) = (T(j-2,j-1:j)*X3)/(L - T(j
                    -2,j-2)*eye(2));
74              x2 = Y(j-2,j-1:j);
75              Y(1:j-3,j-1:j) = sylvester(T(1:j-3,1:j-3)
                    ,-L,-T(1:j-3,j-1:j)*X3-T(1:j-3,j-2)*x2)
                    ;
76              Y(:,j-1:j) = Y(:,j-1:j)/norm(Y(:,j-1:j),'
                    fro');
77              j = j - 2;
78
79          elseif ( j >= 4 ) % The eigenvalues above are
                complex conjugates
80              if ( j > 4 )
81                  [L,Q1] = complexschur(T(j-1:j,j-1:j));
82                  X3 = righteigvec(L);
83                  X3 = Q1*X3;
84                  X3 = [real(X3(:,1)),imag(X3(:,1))]; %
                        The definitive X3
85                  Y(j-1:j,j-1:j) = X3;
86                  L = [real(L(1,1)),imag(L(1,1));-imag(L
                        (1,1)),real(L(1,1))]; % Instead of
                        computing X3^(-1)*T33*X3, we follow
                         theoretical results
87                  Y(j-3:j-2,j-1:j) = sylvester(T(j-3:j
                        -2,j-3:j-2),-L,-T(j-3:j-2,j-1:j)*X3
                        );
88                  X2 = Y(j-3:j-2,j-1:j);
```

```
89                  Y(1:j-4,j-1:j) = sylvester(T(1:j-4,1:j
                       -4),-L,-T(1:j-4,j-3:j-2)*X2-T(1:j
                       -4,j-1:j)*X3);
90              else % Base case: both eigenvalues above
                    are complex conjugates
91                  Y(3:4,3:4) = eye(2);
92                  Y(1:2,3:4) = sylvester(T(1:2,1:2),-T
                       (3:4,3:4),-T(1:2,3:4));
93              end
94              Y(:,j-1:j) = Y(:,j-1:j)/norm(Y(:,j-1:j),'
                    fro');
95              j = j - 2;
96
97          end
98
99      end
100
101 end
102
103 end
```

Appendix B

Precision and accuracy analysis

Once the theoretical and technical details have been settled, it is time to check whether the implementations are accurate or not. Since this work is centered in the two variants of the QR algorithm, we will only check the implementation of the essentials: the Hessenberg reduction, the explicit QR algorithm and its pertinent eigenvectors, and the implicit QR iteration and its eigenvectors.

This addendum has the same composition as Appendix A: the tests have been ordered following the same structure of chapters, sections and subsections as the dissertation. Unless it is otherwise indicated, each analysis contains a brief explanation, a MATLAB script with the numerical simulations, and a graphic of those simulations. All the scripts can be found in the GitHub repository above mentioned: https://github.com/gorkaerana/Bachelors-degree-dissertation.

B.1 First steps towards an efficient QR algorithm

B.1.1 Making the QR iteration practical: the Hessenberg form

This subsection tests the accuracy of the Hessenberg reduction, which is explained in §3.4 and its implementation can be found in A.2.1. Let A be a matrix of order n and $T = Q^*AQ$ its Hessenberg reduction. The following analysis tests if

$$\frac{\|A - QTQ^*\|}{\|A\|} \leq \gamma\epsilon_M \tag{B.1}$$

for some reasonable constant γ. Running the following script

Hessenberg reduction test

```
% Script that computes the backwards error of
   calculating the upper
% Hessenbger form of a matrix.

sim = 1000;
for j = 1:sim
    n = randi([5,30]);
    A = exp(randn(n)*1i + randn(n));
    [H,Q] = hessreduce(A);
    semilogy(j,norm(A - Q*H*Q')/norm(A),'.');
    hold on
end
axis([1 sim eps 10^(-14)])
ylabel('\epsilon_M < y < 10^{-14}')
```

returns Figure B.1 −notice that the axis Y is in logarithmic scale and that $Y = 0$ corresponds to ϵ_M:

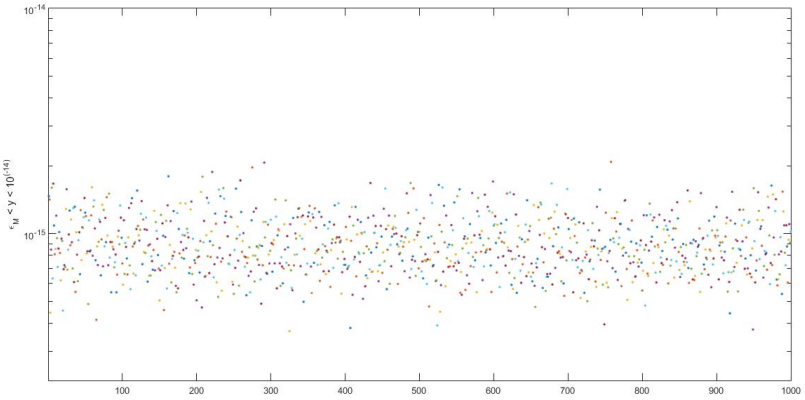

Figure B.1: Backwards error of the Hessenberg reduction.

That is, since in MATLAB $\epsilon_M = 2.2204 \cdot 10^{-16}$, for every complex matrix A, the backwards error of computing the Hessenberg reduction through the `hessreduce` function satisfies

$$\frac{\|A - QTQ^*\|}{\|A\|} \leq 50 \cdot \epsilon_M.$$

But are the matrices Q above unitary? Let us see:

Script to check if the matrices Q of the Hessenberg are unitary

```
% Script that computes how far is the transformation
   matrix Q from the
```

```
% reduction to Hessenberg form from being unitary.

sim = 1000;
for j = 1:sim
    n = randi([5,30]);
    A = exp(randn(n)*1i + randn(n));
    [H,Q] = hessreduce(A);
    semilogy(j,norm(eye(n) - Q*Q'),'.');
    semilogy(j,norm(eye(n) - Q'*Q),'o');
    hold on
end
axis([1 sim eps 10^(-14)])
ylabel('\epsilon_M < y < 10^{-14}')
```

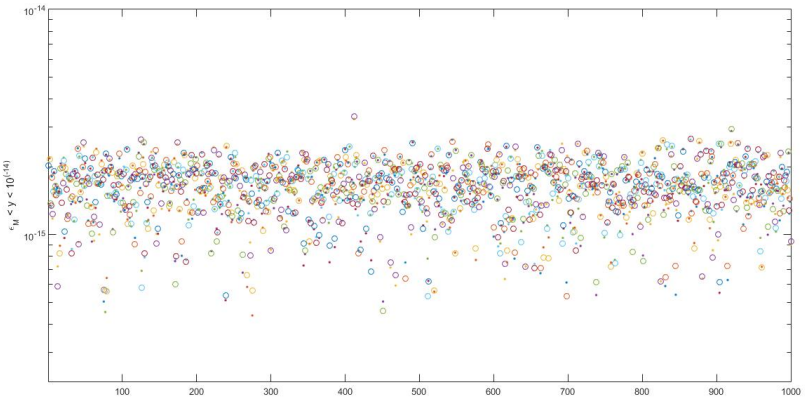

Figure B.2: Unitariness of the matrices Q.

Since

$$\|I - QQ^*\| \leq 50 \cdot \epsilon_M \text{ and } \|I - Q^*Q\| \leq 50 \cdot \epsilon_M,$$

we can say that they are.

B.2 The explicitly shifted QR algorithm

Now let us check on the explicitly shited QR algorithm −explained in §4 and implemented in MATLAB in A.3−. The following script

Complex Schur reduction test

```
% Script that computes the backwards error from
   calculating the complex
% Schur form of a matrix.
```

```
3
4  sim = 1000;
5  for j = 1:sim
6      n = randi([5,30]);
7      A = exp(randn(n)*1i + randn(n));
8      [T,Q] = complexschur(A);
9      semilogy(j,norm(A - Q*T*Q')/norm(A),'.');
10     hold on
11 end
12 axis([1 sim eps 10^(-14)])
13 ylabel('\epsilon_M < y < 10^{-14}')
```

computes the error (B.1), where A is our matrix, and $T = Q^*AQ$ is Schur form of A. Figure B.3 shows the computations.

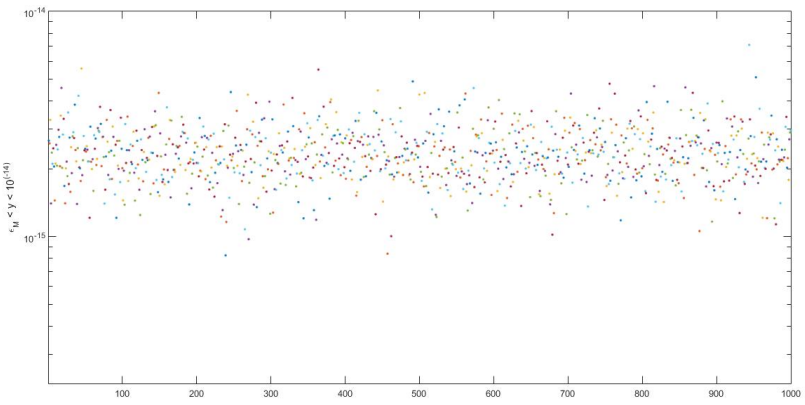

Figure B.3: Backwards error of the Schur reduction.

In this case, for every complex matrix A,

$$\frac{\|A - QTQ^*\|}{\|A\|} \leq 80 \cdot \epsilon_M.$$

As in the section above, are the matrices Q unitary?

Unitariness of the matrices Q of the Schur reduction

```
1  % Script that computes how far is the transformation
       matrix Q from the
2  % reduction to complex Schur form from being unitary.
3
4  sim = 1000;
5  for j = 1:sim
6      n = randi([5,30]);
```

```
    A = exp(randn(n)*1i + randn(n));
    [T,Q] = complexschur(A);
    semilogy(j,norm(eye(n) - Q*Q')/norm(A),'.');
    semilogy(j,norm(eye(n) - Q'*Q)/norm(A),'o')
    hold on
end
axis([1 sim eps 10^(-14)])
ylabel('\epsilon_M < y < 10^{-14}')
```

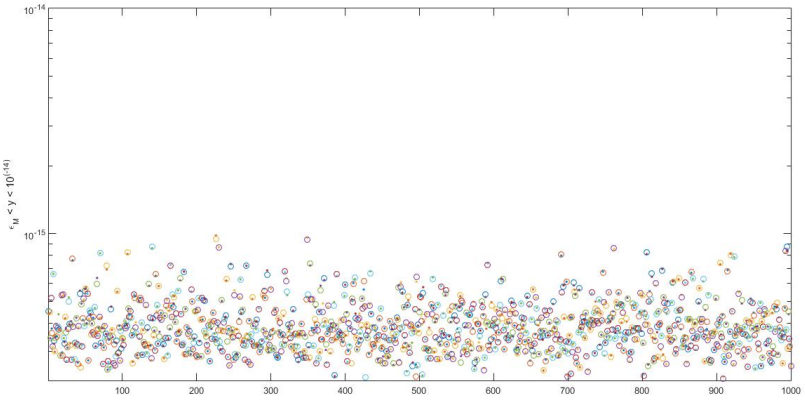

Figure B.4: Unitariness of the matrices Q of the Schur reduction.

As Figure B.4 shows, in this case, for every complex matrix A

$$\|A - QQ^*\| \leq 10 \cdot \epsilon_M \text{ and } \|A - Q^*Q\| \leq 10 \cdot \epsilon_M.$$

Now, since the Schur reduction algorithm works properly, let us see if the eigenvalues are computed correctly. The Schur decomposition of a matrix is not unique so the vector of the eigenvalues computed by MATLAB and the vector obtained by this implementation will not display the eigenvalues in the same order. Thus, instead of comparing each eigenvalue, we will compare the norms of the eigenvalue vectors to check if the computations are correct. This is what the following script does.

Eigenvalue comparison: MATLAB vs. our implementation

```
% Script that computes the backwards error calculating
    the eigenvalues
% using the explicitly shifted QR iteration.

sim = 1000;
for j = 1:sim
```

```
6       n = randi([5,30]);
7       A = exp(randn(n)*1i + randn(n));
8       [T,Q] = complexschur(A);
9       eigenval = abs(eig(A));
10      semilogy(j,norm(eigenval - abs(diag(T)))/norm(
            eigenval),'.');
11      hold on
12  end
13  axis([1 sim eps 10^(-14)])
14  ylabel('\epsilon_M < y < 10^{-14}')
```

And the result is displayed by Figure B.5.

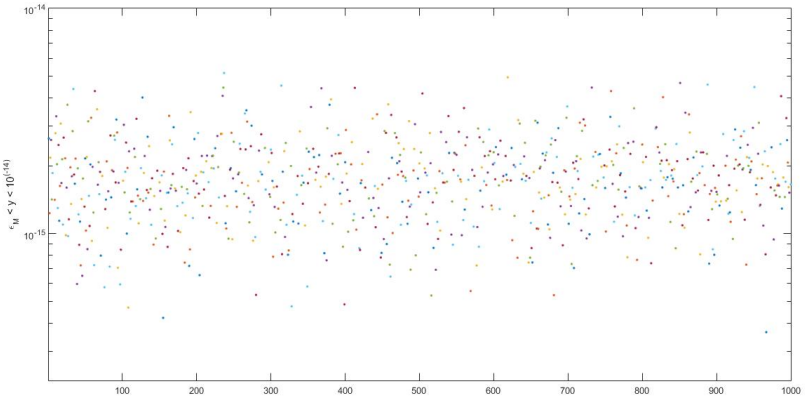

Figure B.5: Eigenvalue comparison chart.

In this case, for every matrix A, if v is its eigenvalue vector computed by MATLAB and v' the one calculated by the implementation this dissertation presents, then

$$\frac{\|v - v'\|}{\|v\|} \leq 80 \cdot \epsilon_M.$$

B.2.1 Eigenvectors of the complex Schur form

For the eigenvectors we will not use the usual test (B.1), but rather one given by Stewart in [1], on page 104. We know that both the Hessenberg reduction and the Schur form reduction work properly. Let A be a complex matrix and $T = Q^*AQ$ its complex Schur form. Then, as it was shown in §4.5, if X is the matrix of right eigenvectors of T, QX will be the matrix of right eigenvectors of A. Hence, the analysis of the accuracy of the eigenvectors will be done over the residuals $r_i = Tx_i - T_{ii}x_i$, i.e.,

$$\frac{\|r_i\|}{\|T\|\,\|x_i\|} \leq \gamma \epsilon_M.$$

Running the script

Norms of the residual $r_i = Tx_i - t_{ii}x_i$

```
% Script that computes the backwards error of the
   eigenvectors of a complex
% Schur form.

sim = 250;
c = 1;
for j = 1:sim
    n = randi([5,30]);
    A = exp(randn(n)*1i + randn(n));
    [T,Q] = complexschur(A);
    X = righteigvec(T);
    for i = 1:n
        semilogy(c,norm(T*X(:,i) - T(i,i)*X(:,i))/(
           norm(T)*norm(X(:,i))),'.');
        hold on
        c = c + 1;
    end

end
```

displays Figure B.6:

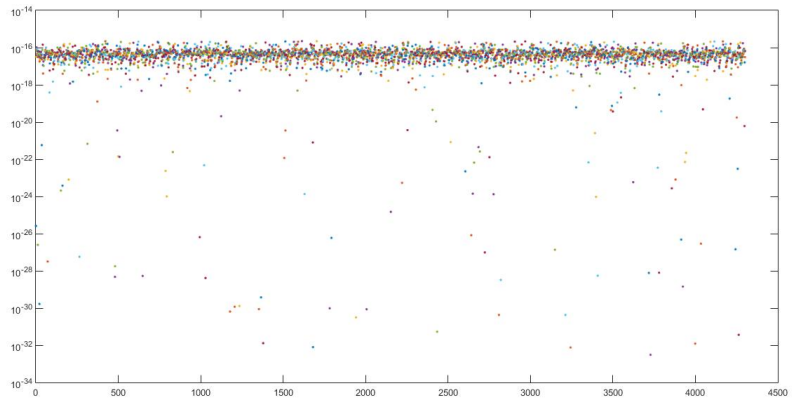

Figure B.6: Norms of the residual $r_i = Tx_i - t_{ii}x_i$.

The eigenvectors computed are extraordinarily precise: for every com-

plex matrix A, the eigenvectors of its Schur form T fulfill

$$\frac{\|r_i\|}{\|T\|\,\|x_i\|} \leq \epsilon_M.$$

B.3 The implicitly shifted QR algorithm

Finally, the implicitly shifted QR iteration, developed in §5 and implemented in A.4.

Reduction to real Schur form

```
% Script that computes the backwards error of
   compouting the real Schur
% form using the implicitly shifted QR algorithm.

sim = 1000;

for j = 1:sim

    n = randi([5 30]);
    A = randn(n);
    [T,Q] = realschur(A);
    nn = norm(A - Q*T*Q')/norm(A);
    semilogy(j,nn,'.');
    hold on

end

hold off
```

This script returns Figure B.7

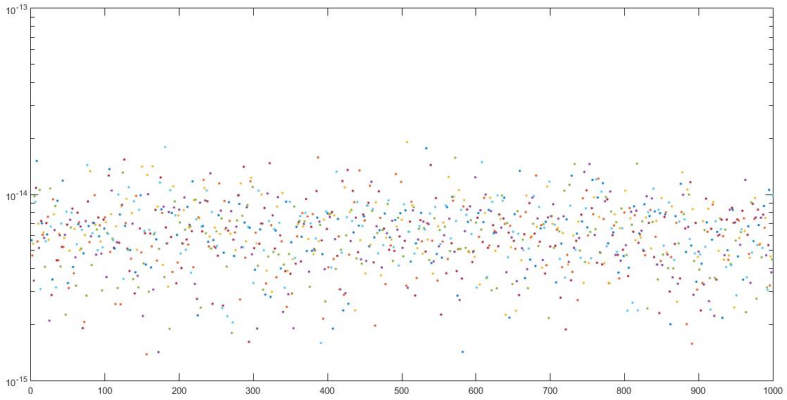

Figure B.7

which shows, as usual, that the relative error

$$\frac{\|A - QTQ^*\|}{\|A\|}$$

is of a size comparable to the machine epsilon.

As for how close the transformation matrices Q from being unitary, run the following script which shows Figure B.8.

Unitariness of the transformation matrix Q

```
% Script that shows how close are the transformation
   matrices Q to being unitary.

sim = 1000;

for j = 1:sim

    n = randi([5 30]);
    A = randn(n);
    [~,Q] = realschur(A);
    semilogy(j,norm(eye(n) - Q*Q')/norm(A),'.');
    semilogy(j,norm(eye(n) - Q'*Q)/norm(A),'o');
    hold on

end

hold off
```

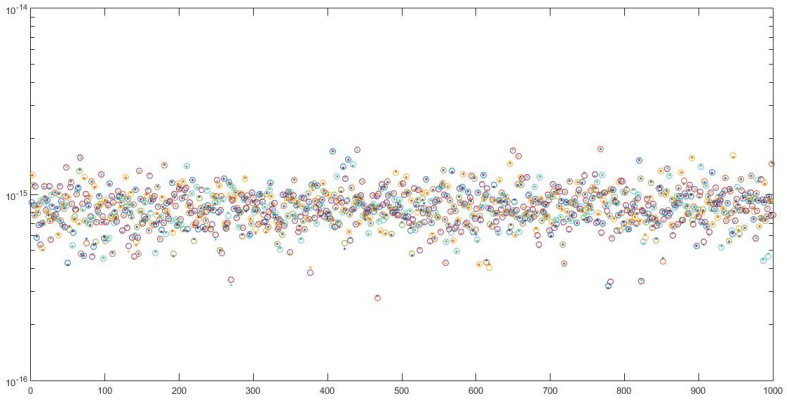

Figure B.8

Again, the implementation returns a unitary matrix Q.

B.3.1 Eigenvectors of the real Schur form

As for the eigenbases of the real Schur form, we designed the next script:

Norms of the residual $r_i = Tx_i - t_{ii}x_i$

```
% Script that computes the backwards error of the
   eigenvectors of a real
% Schur form matrix.

simulations = 500;
counter = 1;
bad = 0;

for k = 1 : simulations
    n = randi([5 10]);
    A = randn(n);
    [T,~] = realschur(A);

    Y = eigenbasis(T);

    % subdiagonal is a vector representing the
       elements of the subdiagonal: 1
    % if it is nonzero, 0 if it is zero
    subdiagonal = not(not(diag(T,-1)));

    for j = 1:(n-1)

```

```
21          if ( j ~= 1 && j ~= (n-1) )
22              if ( subdiagonal(j+1) == 1 ) % The
                   subdiagonal is in between two complex
                   conjugate eigenvalues
23                  continue %  subdiagonal(j-1) == 1 &&
24              elseif ( subdiagonal(j) == 1 )
25                  L = Y^(-1)*T*Y;
26                  a = norm(T*Y(:,j:j+1)-Y(:,j:j+1)*L(j:j
                       +1,j:j+1))/(norm(T)*norm(Y(:,j:j+1)
                       ));
27              else % Simple eigenvalue
28                  L = Y^(-1)*T*Y;
29                  a = norm(T*Y(:,j+1)-Y(:,j+1)*L(j+1,j
                       +1))/(norm(T)*norm(Y(:,j+1)));
30              end
31
32          elseif ( j == 1 )
33              if ( subdiagonal(1) == 1 ) % Complex
                   eigenvalue
34                  L = Y^(-1)*T*Y;
35                  a = norm(T*Y(:,1:2)-Y(:,1:2)*L
                       (1:2,1:2))/(norm(T)*norm(Y(:,1:2)))
                       ;
36              else % Simple eigenvalue
37                  L = Y^(-1)*T*Y;
38                  a = norm(T*Y(:,1)-Y(:,1)*L(1,1))/(norm
                       (T)*norm(Y(:,1)));
39              end
40
41          elseif ( j == (n-1) )
42              if ( subdiagonal(n-1) == 1 ) % Complex
                   eigenvalue
43                  L = Y^(-1)*T*Y;
44                  a = norm(T*Y(:,n-1:n)-Y(:,n-1:n)*L(n
                       -1:n,n-1:n))/(norm(T)*norm(Y(:,n-1:
                       n)));
45              else % Simple eigenvalue
46                  L = Y^(-1)*T*Y;
47                  a = norm(T*Y(:,n)-Y(:,n)*L(n,n))/(norm
                       (T)*norm(Y(:,n)));
48              end
49
50          end
51
```

```
52          if (a > 10^(-13))
53              bad = bad + 1;
54              disp(A);
55              return
56          end
57
58          semilogy(counter,a,'.')
59          hold on
60          counter = counter + 1;
61
62      end
63
64  end
65
66  ylabel('10^{-18} < y < 10^{-15}')
67  axis([1 counter 10^(-18) 10^(-15)])
68
69  hold off
```

This program computes the relative error −with the same criteria used for the eigenvectors of the complex Schur form− of either the eigenbases or the eigenvectors, depending on the size of the block. Figure B.9 shows the output.

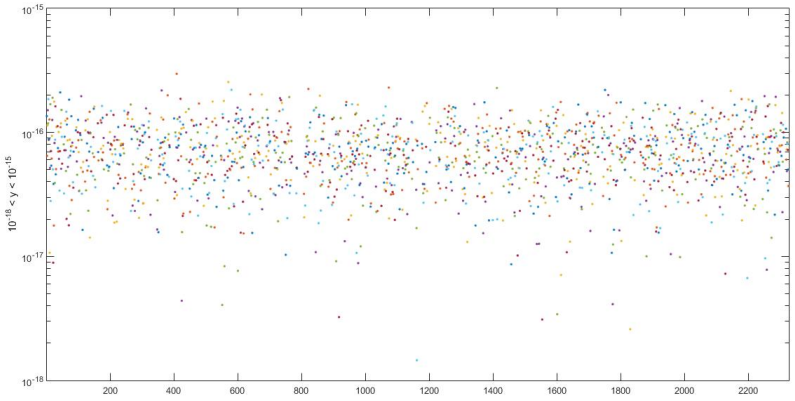

Figure B.9: Relative error of eigenbases and eigenvectors

Since for nearly every eigenbasis and eigenvector

$$\frac{\|r_i\|}{\|T\|\,\|x_i\|} \leq 10\epsilon_M,$$

the computation of those is accurate −do not forget the examples given on

the final part of §5.6.

Bibliography

[1] G. W. STEWART, *Matrix Algorithms Volume II: Eigensystems*, SIAM, Philadelphia, 2001.

[2] D. S. WATKINS, *Understanding the QR algorithm*, SIAM Rev., 24 (1982), pp. 427-440.

[3] D. S. WATKINS, *The QR algorithm revisited*, SIAM Rev., 50 (2008), pp. 133-145.

[4] J. W. DEMMEL, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[5] G. GOLUB, F. UHLIG, *The QR algorithm: 50 years later its genesis by John Francis and Vera Kublanovskaya and subsequent developments*, IMA Journal of Numerical Analysis, 29 (2006), pp. 467-485.

[6] H. J. BUUREMA, *A geometric proof of convergence for the QR method*, Ph.D. Thesis, Rijksuniversiteit te Groningen, The Netherlands, 1970.

[7] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

[8] J. G. F. FRANCIS, *The QR transformation, a unitary analogue to the LR transformation−part 1*, Comput. J., 4 (1961), pp. 265-271.

[9] J. G. F. FRANCIS, *The QR transformation−part 2.*, Comput. J., 4 (1961), pp. 332-345.

[10] V. N. KUBLANOVSKAYA, *The solution of the eigenvalue problem for an arbitrary matrix*, Trudy Mat. Inst. Steklov, 66 (1962), pp. 113-115 (in Russian).

[11] H. RUTISHAUSER, *Solution of eigenvalue problems with the LR-transformation* Nat. Bur. Stand. Appl. Math. Ser, 49 (1968), pp. 47-81.

[12] C. T. FIKE, *Note on the practical computation of proper values*, J. Assoc. Comput. Mach., 6 (1959), pp. 360-362.

[13] D. DAY, *How the QR algorithm fails to converge and how to fix it*, Technical Report 96-0913J, Albuquerque, NM, April 1996.

[14] S. BATTERSON, *Convergence of the shifted QR algorithm on 3 by 3 normal matrices*, Numer. Math., 58 (1990), pp. 341-352.

[15] F. CHATELIN, *Eigenvalues of Matrices: Revised Edition*, SIAM, Philadelphia, 2012.

[16] J. B. ZABALLA, *El problema de los valores propios*, *http://www.ehu.eus/izaballa/Ana_Matr/Apuntes/lec10.pdf*.

[17] J. B. ZABALLA, *Normas de vectores y matrices*, *http://www.ehu.eus/izaballa/Ana_Matr/Apuntes/lec2.pdf*.

[18] J. B. ZABALLA, *Valores singulares*, *http://www.ehu.eus/izaballa/Ana_Matr/Apuntes/lec3.pdf*.