Parallel computation of the hyperbolic QR factorization

Čaklović, Gayatri

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Gayatri Čaklović

PARALLEL COMPUTATION OF THE HYPERBOLIC QR FACTORIZATION

Diplomski rad

Voditelj rada:
prof. dr. sc. Sanja Singer

Zagreb, June, 2018

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Contents

Chapter 1

Introduction

Let $G \in \mathbb{F}^{m\times n}$ be a matrix, where $m \geq n$ and $\mathbb{F}$ is either $\mathbb{R}$ or $\mathbb{C}$. These fields are equipped with the Euclidian scalar product, therefore a simple QR factorization of matrix G exists

$$G = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

where Q is an unitary matrix and R_1 is upper triangular.

Suppose we do not have the Euclidian scalar product, but a different one, described by a diagonal matrix $J = \mathrm{diag}(j_1, j_2, \ldots, j_m)$, $j_i \in \{-1, 1\}$. Our non-Euclidian J-scalar product can then be represented as

$$[x, y] = \langle Jx, y\rangle = y^* Jx = \sum_{i=1}^{m} j_i x_i \overline{y_i}, \quad x, y \in \mathbb{F}^m. \tag{1.1}$$

Note that the "J-scalar product" is a scalar product only if the matrix J is Hermitian and positive definite. If this is not the case, we lost the property of non-negativity of the norm of the vectors and the characterization of orthogonality, hence we need a new definition of orthogonal matrices as well as a new definition for QR factorization. The new QR factorization, with expect to the given J is usually called the JQR factorization. If J is diagonal, where the diagonal elements are -1 or 1, the corresponding factorization is called a hyperbolic QR factorization.

The purpose of this thesis is to present and explain a numerically stable algorithm with a functional pivoting strategy for computing the hyperbolic QR factorization. In every step of the algorithm, depending on the strategy, a Givens-like (block) rotations or a Householder-like (block) J-reflectors will be used. In the next few chapters we will provide basic definitions of these objects and our implementation of them in the algorithm. The pivoting strategy will be explained at the end. After presenting the theoretical background and a sequential pseudocode, an explanation of how the algorithm should look like in parallel will be given. The implementation was tested on Intel Xeon Phi 7210.

Chapter 2

Basic definitions

This chapter is meant to be a collection of all definitions used in this thesis. We begin with an earlier mentioned definition of a J-scalar product.

Definition 2.1. *Let J be a $\mathbb{C}^{m\times m}$ matrix. A J-scalar product is*

$$[x, y] = \langle Jx, y\rangle = y^* Jx, \quad x, y \in \mathbb{F}^m. \tag{2.1}$$

If J is of form $J = \operatorname{diag}(j_1, j_2, \ldots, j_m)$, where the diagonal elements satisfy $j_i \in \{-1, 1\}$, then the product is called hyperbolic scalar product. If J is given by

$$J = \operatorname{diag}(J_0, J_0, \ldots, J_0), \quad J_0 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

then it is called symplectic.

From now on, our J will have a hyperbolic form, or often referred as indefinite. In an Euclidian scalar product, unitary matrices do not change the angles between vectors, in other words, they preserve the scalar product: $\langle Ux, Uy\rangle = \langle x, y\rangle$. This motivates our next definition.

Definition 2.2. *A matrix $U \in \mathbb{C}^{m\times m}$ is called a J-unitary, i.e., unitary according to the scalar product defined by* (1.1)*, if*

$$[Ux, Uy] = [x, y], \quad x, y \in \mathbb{C}^m. \tag{2.2}$$

As a consequence of the definition 2.2 and (2.1), U is J-unitary if and only if $U^*JU = J$ is satisfied. Due to simple calculations

$$0 = [Ux, Uy] - [x, y] = \langle x, (U^*JU - J)y\rangle, \quad \forall x, y \in \mathbb{C}^m$$

and choosing $x = (U^*JU - J)y$, we get $U^*JU = J$. The other way of the if statement is the consequence of the fact $\langle v, x\rangle = 0, \forall x \Rightarrow v = 0$.

Let us now properly define the main object of our interest: the hyperbolic QR factorization.

Definition 2.3. *(Hyperbolic QR factorization). Let $G \in \mathbb{C}^{m\times n}$, $m \geq n$ and let J have a hyperbolic form, such that $A = G^*JG$ is non-singular. A factorization*

$$G = P_1QRP_2^* = P_1Q\begin{bmatrix}R_1\\0\end{bmatrix}P_2^*, \quad Q^*J'Q = J', \quad J' = P_1^*JP_1, \tag{2.3}$$

where P_1 and P_2 are permutation matrices, matrix Q is J'-unitary and R_1 is block upper triangular with diagonal blocks of order 1 or 2, is called a hyperbolic QR factorization of G according to J.

The definition is slightly different from the ordinary QR factorization, where row and column permutations are not necessary for existence of the factorization. The necessity for column permutations P_2 will be clear later. For now, we can think of them as standard Householder reflectors for a single column reduction, but the column needs to have some extra properties which are crucial for the existence of the reflector. Let's now explain why row permutations are needed through a following example.

Example 2.4. *We want to find the hyperbolic QR decomposition for a matrix*

$$G = \begin{bmatrix}x\\y\end{bmatrix}, \quad J = \operatorname{diag}(1, -1).$$

where $|y| \geq |x|$. Our aim is to find a matrix Q such that

$$G = QR = \begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}r\\0\end{bmatrix}$$

and Q is J-unitary. From the characterization $Q^*JQ = J$ we get three different conditions:

$$\begin{aligned}|a|^2 - |c|^2 &= 1,\\ |b|^2 - |d|^2 &= -1,\\ a\overline{b} - c\overline{d} &= 0,\end{aligned}$$

and from the $G = QR$ we obtain

$$\begin{aligned}x &= ar\\ y &= cr\end{aligned}$$

and finally

$$\frac{x}{y} = \frac{a}{c}.$$

Using the fact that $c \neq 0$ *(which is left for the reader to prove) we get*

$$1 \geq \left|\frac{y}{x}\right|^2 = \left|\frac{a}{c}\right|^2 = 1 + \frac{1}{|c|^2}$$

which contradicts the fact $1 \geq 1 + 1/|c|^2$.

The existence is yet to be shown. The basic idea is to compute a sequence of J-unitary matrices $Q^{(i)}$ and apply them on G. The method is practically identical to the standard QR factorization, but with a catch: we still need to find the reduction tools to accomplish that.

Just as J-unitary matrices have motivational origins as unitary matrices have, Hermitian ones will have too, but an equivalent statement as a definition will be provided instead.

Definition 2.5. *A matrix* $A \in \mathbb{C}^{n\times n}$ *is called J-Hermitian if* $A = J^{-1}A^*J$ *is satisfied.*

A generalized concept of reflectors is also an analogue definition of the properties satisfied by unitary reflectors.

Definition 2.6. *A matrix* $Q \in \mathbb{C}^{m\times m}$ *will be called a J-reflector if the following conditions are met:*

1. *Q is J-unitary:* $Q^*JQ = J$
2. *Q is J-Hermitian:* $JQ = Q^*J$
3. *Q is a reflector:* $Q^2 = I$.

Moreover, just like in an unitary case, any two of the conditions impose the third one (proof of that fact can be found in [16]).

The next chapter will provide conditions when such a reflector exists.

Chapter 3

J-unitary Householder-like reflectors

Our aim in this chapter is to show some properties of J-reflectors as well as their existence. Let $\mathscr{M}$ be an arbitrary subset of $\mathbb{F}^m$ and let J have a hyperbolic form. The J-orthogonal companion od $\mathscr{M}$ in $\mathbb{F}^m$ is defined as

$$\mathscr{M}^{\perp} = \{x \in \mathbb{F}^m \mid [x, y] = 0,\ \forall y \in \mathscr{M}\}.$$

From now on, $\mathscr{M}$ will be a subspace of $\mathbb{F}^m$. It can also be shown that $\dim \mathscr{M} + \dim \mathscr{M}^{\perp} = m$ and $(\mathscr{M}^{\perp})^{\perp} = \mathscr{M}$. Further on, a subspace $\mathscr{M}$ will be called *nondegenerate*, with respect to the J-scalar product, if

$$\mathscr{M} \cap \mathscr{M}^{\perp} = \{0\}.$$

In other words, if $x \in \mathscr{M}$ and $[x, y] = 0$, $\forall y \in \mathscr{M} \Rightarrow x = 0$. Otherwise, $\mathscr{M}$ is *degenerate*. Therefore, we can conclude that when $\mathscr{M}$ or $\mathscr{M}^{\perp}$ is nondegenerate, then the subspaces are in a direct complement, meaning $\mathscr{M} \dot{+} \mathscr{M}^{\perp} = \mathbb{F}^m$ and vice versa.

Shifting now to matrices, a matrix $W \in \mathbb{F}^{n \times p}$, for some p, is said to be *nondegenerate*, with respect to the J-scalar product, if and only if $\operatorname{Im}(W)$ is a nondegenerate subspace, otherwise, W is *degenerate*.

At the moment, these definitoins seem disconnected from our theory, but they will play an important role for the existence of such J-unitary reflectors. If one recalls, a Householder reflector in an Euclidian space is defined as

$$\widetilde{H}(w) = I_n - \frac{2}{w^* w} w w^*,$$

where if we set $w = f - g$ for some $f, g \in \mathbb{F}^m$ such that $\|f\| = \|g\|$ and $f^* g \neq 0$, we get $\widetilde{H}(w) f = g$. If observed more closely, we have a scalar product $w^* w$. Our aim is to construct a similar object, so instead of an Euclidean scalar product, we will simply put a J-scalar product there. If we proceed in that style, it is important to have w which satisfies $w^* J w \neq 0$ and a functional J-reflector $H(w)$ (see definition 2.6 for J-reflectors), preferably

with an ability to do things such as $H(w)f = g$, for some $f, g \in \mathbb{F}^m$ that satisfy $f^*Jf = g^*Jg$ and $f^*Jg \neq 0$.

The requirement $H^2 = I$ contains a lot of information. The fact that $(H - I)(H + I) = 0$ holds, indicates that H has eigenvalues in $\{-1, 1\}$. If $H \neq I$, then H has exactly two subspaces $\mathscr{M}_-$ and $\mathscr{M}_+$ which correspond to eigenvalues -1 and 1 respectively. The spaces also satisfy $\mathscr{M} \dotplus \mathscr{M}^\perp = \mathbb{F}^m$ and

$$Hx = -x, \quad \forall x \in \mathscr{M}_-, \tag{3.1}$$
$$Hy = y, \quad \forall y \in \mathscr{M}_+. \tag{3.2}$$

Since H actually reflects $\mathscr{M}_-$ with respect to $\mathscr{M}_+$, the name reflector is clarified. In the Euclidean case, $f - g$ is a normal of the hiperplane over which the reflection is done.

In addition, if H is J-unitary (2.2), then $\forall x \in \mathscr{M}_-$ and $\forall y \in \mathscr{M}_+$ we have

$$[x, y] = [Hx, Hy] = [-x, y] = -[x, y],$$

from where we get $[x, y] = 0$. From this, we can conclude that $\mathscr{M}_-^\perp = \mathscr{M}_+$ and vice versa, $\mathscr{M}_+^\perp = \mathscr{M}_-$, so both spaces are nondegenerate. Now we see that H reflects a nondegenerate space to its J-orthogonal complement. This discussion can be summarized as a proposition.

Proposition 3.1. *Let $J \in \mathbb{F}^{m\times m}$ have a hyperbolic form.*

1. *A matrix $H \in \mathbb{F}^{m\times m}$ is a reflector if and only if there exists a pair of complementary spaces $\mathscr{M}_-$ and $\mathscr{M}_+$ such that*

$$Hx = -x, \quad \forall x \in \mathscr{M}_-,$$
$$Hy = y, \quad \forall y \in \mathscr{M}_+.$$

2. *Let H be a reflector. H is a J-reflector if and only if the above spaces $\mathscr{M}_-$ and $\mathscr{M}_+$ are mutually J-orthogonal. If so, both subspaces are nondegenerate.*

The complete proof of Proposition 3.1 can be found in [16]. A statement similar to statement 2 in proposition 3.1 is also valid: H is a J-reflector if and only if there exists a nondegenerate subspace $\mathscr{M}_-$ such that (3.1) holds.

Let's now make our first steps towards an existence of a such reflector.

3.1 Basic J-reflectors

Before we give a proper definition, an explanation of what a Moore–Penrose inverse is in order.

Definition 3.2. *For $A \in \mathbb{F}^{n\times p}$, a pseudoinverse of A, also known as the Moore–Penrose inverse, is defined as a matrix $A^+ \in \mathbb{F}^{p\times n}$ if it satisfies the following criteria:*

1. $AA^+A = A$
2. $A^+AA^+ = A^+$
3. $(AA^+)^* = AA^+$
4. $(A^+A)^* = A^+A$.

Now we have all the preliminaries to define a basic J-reflector.

Definition 3.3. *(Basic J-reflector). Let $J \in \mathbb{F}^{m\times m}$ be a given hyperbolic scalar product matrix. For a given vector $w \in \mathbb{F}^m$, a matrix $H \in \mathbb{F}^{m\times m}$ defined by*

$$H = H(W) = I_m - 2w(w^*Jw)^+w^*J \tag{3.3}$$

will be called a basic J-reflector generated by w.

Moreover, it can be shown (see [16]) that (3.3) satisfies all three conditions from definition 2.6.

Also one thing remains to be clarified: what is $(w^*Jw)^+$ in our case? Since w^*Jw is a scalar, from statement in definition 3.2, we can easily calculate

$$(w^*Jw)^+ = \begin{cases} \dfrac{1}{w^*Jw}, & \text{if } w^*Jw \neq 0, \\ 0, & \text{if } w^*Jw = 0. \end{cases}$$

3.2 Mapping by J-reflectors

Suppose that vectors f and g are given, and we are interested under which conditions there exists a basic J-reflector H such that $Hf = g$. Just from the definition of J-reflectors, we can get two necessary conditions. From the first statement in the definition 2.6 of a J-reflector, we get a J-isometry property:

$$f^*Jf = g^*Jg \tag{3.4}$$

and from the second statement from the same proposition, we get a J-symmetry property:

$$g^*Jf = f^*Jg. \tag{3.5}$$

In the unitary case (just put $J = I$) the above mentioned conditions are also sufficient conditions fot the existence of such a mapping. We will show through an example that, for a hyperbolic scalar product, (3.4) and (3.5) are not sufficient.

Example 3.4. *Let's try to find a J-reflector H so that*

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} f = g, \quad \textit{where} \quad f = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad g = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \quad J = \operatorname{diag}(1, -1).$$

One can see that J-isometry and J-symmetry are satisfied since

$$f^* J f = g^* J g = 0, \quad g^* J f = f^* J g = 0.$$

From the equation $Hg = f$ we get

$$a + b = 2 \tag{3.6}$$
$$c + d = 2. \tag{3.7}$$

Since H is a J-reflector, H has to be J-unitary. From $H^*JH = J$ we get

$$|a|^2 - |c|^2 = 1 \tag{3.8}$$
$$\overline{a}b - \overline{c}d = 0 \tag{3.9}$$
$$|b|^2 - |d|^2 = -1. \tag{3.10}$$

H also needs to be J-Hermitian, meaning $JH = H^*J$:

$$a = \overline{a} \Rightarrow a \in \mathbb{R}, \tag{3.11}$$
$$d = \overline{d} \Rightarrow d \in \mathbb{R}, \tag{3.12}$$
$$b + \overline{c} = 0. \tag{3.13}$$

Combining $a \in \mathbb{R}$, (3.9) *and* (3.13)*, we get*

$$b(a + d) = 0.$$

If $b = 0$, from (3.6) *we get that $a = 2$ and from* (3.10) *we get $|d|^2 = 1$ which gives $d = 1$ or $d = -1$, since d is real. From* (3.7) *we get $c = 1$ or $c = 3$, but this must be wrong, because* (3.8) *says $|c|^2 = 3$.*

Suppose now that $(a + d) = 0$, so summing up (3.6) *and* (3.7) *gives us $c + b = 4$. Knowing $b = -\overline{c}$, we get that $c - \overline{c} = 4$ which is impossible.*

We can conclude that for vectors f, g and J as such, a J-reflector does not exist, meaning J-isometry and J-symmetry are not sufficient enough conditions.

At last, two the following theorems will provide sufficient conditions for the existence of such mappings.

Theorem 3.5. *(Basic J-reflector mapping theorem). Let $J \in \mathbb{F}^{m\times m}$ be a hyperbolic scalar product matrix, and let $f, g \in \mathbb{F}^m$ be two discint vectors. There exists a basic J-reflector $H = H(w)$ such that $H(w)f = g$ if and only if*

1. *vectors g and f satisfy the J-isometry* (3.4) *and J-symmetry* (3.5) *properties*
2. *$d = g - f \neq 0$ is nondegenerate, meaning $d^*Jd \neq 0$.*

Furthermore, whenever H exists, it is unique. More precisely, H can be generated by any vector $w \in \mathbb{F}^m$ such that

$$w = \lambda d, \quad \lambda \in \mathbb{F}, \quad \lambda \neq 0.$$

Then w is also nondegenerate and $H(w) = H(d)$.

Finally, the same remains valid if we replace f with $-f$ and d with $s = f + g$.

Theorem 3.5 gives us existence of mapping when $[f, f] = [g, g]$.

Theorem 3.6. *Let $J \in \mathbb{F}^{m\times m}$ be as hyperbolic scalar product matrix and let $f, g \in \mathbb{F}^m$ be two vectors such that $f^*Jf = g^*Jg \neq 0$. There exists a basic J-reflector $H = H(w)$ such that*

$$Hf = \sigma g.$$

In the complex case $\mathbb{F} = \mathbb{C}$, the factor σ is given by

$$\sigma = -\operatorname{sign}(g^*Jg)\frac{f^*Jg}{|f^*Jg|}, \quad if \quad f^*Jg \neq 0,$$

and if $f^*Jg = 0$, we can take any $\sigma \in \mathbb{C}$ such that $|\sigma| = 1$.

Proofs of these theorems can be found in [16], as well as their generalizations for mapping matrices $F \in \mathbb{F}^{m\times n}$ into $G \in \mathbb{F}^{m\times n}$. This concludes the existence of such objects and gives a way to construct them.

3.3 Householder J-reflectors in hyperbolic QR

Now we have a theoretical background, we are interested in how to use it. Suppose we have a given matrix $J = \operatorname{diag}(j_1, j_2, \ldots, j_m)$ in a hyperbolic form and a matrix $G \in \mathbb{F}^{m\times n}$, $m \geq n$, so that $A = G^*JG$ is non-singular. We want to find a sequence of J-unitary reflectors $H^{(k)}$, $k = 1, \ldots, n$ so that

$$HG = H^{(k)}H^{(k-1)} \ldots H^{(2)}H^{(1)}G = G^{(k)}, \tag{3.14}$$

where $G^{(k)}$ has the form

$$G^{(k)} = \begin{bmatrix} g_{11}^{(k)} & g_{12}^{(k)} & g_{13}^{(k)} & \cdots & g_{1k}^{(k)} & \cdots & g_{1n}^{(k)} \\ 0 & g_{22}^{(k)} & g_{23}^{(k)} & \cdots & g_{2k}^{(k)} & \cdots & g_{2n}^{(k)} \\ 0 & 0 & g_{33}^{(k)} & \cdots & g_{3k}^{(k)} & \cdots & g_{3n}^{(k)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & g_{kk}^{(k)} & \cdots & g_{kn}^{(k)} \\ 0 & 0 & 0 & \cdots & g_{k+1,k}^{(k)} & \cdots & g_{k+1,n}^{(k)} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & g_{m-1,k}^{(k)} & \cdots & g_{m-1,n}^{(k)} \\ 0 & 0 & 0 & \cdots & g_{mk}^{(k)} & \cdots & g_{mn}^{(k)} \end{bmatrix} = \begin{bmatrix} G_{11}^{(k)} & G_{12}^{(k)} \\ 0 & G_{22}^{(k)} \end{bmatrix}, \quad J = \begin{bmatrix} J_1 & 0 \\ 0 & J_2 \end{bmatrix}.$$

Just for the sake of explaining how the reduction with Householder reflectors should work, suppose that in every step k of the method, there exists at least one column $g_l^{(k)}[k:m]$, in a not yet reduced submatrix $G_{22}^{(k)}$, so that $(g_l^{(k)}[k:m])^* J_2 g_l^{(k)}[k:m] \neq 0$.

When we were explaining the purpose of row permutations P_1, we did not properly clarify the need for column permutations P_2. For example, if we want to proceed with the $(k+1)th$ step of reduction and $g_k^{(k)}[k:m]$ is J_2-orthogonal, we would need to find a nondegenerate column $g_l^{(k)}[k:m]$ in the submatrix $G_{22}^{(k)}$, and swap columns $g_k^{(k)}$ and $g_l^{(k)}$ in $G^{(k)}$, in other words, we continue working on $\hat{G}^{(k)} = G^{(k)} P_2$.

At this point, $\hat{g}_k^{(k)}[k:m]$ is a nondegenerate column of the unreduced submatrix in $\hat{G}_{22}^{(k)}$, and we want to find a J-reflector which will give us

$$H^{(k+1)} g = f, \quad \text{where} \quad g = \begin{bmatrix} g_k \\ g_{k+1} \\ \vdots \\ g_m \end{bmatrix} = \begin{bmatrix} \hat{g}_{kk}^{(k)} \\ \hat{g}_{k+1,k}^{(k)} \\ \vdots \\ \hat{g}_{mk}^{(k)} \end{bmatrix} \quad \text{and} \quad f^* J_2 f = g^* J_2 g.$$

If we define

$$f = (|g^* J_2 g|^{\frac{1}{2}}, 0, \ldots, 0)$$

then $f^* J_2 f = g^* J_2 g$ holds if $g^* J_2 g > 0$ and $j_k = 1$. If $j_k = -1$, then there exists an element $j_i = 1$ on the diagonal of J, but in J_2. We now use a permutation matrix P_1 for swapping rows k and i in $\hat{G}^{(k)}$ and also for swapping the diagonal elements j_k and j_i in J. Now, our matrix J changed into $\widetilde{J} = P_1^* J P_1$, matrix $\hat{G}^{(k)}$ changed into $\widetilde{G}^{(k)} = P_1^* \hat{G}^{(k)}$ and g changed into $\tilde{g} = P_1^* g$. The same thing is done if $g^* J_2 g < 0$ and $j_k = 1$.

Now $f^* \widetilde{J}_2 f = \tilde{g}^* \widetilde{J}_2 \tilde{g}$ holds. Note that with swapping diagonal and row elements, we did not change the nature of the J-scalar product in G, in other words, $\tilde{g}^* \widetilde{J}_2 \tilde{g} = g^* J_2 g$ holds.

Let's proceed with computing σ from Theorem 3.6:

$$\sigma = -\operatorname{sign}(\tilde{g}^*\widetilde{J}_2\tilde{g})\frac{f^*\widetilde{J}_2\tilde{g}}{|f^*\widetilde{J}_2\tilde{g}|} = -\tilde{j}_k\frac{|f^*\widetilde{J}_2\tilde{g}|\tilde{j}_k\tilde{g}_k}{\left||\tilde{g}^*\widetilde{J}_2\tilde{g}|^{\frac{1}{2}}\tilde{g}_k\right|} = -\frac{\tilde{g}_k}{|\tilde{g}_k|} = -\frac{\tilde{g}_{kk}^{(k)}}{|\tilde{g}_{kk}^{(k)}|}. \tag{3.15}$$

Put $w = (w_k, w_{k+1}, \ldots, w_m) = \sigma f - \tilde{g}$ and instead of just brainlessly computing $w^*\widetilde{J}_2 w$ in form of a sum, which consumes computing time, we can reuse g^*J_2g as in the below formulas:

$$\begin{aligned} |w_k| &= \left| -\frac{\tilde{g}_k}{|\tilde{g}_k|}|g^*J_2g|^{\frac{1}{2}} - \tilde{g}_k\right| = \left|\tilde{g}_k\left(\frac{|g^*J_2g|^{\frac{1}{2}}}{|\tilde{g}_k|} + 1\right)\right| = \left(\frac{|g^*J_2g|^{\frac{1}{2}}}{|\tilde{g}_k|} + 1\right)|\tilde{g}_k| = |g^*J_2g|^{\frac{1}{2}} + |\tilde{g}_k|, \\ w^*\widetilde{J}_2w &= |w_k|^2\tilde{j}_k + \cdots + |w_m|^2\tilde{j}_m = \tilde{j}_k(|g^*J_2g| + 2|\tilde{g}_k||g^*J_2g|^{\frac{1}{2}}) + g^*J_2g. \end{aligned} \tag{3.16}$$

We see now that w is nondegenerate, because $\tilde{j}_k = \operatorname{sign}(g^*J_2g)$, therefore $w^*\widetilde{J}_2w \neq 0$. The only thing left is to define a J-reflector that satisfies $H(w)\tilde{g} = \sigma f$. The reflector $H(w)$ then has a form:

$$H(w) = I_{m-k} - 2\frac{w^*w\widetilde{J}_2}{w^*\widetilde{J}_2w}. \tag{3.17}$$

Placing $H(w)$ in a matrix as

$$H^{(k+1)} = \begin{bmatrix} I_k & 0 \\ 0 & H(w) \end{bmatrix}$$

we get

$$G^{(k+1)} = H^{(k+1)}\widetilde{G}^{(k)} = \begin{bmatrix} \tilde{g}_{11}^{(k+1)} & \tilde{g}_{12}^{(k+1)} & \tilde{g}_{13}^{(k+1)} & \cdots & \tilde{g}_{1k}^{(k+1)} & \cdots & \tilde{g}_{1n}^{(k+1)} \\ 0 & \tilde{g}_{22}^{(k+1)} & \tilde{g}_{23}^{(k+1)} & \cdots & \tilde{g}_{2k}^{(k+1)} & \cdots & \tilde{g}_{2n}^{(k+1)} \\ 0 & 0 & \tilde{g}_{33}^{(k+1)} & \cdots & \tilde{g}_{3k}^{(k+1)} & \cdots & \tilde{g}_{3n}^{(k+1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & \tilde{g}_{kk}^{(k+1)} & \cdots & \tilde{g}_{kn}^{(k+1)} \\ 0 & 0 & 0 & \cdots & 0 & \cdots & \tilde{g}_{k+1,n}^{(k+1)} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 0 & \cdots & \tilde{g}_{m-1,n}^{(k+1)} \\ 0 & 0 & 0 & \cdots & 0 & \cdots & \tilde{g}_{mn}^{(k+1)} \end{bmatrix}.$$

Does this method really form a hyperbolic QR factorization in the sense of Definition 2.3 is unclear. Due to all row and column permutations, instead of the form (3.14), we got

$$H^{(n)}(P_1^{(n)})^*H^{(n-1)}(P_1^{(n-1)})^* \ldots H^{(2)}(P_1^{(2)})^*H^{(1)}(P_1^{(1)})^*GP_2^{(1)}P_2^{(2)} \ldots P_2^{(n-1)}P_2^{(n)} = R. \tag{3.18}$$

The proof that the sequence of transformations (3.18) really satisfies a hyperbolic QR factorization can be found in [15]. The final matrices, as in the Definition 2.3 of the hyperbolic QR factorization, would then be:

$$\widetilde{P}_1 = P_1^{(1)} P_1^{(2)} \dots P_1^{(n)}, \tag{3.19}$$

$$\widetilde{P}_2 = P_2^{(1)} P_2^{(2)} \dots P_2^{(n)}, \tag{3.20}$$

$$\widetilde{Q} = (P_1^{(n)})^* \dots (P_1^{(2)})^* H^{(1)} P_1^{(2)} H^{(2)} P_1^{(3)} H^{(3)} \dots P_1^{(n)} H^{(n)}, \tag{3.21}$$

$$\widetilde{A} = (\widetilde{P}_2)^* A \widetilde{P}_2. \tag{3.22}$$

Of course, this approach is not always possible. Problems occur when there are no more nondegenerate columns left in the submatrix of $\widetilde{G}^{(k)}$, or in the numerical world, a J_2-scalar product is smaller than some treshold $\varepsilon > 0$, $(\tilde{g}^{(k)}[k : m])^* J_2 (\tilde{g}^{(k)}[k : m]) < \varepsilon$. In addition, a question of numerical stability rises which is why another method will be presented in the next chapter.

Chapter 4

A Givens-like algorithm

As we saw in the previous chapter, problems occur when there are no more nondegenerate columns in the working submatrix. In such case we then need to combine two columns and figure out how to annihilate them both. In that case, G may have a 2×2 block in the reduced form. This is still compatible with the definition of a hyperbolic QR factorization 2.3. From now on, let $J = (j_1, j_2, \ldots, j_m)$ have a hyperbolic form, $G \in \mathbb{C}^{m \times n}$, $m \geq n$, and let $A = G^* JG$ be non-singular.

Let us introduce elementary J-unitary rotations $U_g, U_h \in \mathbb{R}^{m \times m}$. They have value 1 on the diagonal and 0 everywhere else, except in some arbitrary positions (s, s), (s, l), (l, s), (l, l):

$$U_g([s, l], [s, l]) = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix}, \tag{4.1}$$

and

$$U_h([s, l], [s, l]) = \begin{bmatrix} \cosh(\varphi) & \sinh(\varphi) \\ \sinh(\varphi) & \cosh(\varphi) \end{bmatrix}. \tag{4.2}$$

U_g is an ordinary Givens rotation. There exists an angle φ so that U_g annihilates an element at the position (s, l). Matrix U_g is also J-unitary if the elements in J satisfy $j_s = j_l$. Matrix U_h is J-unitary if the elements of J satisfy $j_s = -j_l$.

Example 4.1. *For given matrices G, and J*

$$G = \begin{bmatrix} 1 & 4 & 2 \\ 1 & 4 & -2 \\ 1 & -1 & 0 \\ 1 & 4 & 2 \end{bmatrix}, \quad J = \operatorname{diag}(1, -1, 1, 1)$$

find the J-unitary rotation $U([1, 3], [1, 3])$ *that will annihilate the element at the position* $(1, 3)$. *First note that the J-unitary rotation is a trigonometric rotation* U_g *with the follow-*

ing form

$$U_g([1,3],[1,3]) = \begin{bmatrix} \cos(\varphi) & 0 & \sin(\varphi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We want to to find the angle φ such that the new element at the position $(1,3)$ is equal to zero, i.e.,

$$U_g([1,3],[1,3])\, G([1,3],[1,3]) = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix}.$$

Simple computation yields

$$\begin{aligned} \cos(\varphi) + \sin(\varphi) &= r, \\ -\sin(\varphi) + \cos(\varphi) &= 0, \end{aligned}$$

and we get a solution $\varphi = \frac{\pi}{4} \Rightarrow \cos(\varphi) = \frac{1}{\sqrt{2}}, \sin(\varphi) = \frac{1}{\sqrt{2}}$. Computing $U_g G$ we really see that the element in position $(1,3)$ is 0.

$$U_g G = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 2 \\ 1 & 4 & -2 \\ 1 & -1 & 0 \\ 1 & 4 & 2 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & \frac{3\sqrt{2}}{2} & \sqrt{2} \\ 1 & 4 & -2 \\ 0 & -\frac{5\sqrt{2}}{2} & -\sqrt{2} \\ 1 & 4 & 2 \end{bmatrix}.$$

Note that rows $\neq 1, 3$ did not change.

If one recalls, our matrix $G^{(k)}$ after the $k-1$ steps had the form

$$G^{(k)} = \begin{bmatrix} g_{11}^{(k)} & g_{12}^{(k)} & g_{13}^{(k)} & \cdots & g_{1k}^{(k)} & \cdots & g_{1n}^{(k)} \\ 0 & g_{22}^{(k)} & g_{23}^{(k)} & \cdots & g_{2k}^{(k)} & \cdots & g_{2n}^{(k)} \\ 0 & 0 & g_{33}^{(k)} & \cdots & g_{3k}^{(k)} & \cdots & g_{3n}^{(k)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & g_{kk}^{(k)} & \cdots & g_{kn}^{(k)} \\ 0 & 0 & 0 & \cdots & g_{k+1,k}^{(k)} & \cdots & g_{k+1,n}^{(k)} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & g_{m-1,k}^{(k)} & \cdots & g_{m-1,n}^{(k)} \\ 0 & 0 & 0 & \cdots & g_{mk}^{(k)} & \cdots & g_{mn}^{(k)} \end{bmatrix} = \begin{bmatrix} G_{11}^{(k)} & G_{12}^{(k)} \\ 0 & G_{22}^{(k)} \end{bmatrix}, \quad J = \begin{bmatrix} J_1^{(k)} & 0 \\ 0 & J_2^{(k)} \end{bmatrix}.$$

Moreover, let the norms of the columns of $G_{22}^{(k)}$ are zeros or very small even if the elements in such columns are of reasonable size. This is possible only if there are mixed signs in

J. If the norms of the columns are exactly equal to 0 (and the elements in the columns are not 0), than there is no hyperbolic rotation that will annihilate the given sub-column to one element. For example, there is no hyperbolic rotation which annihilates

$$G_{22}^{(k)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad J_2^{(k)} = \mathrm{diag}(1, -1).$$

Moreover, even if the J-norm of the column of $G_{22}^{(k)}$ is small (but the elements of $G_{22}^{(k)}$ reasonable in size), the hyperbolic angle will be quite big, and corresponding transformation U_h catastrophically conditioned. Therefore, we need to find another way to make matrix G close to the triangular. An idea is to work with two columns if the norms of the columns are small, and their J-scalar product is big enough.

Suppose we have a criteria that guarantees us which two columns in $G_{22}^{(k)}$ would be good candidates for the annihilation process. How we find those two pivot columns will be explained later, but for now, just for the sake of explaining the method, a pair of columns $(g_s^{(k)}[k:m], g_l^{(k)}[k:m])$ in $G_{22}^{(k)}$ will be chosen as pivot columns if the matrix

$$A_2 := \begin{bmatrix} (g_s^{(k)}[k:m])^* J_2^{(k)} (g_s^{(k)}[k:m]) & (g_s^{(k)}[k:m])^* J_2^{(k)} (g_l^{(k)}[k:m]) \\ (g_l^{(k)}[k:m])^* J_2^{(k)} (g_s^{(k)}[k:m]) & (g_l^{(k)}[k:m])^* J_2^{(k)} (g_l^{(k)}[k:m]) \end{bmatrix} \tag{4.3}$$

is non-singular, indefinite and with nonzero off-diagonal elements. It will be shown later that, if the case is that we cannot perform annihilation using Householder J-reflectors, then there always exist such columns in the working submatrix $G_{22}^{(k)}$ that satisfy (4.3) if and only if $A = G^* J G$ is non-singular.

After we found two good pivot columns, if necessary, two column permutations of $G^{(k)}$ are performed in order to swap columns $k \longleftrightarrow s$ and $k+1 \longleftrightarrow l$, thus $\hat{G}^{(k)} = G^{(k)} P_2$.

The main idea is to reduce column $\hat{g}_k^{(k)}[k:m]$ up to 2 elements using U_g. The simplest way to apply the Givens rotations in the complex case is to make column $\hat{g}_k^{(k)}[k:m]$ real. This is done by multiplying $\hat{G}^{(k)}$ with

$$\Phi_1 = \mathrm{diag}\left(1, 1, \ldots, 1, \frac{\hat{g}_{kk}^{(k)}}{|\hat{g}_{kk}^{(k)}|}, \frac{\hat{g}_{k+1,k+1}^{(k)}}{|\hat{g}_{k+1,k+1}^{(k)}|}, \ldots, \frac{\hat{g}_{mm}^{(k)}}{|\hat{g}_{mm}^{(k)}|}\right).$$

Note that a matrix $\Phi = \mathrm{diag}(e^{i\varphi_2}, e^{i\varphi_2}, \ldots, e^{i\varphi_m})$ is J-unitary, for all J of hyperbolic forms, meaning $\Phi^* J \Phi = J$.

Now, $\hat{\hat{G}}^{(k)} = \Phi_1 \hat{G}^{(k)}$, and we simply annihilate all the elements in $\hat{\hat{g}}_k^{(k)}[k:m]$ corresponding to sign j_k, and the element $\hat{\hat{g}}_{k,k}^{(k)}$ is replaced by their norm. Similar holds for the elements that correspond to sign $-j_k$ — element $\hat{\hat{g}}_{k,k+1}^{(k)}$ is replaced by their norm. If the element at position $(k, k+1)$ is not corresponding to sign $-j_k$, we perform another row permutation $P_1^* \hat{\hat{G}}^{(k)}$.

The same procedure is repeated on $P_1^*\hat{\hat{g}}_{k+1}^{(k)}[r+1:m]$, where r is the last row in $P_1^*\hat{\hat{g}}_k^{(k)}[k:m]$ that has a nonzero element after performing Givens rotations on the kth column of the working submatrix (note that $r = 1$ or r = 2). Remember that after every row permutation on $\hat{\hat{G}}^{(k)}$, we need to apply a diagonal permutation on J, since we need to preserve the J-scalar product.

This yields

$$\widetilde{G}^{(k)} = U_{k+1,-1}P_{1,4}^*U_{k+1,1}P_{1,3}^*\Phi_2U_{k,-1}P_{1,2}^*U_{k,1}P_{1,1}^*\Phi_1G^{(k)}P_2,$$

where $U_{i,j}$ is a composition of multiple Givens rotations and since a composition of multiple J-unitary matrices is again a J-unitary matrix, $U_{i,j}$ is J-unitary. To simplify, we can accomplish the same thing if we do all the row permutations first and then annihilate the elements. This yields

$$\widetilde{G}^{(k)} = (U_{k+1,-1}U_{k+1,1}\Phi_2U_{k,-1}U_{k,1}\Phi_1)(P_{1,4}^*P_{1,3}^*P_{1,2}^*P_{1,1}^*)G^{(k)}P_2 = Q^{(k)}P_1^*G^{(k)}P_2, \tag{4.4}$$

$$\widetilde{J}^{(k)} = P_1^*JP_1. \tag{4.5}$$

Note that $Q^{(k)}$ is J-unitary, $(Q^{(k)})^*JQ^{(k)} = J$, since it is a composition of J-unitary matrices.

There are five possible forms we can get after performing Givens rotations on two pivot columns. We will split them in cases **A** and **B**, depending on how many nonzero elements we have in kth column of $\widetilde{G}_{22}^{(k)}$.

Case A: The block $\widetilde{G}^{(k)}[k:k+4, k:k+2]$ has exactly two nonzero elements in the first column. This small part of $\widetilde{G}^{(k)}$ will be denoted by $G_{r2} = \widetilde{G}^{(k)}[k:k+r, k:k+2]$ and the corresponding part of $\widetilde{J}^{(k)}$ will be $J_{rr} = \widetilde{J}^{(k)}[k:k+r, k:k+r]$.

A1: Four nontrivial rows:

$$G_{42} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \\ 0 & g_{32} \\ 0 & g_{42} \end{bmatrix}, \quad J_{44} = \operatorname{diag}(j_{11}, -j_{11}, j_{33}, -j_{33}), \tag{4.6}$$

where $g_{11}, g_{21}, g_{32}, g_{42}$ are real and nonzero. At least g_{12} or g_{22} is nonzero, otherwise A_2 would have a zero off-diagonal element and since J-unitary transformations preserve J-scalar products (by definition), A_2 remains unchanged.

A2: Three nontrivial rows:

$$G_{32} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \\ 0 & g_{32} \end{bmatrix}, \quad J_{33} = \operatorname{diag}(j_{11}, -j_{11}, j_{33}), \tag{4.7}$$

with g_{11}, g_{21}, g_{32} real and nonzero. Also, g_{12} is nonzero or g_{22} is nonzero.

A3: Two nontrivial rows:

$$G_{22} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}, \quad J_{22} = \operatorname{diag}(j_{11}, -j_{11}), \tag{4.8}$$

with g_{11}, g_{21} real and nonzero. Also, g_{12} is nonzero or g_{22} is nonzero. This is the case where we already have a diagonal block of order 2 which is the form we needed.

Case B: The block $\widetilde{G}^{(k)}[k : k+4, k : k+2]$ has exactly one nonzero elements in the first column. This small part of $\widetilde{G}^{(k)}$ will be denoted by $G_{r2} = \widetilde{G}^{(k)}[k : k+r, k : k+2]$ and the corresponding part of $\widetilde{J}^{(k)}$ will be $J_{rr} = \widetilde{J}^{(k)}[k : k+r, k : k+r]$.

B1: Three nontrivial rows:

$$G_{32} = \begin{bmatrix} g_{11} & g_{12} \\ 0 & g_{22} \\ 0 & g_{32} \end{bmatrix}, \quad J_{33} = \operatorname{diag}(j_{11}, -j_{33}, j_{33}), \quad j_{11} = j_{33}, \tag{4.9}$$

with g_{11}, g_{22}, g_{32} real and nonzero, while g_{12} is nonzero.

B2: Two nontrivial rows:

$$G_{22} = \begin{bmatrix} g_{11} & g_{12} \\ 0 & g_{22} \end{bmatrix}, \quad J_{22} = \operatorname{diag}(j_{11}, j_{22}), \quad j_{11} = -j_{22}, \tag{4.10}$$

with g_{11}, g_{22} real and nonzero, while g_{12} is nonzero. This is also the case where we already have a diagonal block of order 2 which is the form we needed.

Remark 4.2. *Case **A1** can be transformed into **A2** and case **B2** can be transformed into **B1** using U_h transformations, but U_h can be very ill conditioned if $|\tanh(\varphi)| \approx 1$, meaning we will get wrong results if implementing it in our algorithm. It can be done for example if $|\tanh(\varphi)| \leq 0.5$, but since the following steps of reduction will have the same complexity as applying U_h, this was not implemented.*

4.1 Proper forms

We saw all possible forms of G_{r2}. The matrix can then be rewritten as

$$G_{r2} = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix},$$

where

$$G_1 = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}, \quad \text{and} \quad G_2 = \begin{cases} \begin{bmatrix} 0 & g_{32} \\ 0 & g_{42} \end{bmatrix}, & \text{if} \quad r = 4, \\ \begin{bmatrix} 0 & g_{32} \end{bmatrix}, & \text{if} \quad r = 3. \end{cases} \tag{4.11}$$

Forms of G_{r2} as described in **A1**, **A2** and **B1** will be called *proper forms* if $\det G_1 \neq 0$. If looked more closely, **B1** is already a proper form. The next steps will describe how to change forms **A1** and **A2** into proper forms (if they are not) and how to transform those forms into 2×2 blocks.

We give the transformation to proper forms as lemmas without proof (for proof see [15]).

Lemma 4.3. *Let G_{32}, J_{33} be in form **A2** as in* (4.7)*. There exist permutation matrices P'_1, P'_2 and J'_{33}-unitary matrix U, such that*

$$G'_{32} := U^{-1}(P'_1)^* G_{32} P'_2, \quad U^* J'_{33} U = J'_{33}, \quad J'_{33} := (P'_1)^* J_{33} P'_1,$$

and G'_{32}, J'_{33} are in proper form, meaning $\det G'_1 \neq 0$.

Lemma 4.3 does not give us the answer how to do that, just guarantees the existence. However, the proof of the lemma contains a recipe for that. If G_{32} is not in proper form, we swap columns in G_{32} and make the first column real using a J_{33}-unitary matrix Φ as described earlier. This gives us

$$\widetilde{G}_{32} = \Phi G_{32} P'_2 = \begin{bmatrix} \tilde{g}_{11} & \tilde{g}_{12} \\ \tilde{g}_{21} & \tilde{g}_{22} \\ \tilde{g}_{31} & 0 \end{bmatrix}. \tag{4.12}$$

Now, if $j_{11} = j_{33}$, by using $U_g([1,3],[1,3])$, we annihilate $\tilde{g}_{31}$. If $j_{11} \neq j_{33}$, we annihilate $\tilde{g}_{31}$ by using $U_g([2,3],[2,3])$, since $j_{22} = j_{33}$. This gives

$$G'_{32} = U_g \Phi G_{32} P'_2 = \begin{bmatrix} g'_{11} & g'_{12} \\ g'_{21} & g'_{22} \\ 0 & g'_{32} \end{bmatrix}. \tag{4.13}$$

Note that G'_{32} has still the same form as in **A2**, but now it is in a proper form, meaning $\det G'_1 \neq 0$ (see the proof of the lemma for that fact [15]). Remember, all the transformations we did on that small block must also be done on $\widetilde{G}^{(k)}$. These transformations are in the same fashion as before, so instead of using notation $\widetilde{\widetilde{G}}^{(k)}$, we will simply continue using $\widetilde{G}^{(k)}$. Therefore, without loss of generality, (4.4) still holds.

Lemma 4.4. *Let G_{42}, J_{44} be in form **A1** as in* (4.6)*. There exist permutation matrices P'_1, P'_2 and J'_{44}-unitary matrix U, such that*

$$G'_{42} := U^{-1}(P'_1)^* G_{42} P'_2, \quad U^* J'_{44} U = J'_{44}, \quad J'_{44} := (P'_1)^* J_{44} P'_1,$$

and G'_{42}, J'_{44} are in proper form, meaning $\det G'_1 \neq 0$.

Transforming G_{42} into G'_{42} is done the same way as (4.13), but instead of one Givens rotation, we will have to use two, since G_{42} has four nonzero elements in the first column and two in the second column.

4.2 Block J-unitary reduction

Reduction of proper forms has to be done with a different kind of J-unitary matrices which can annihilate blocks. There exist such block J-rotations and they can be represented as (see [18])

$$U = \begin{bmatrix} (I - YX)^{1/2} & Y \\ -X & (I - XY)^{1/2} \end{bmatrix}, \tag{4.14}$$

where X, Y are rectangular and the dimensions of X and Y^* are the same. Let YX and XY have the same orders as $\widetilde{J}_1^{(k)}$ and $\widetilde{J}_2^{(k)}$, respectively. If $Y = \widetilde{J}_1^{(k)} X^* \widetilde{J}_2^{(k)}$, then U is $\widetilde{J}^{(k)}$-unitary (see [18]) and

$$U^{-1} = \begin{bmatrix} (I - YX)^{1/2} & -Y \\ X & (I - XY)^{1/2} \end{bmatrix}.$$

The following lemmas provide existence and shape of U that we need for the further reduction of proper forms (for proof see [15]), but first we need to define some auxiliary variables

$$z := (\det G_1)^{-1} \tag{4.15}$$

$$a := j_{11} j_{33} |z|^2 (g_{21}^2 - g_{11}^2)(g_{32}^2 - g_{42}^2) \tag{4.16}$$

$$r := (1 + a)^{-\frac{1}{2}}. \tag{4.17}$$

These variables were defined for the most general form G_{42}. If the block is G_{32} of form (4.7), just put $g_{42} = 0$. The same goes for other forms.

Variable z is well defined, since we are dealing with proper forms. The definition of r, on the other hand, needs more clarification. Matrix A_2 (4.3) did not change, because J-unitary matrices preserve J-scalar products, therefore it is non-singular and indefinite. By the Sylvester's criterion,

$$0 > \det(A_2) = \det(G_{42}^* J_{44} G_{42}) = -j_{11} j_{33} (g_{21}^2 - g_{11}^2)(g_{32}^2 - g_{42}^2) \tag{4.18}$$

and (4.18) can be rewritten as

$$-(a + 1)/|z| < 0,$$

so $a > -1$, thus r is well defined.

Lemma 4.5. *Let G_{42} and J_{44} be in proper form of **A1** (4.6). There exists a block J_{44}-unitary matrix U defined by (4.14) such that*

$$U^{-1} G_{42} = G_{42} = \begin{bmatrix} G_1' \\ 0 \end{bmatrix}, \quad G_1' = \begin{bmatrix} g_{11}' & g_{12}' \\ g_{21}' & g_{22}' \end{bmatrix}.$$

The blocks of U are

$$X = rz\begin{bmatrix} g_{21}g_{32} & -g_{11}g_{32} \\ g_{21}g_{42} & -g_{11}g_{42} \end{bmatrix}, \quad Y = j_{11}j_{33}r\bar{z}\begin{bmatrix} g_{21}g_{32} & -g_{21}g_{42} \\ g_{11}g_{32} & -g_{11}g_{42} \end{bmatrix},$$

$$(I - XY)^{1/2} = I + \frac{j_{11}j_{33}r|z|^2(g_{21}^2 - g_{11}^2)}{1 + \sqrt{a+1}}\begin{bmatrix} -g_{32}^2 & g_{32}g_{42} \\ -g_{32}g_{42} & g_{42}^2 \end{bmatrix},$$

$$(I - YX)^{1/2} = I + \frac{j_{11}j_{33}r|z|^2(g_{32}^2 - g_{42}^2)}{1 + \sqrt{a+1}}\begin{bmatrix} -g_{21}^2 & g_{11}g_{21} \\ -g_{11}g_{21} & g_{11}^2 \end{bmatrix}.$$

Lemma 4.6. *Let G_{32} and J_{33} be in proper form of **A2*** (4.7) *or **B1*** (4.9)*. There exists a block J_{33}-unitary matrix U defined by* (4.14) *such that*

$$U^{-1}G_{32} = G_{32} = \begin{bmatrix} G_1' \\ 0 \end{bmatrix}, \quad G_1' = \begin{bmatrix} g_{11}' & g_{12}' \\ g_{21}' & g_{22}' \end{bmatrix}.$$

The blocks of U are

$$X = rz\begin{bmatrix} g_{21}g_{32} & -g_{11}g_{32} \end{bmatrix}, \quad Y = j_{11}j_{33}r\bar{z}\begin{bmatrix} g_{21}g_{32} \\ g_{11}g_{32} \end{bmatrix}, \quad (I - XY)^{1/2} = r,$$

$$(I - YX)^{1/2} = I + \frac{j_{11}j_{33}r|z|^2 g_{32}^2}{1 + \sqrt{a+1}}\begin{bmatrix} -g_{21}^2 & g_{11}g_{21} \\ -g_{11}g_{21} & g_{11}^2 \end{bmatrix}.$$

The end of this section needs one final comment. After all the transformations we did, we preserved the form (4.4)

$$G^{(k+1)} = Q^{(k)}P_1^* G^{(k)} P_2,$$
$$\widetilde{J}^{(k)} = P_1^* J P_1.$$

This is the same set of transformations as we have done with Householder reflectors, where we had the form (3.18), so again, the sequence (4.4) really satisfies a hyperbolic QR factorization (see [15]). The final matrices, as in the definition of the hyperbolic QR decomposition 2.3, would then be just the same as in the Householder case:

$$\widetilde{P}_1 = P_1^{(1)}P_1^{(2)}\dots P_1^{(n)}, \tag{4.19}$$
$$\widetilde{P}_2 = P_2^{(1)}P_2^{(2)}\dots P_2^{(n)}, \tag{4.20}$$
$$\widetilde{Q} = (P_1^{(n)})^*\dots(P_1^{(2)})^* Q^{(1)}P_1^{(2)}Q^{(2)}P_1^{(3)}Q^{(3)}\dots P_1^{(n)}Q^{(n)}, \tag{4.21}$$
$$\widetilde{A} = (\widetilde{P}_2)^* A \widetilde{P}_2. \tag{4.22}$$

Chapter 5

Pivoting

Until now, we saw that the earlier mentioned algorithms do well when we have a good pivoting strategy, but no particular strategy was yet presented. We will explain how the algorithm for the hyperbolic QR factorization is connected to the Hermitian indefinite factorization.

In the 1970s, Bunch, Kaufman and Parlett [4, 5, 6, 7, 8] developed a certain amount of algorithms for computing the Hermitian indefinite factorization (HIF), an indefinite analogue of the Cholesky factorization. Exactly those strategies will be useful to us to generate a consistent pivoting strategy which guarantees numerical stability.

5.1 Connections between HIF and hyperbolic QR

The Hermitian indefinite factorization of a matrix A has a form

$$A = PR^*JRP^*, \tag{5.1}$$

where P is a permutation matrix, R is block upper triangular with diagonal blocks of order 1 or 2, and J is a signature matrix. Its existence is due to the following proposition.

Proposition 5.1. *Let A be Hermitian and non-singular. Then A can be factored as*

$$A = P^T LDL^* P, \tag{5.2}$$

where P is a permutation matrix, D is Hermitian with diagonal blocks of order 1 or 2 and L is a lower triangular matrix with ones on the diagonal if D has a 1×1 block, or a diagonal block of I_2 if D has a 2×2 block.

Since we can compute an eigenvalue decomposition of D as $D = S^* \sqrt{|\Delta|} J \sqrt{|\Delta|} S$, where J is the signature matrix of eigenvalues, then (5.2) can be rewritten as (5.1).

Now let A be given as $A = G^*JG$ by G and J. If we have a hyperbolic QR factorization of G, as in definition 2.3, we instantly have a Hermitian indefinite factorization of A (5.1),

$$A = (P_1QRP_2^*)^*J(P_1QRP_2^*) \quad \Rightarrow \quad A = P_2R^*J'RP_2^*, \quad Q^*J'Q = J', \quad J' = P_1^*JP_1. \tag{5.3}$$

First we will explain an outline of the method how to compute the Hermitian indefinite factorization. We will not go into details, yet explain the basic idea which will give us a pivoting strategy we can use for our hyperbolic QR factorization, since the connection between elements of A and G^*JG is simply

$$a_{ij} = g_i^*Jg_j,$$

where g_i is the ith column of G. Second, the pivoting strategy will provide numerical stability, because the strategy will be derived from the fact that we want to minimize the pivot growth in every step of computing the Hermitian indefinite factorization. Third, an explanation that the algorithm for Hermitian indefinite factorization and the algorithm for the hyperbolic QR factorization provide almost the same result will be given, if the same pivoting strategy is used. The main difference is observed in the case of 2×2 pivots, where zeroes on the diagonal in matrix A can be represented as different difference of squares, i.e., if

$$G = \begin{bmatrix} g^2 \\ g^2 \end{bmatrix}, \quad J = \operatorname{diag}(1,-1),$$

then $A = G^*JG = 0$ for all elements g.

An outline for computing the Hermitian indefinite factorization

The proof of proposition 5.1 (see, for example [17, chapter 5]) gives us a purely theoretical approach on how to compute (5.1). It is purely theoretical, because it is potentially numerically unstable. First if there exists a nonzero element on the diagonal of A, we put in place $(1,1)$. If all diagonal elements are zero, then there exists a off-diagonal element a_{ij} in A which is nonzero, otherwise A would be singular. Now we take the 2×2 submatrix A_{11} as

$$A_{11} = \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix}$$

and place it in position $(1:2, 1:2)$. The Hermitian structure of A is then left intact. Thus,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{12}^* & A_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ L_{12} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & L_{21}^* \\ 0 & I \end{bmatrix}, \tag{5.4}$$

Where A_{11} is a 1×1 or a 2×2 matrix and the other blocks are

$$L_{21} = A_{12}^*A_{11}^{-1}, \quad S = A_{22} - L_{21}A_{11}L_{21}^* = A_{22} - A_{12}^*A_{11}^{-1}A_{12}.$$

The same technique is then repeated on S.

A following example taken from [8] shows why this strategy is numerically unstable.

Example 5.2. *Let $Ax = b$ be a linear 2×2 system, where*

$$A = \begin{bmatrix} \varepsilon & 1 \\ 1 & \varepsilon d \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad 0 < \varepsilon \ll d \leq 1.$$

The condition number of A,

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = \frac{(1+\varepsilon)^2}{1 - d\varepsilon^2} = 1 + \varepsilon + \mathcal{O}(\varepsilon^2),$$

is a fairly small number, so the solution to the linear system can be computed with small relative errors. The Hermitian indefinite factorization, as in (5.1)*, is computed as*

$$A = \begin{bmatrix} \varepsilon^{1/2} & 0 \\ \varepsilon^{-1/2} & (1/\varepsilon - d\varepsilon)^{1/2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \varepsilon^{1/2} & (1/\varepsilon - d\varepsilon)^{1/2} \\ 0 & \varepsilon^{-1/2} \end{bmatrix}.$$

If ε is very small, then $(1/\varepsilon - d\varepsilon)^{1/2} \approx \varepsilon^{-1/2}$. Taking this into consideration, the solution $\hat{x}$ derived from this factorization is then

$$\hat{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

while the exact solution is

$$x = \frac{1}{1 - d\varepsilon^2} \begin{bmatrix} -d\varepsilon \\ 1 \end{bmatrix}$$

and the relative error is

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq d\varepsilon.$$

However, if we solve the same system with the Gaussian elimination with partial pivoting, we get

$$\tilde{x} = \begin{bmatrix} -d\varepsilon \\ 1 \end{bmatrix},$$

with a relative error

$$\frac{\|x - \tilde{x}\|_\infty}{\|x\|_\infty} \leq d\varepsilon^2,$$

which is considerably better.

Pivot growth

All the results stated in this subsection can be found in [17, chapters 6–7,9–11]. First, let us define some auxiliary variables. Let $A \in \mathbb{F}^{n\times n}$ be non-singular and Hermitian.

$$\mu_0 = \max_{i,j} |a_{ij}|, \quad \mu_1 = \max_i |a_{ii}|, \quad \nu = |a_{11}a_{22} - a_{12}^2|. \tag{5.5}$$

Lemma 5.3. *Suppose a pivot of order* 1 *is chosen which is put at the position* $(1,1)$*, in other words* $\mu_1 = |a_{11}| \neq 0$*. Then*

1. $\max_i |(L_{21}^*)_{1i}| \leq \dfrac{\mu_0}{\mu_1}$,
2. $\max_{i,j} |S_{ij}| \leq \mu_0 \left(1 + \dfrac{\mu_0}{\mu_1}\right)$.

If a pivot of order 2 *is chosen, and* A_{11} *from* (5.4) *is put at the position* $(1:2, 1:2)$*, and if* $|\det A_{11}| = \nu \geq \mu_0^2 + \mu_1^2$ *holds, then*

1. $\max_{i,j} |(L_{21}^*)_{ij}| \leq \mu_0 \dfrac{\mu_0 + \mu_1}{\nu}$, *for* $i = 1, 2$,
2. $\max_{i,j} |S_{ij}| \leq \mu_0 \left(1 + \dfrac{2}{1 - \frac{\mu_0}{\mu_1}}\right)$.

We now want to minimize the pivot growth of matrix S in (5.4). Suppose that $A^{(k)}$ is the submatrix of order k which we yet need to factorize and let $F_i^{(k)}$, $i = 1, 2$, denote the pivot growth of matrix $S^{(k)}$ depending on the strategy. Lemma 5.3 gives

$$F_1^{(k)} = 1 + \frac{\mu_0^{(k)}}{\mu_1^{(k)}}, \quad F_2^{(k)} = 1 + \frac{2}{1 - \frac{\mu_0^{(k)}}{\mu_1^{(k)}}}.$$

Suppose that we have a strategy S_α, $0 < \alpha < 1$ which is a pivot of order 1 if and only if

$$\frac{\mu_1^{(k)}}{\mu_0^{(k)}} \geq \alpha.$$

This strategy S_α gives

$$F_1^{(k)} \leq 1 + \frac{1}{\alpha}, \quad F_2^{(k)} \leq 1 + \frac{2}{1-\alpha}.$$

The idea is to compare the pivot growth when two steps of order 1 are performed and one step of order two. In other words, we need to compare $F_1^{(k)}$ and $F_1^{(k+1)}$ with $F_2^{(k)}$. Now define

$$G(\alpha) = \max\left\{\left(1 + \frac{1}{\alpha}\right)^2, 1 + \frac{2}{1-\alpha}\right\}.$$

We are looking for $\alpha \in \langle 0, 1\rangle$ that minimizes $G(\alpha)$. Since both functions are continuous, the first one is monotone decreasing and the second one is monotone increasing for $\alpha \in \langle 0, 1\rangle$, we get that the α_0 which minimizes G is given as an intersection between these two functions,

$$\left(1 + \frac{1}{\alpha}\right)^2 = 1 + \frac{2}{1-\alpha},$$

from where we get

$$\alpha_0 = \frac{1 + \sqrt{17}}{8} \approx 0.6403882\ldots \tag{5.6}$$

The second part of the Lemma 5.3 required that $|\det A_{11}| = \nu \geq \mu_0^2 + \mu_1^2$, so this discussion was made under that assumption.

Pivoting strategy

The first attempt of a consistent pivoting strategy is very obvious. Let $A^{(k)}$ be the unreduced part of A in step k, in other words $S = A^{(k)}$. The steps of a pivoting strategy, so-called the complete pivoting strategy, that minimizes pivot growth are:

1. Find $\mu_0 = \max_{i,j} |a_{ij}^{(k)}| = |a_{pl}|$ and $\mu_1 = \max_j |a_{jj}^{(k)}| = |a_{rr}^{(k)}|$.
2. If $\mu_1^{(k)} \geq \alpha_0 \mu_0^{(k)}$ do pivot 1, otherwise do pivot 2.
3. In the case of pivot 1, put element $a_{rr}^{(k)}$ at the position $(1, 1)$
4. In the case of pivot 2, put matrix $A_{11}^{(k)}$ at the position $(1:2, 1:2)$, defined as

$$A_{11}^{(k)} = \begin{bmatrix} a_{pp}^{(k)} & a_{pl}^{(k)} \\ a_{lp}^{(k)} & a_{ll}^{(k)} \end{bmatrix}.$$

It can be shown that, if we choose this strategy, then $\nu \geq \mu_0^2 + \mu_1^2$ is satisfied.

However, this obvious strategy includes going through all elements of $A^{(k)}$, therefore we need to compute the whole submatrix $A^{(k)} = (G_{22}^{(k)})^* J_2 G_{22}^{(k)}$ in every step. This consumes a lot of computing time, so instead of the complete pivoting strategy that searches the rest of the matrix for pivots, somewhat relaxed pivoting strategy, so-called partial pivoting strategy, exists and better suits our situation (see, for example [7]).

The steps of partial pivoting for an Hermitian indefinite factorization of a yet unreduced submatrix $A^{(k)}$ are:

1. Determine $\lambda = |a_{r1}^{(k)}| = \max_{2 \leq i \leq k} |a_{i1}^{(k)}|$.
2. If $|a_{11}^{(k)}| \geq \alpha_0 \lambda$, take $a_{11}^{(k)}$ as pivot.
3. Determine $\sigma = \max_{1 \leq i \leq k} |a_{ir}^{(k)}|$, $i \neq r$.

4. If $|a_{11}^{(k)}|\sigma \geq \alpha_0 \lambda^2$, take $a_{11}^{(k)}$ as pivot.
5. If $|a_{rr}^{(k)}| \geq \alpha_0 \sigma$, take $a_{rr}^{(k)}$ as pivot.
6. Otherwise, do pivot of order 2 where

$$A_{11}^{(k)} = \begin{bmatrix} a_{11}^{(k)} & a_{1r}^{(k)} \\ a_{r1}^{(k)} & a_{rr}^{(k)} \end{bmatrix}.$$

5.2 Pivoting in hyperbolic QR

As we saw, the connection between these two factorizations is

$$a_{ij} = g_i^* J g_j,$$

more precisely, if

$$A = \begin{bmatrix} I & 0 \\ L_{12} & I \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & L_{21}^* \\ 0 & I \end{bmatrix} \tag{5.7}$$

and if $A = G^* J G$, where

$$G = \begin{bmatrix} G_{11} & G_{12} \\ 0 & G_{22} \end{bmatrix}, \tag{5.8}$$

then A also has the following form

$$A = \begin{bmatrix} I & 0 \\ G_{12}^* (G_{11}^*)^{-1} & I \end{bmatrix} \begin{bmatrix} G_{11}^* J_1 G_{11} & 0 \\ 0 & G_{22}^* J_2 G_{22} \end{bmatrix} \begin{bmatrix} I & G_{11}^{-1} G_{12} \\ 0 & I \end{bmatrix}. \tag{5.9}$$

From here, we see that

$$A^{(k)} = S = (G_{22}^{(k)})^* J_2 G_{22}^{(k)}$$

in every step. Note that G_{11}^{-1} is always well defined. It is either block of order 1 in the case of a single pivot, where a non-degenerate column was chosen, therefore $G_{11} \neq 0$, or a block of order 2 in case of 2 pivot columns, where $G_{11} = G_1'$ and G_1' is the 2×2 matrix in the proper form.

This is the main reason why in the Definition 2.3 of the hyperbolic QR matrix $G^* J G$ is non-singular. Now we can see that we cannot avoid 2×2 blocks on the diagonal of G.

If the partial pivoting strategy choses a single element a_{ii} as pivot, then we do a reduction of the first column in $\widetilde{G}_{22}^{(k)}$ by the J-unitary Householder reflector. The pivot column is then $\tilde{g}_i$, located as the first column in $\widetilde{G}_{22}^{(k)}$. Note that $\tilde{g}_i$ is then nondegenerate, so we construct the reflector as before. If pivot of order 2 is chosen, then the reduction would be done with the Givens-like algorithm.

There is one thing left unclear: all the conditions we imposed on A_2 from (4.3) when we were explaining which two columns we consider 'good' for our Givens algorithm.

The non-singularity and nonzero off-diagonal elements are clear, since $A_2 = A_{11}$ and A_{11} satisfies the same. The indefiniteness, which as we saw was crucial for building the J-unitary matrices U (4.14), needs justification. First, note that $\det A_{11}$ is a real number. This is clear from the definition (4.3) of A_2. If we follow the partial pivoting strategy, the pivot of order 2 is chosen if

$$|a_{11}| < \alpha_0 |a_{r1}|, \tag{5.10}$$
$$|a_{11}|\sigma < \alpha_0 |a_{r1}|, \tag{5.11}$$
$$|a_{rr}| < \alpha_0 \sigma, \tag{5.12}$$
$$|a_{r1}| \neq 0. \tag{5.13}$$

If $\det A_{11} \geq 0$, then

$$|a_{11}||a_{rr}| \geq |a_{r1}|^2.$$

But,

$$|a_{r1}|^2 \leq |a_{11}||a_{rr}| < |a_{11}|\alpha_0 \sigma < |a_{11}|\alpha_0 \frac{\alpha_0 |a_{r1}|^2}{|a_{11}|} = \alpha_0^2 |a_{r1}|^2$$

and, since $|a_{r1}| \neq 0$, this is a contradiction with the fact that $1 < \alpha_0^2$, therefore A_{11} is indefinite and so is A_2.

This shows why a consistent pivoting strategy for the indefinite QR would not break down if $A = G^* J G$ is non-singular.

Chapter 6

Implementation

In this chapter, we will sum up the algorithm in form of code. Fortran open source routines BLAS [3] and LAPACK [11] were used as much as possible. These routines, or functions, have optimal way of performing linear algebra operations. In order to be optimal, BLAS and LAPACK, assume all matrices and vectors are in a column major layout, meaning that a matrix $G \in \mathbb{C}^{m \times n}$ is stored in one array and in the first m spaces is the first column, in the second m spaces is the second column, etc. We then access an element G_{ij} as $G[i + M * j]$.

First a sequential implementation will be presented after which we will discuss which regions were parallelised and how.

6.1 Sequential implementation

Assume we have a matrix G of the dimension $M \times N$ as an input, where $M \gg N$, and the signature matrix J of the dimension $M \times M$. After allocating memory and reading the data, we enter a for loop.

We consider a variable x to be zero if $|x| <$ EPSILON, where EPSILON is a constant from library *float.h* defined as the minimal positive number such that 1.0+EPSILON $\neq$ 1.0.

```
1  for(int k = 0; k < N; ++k){
2
3    // ------------------- PARTIAL PIVOTING ------------------
4
5    compute Akk;
6    if(k == N-1) goto PIVOT_1;
7
8    find lambda;
9    if(cabs(Akk) >= ALPHA * lambda) goto PIVOT_1;
10
11   find sigma;
12   if(cabs(Akk) * sigma >= ALPHA * lambda * lambda) goto PIVOT_1;
13
14   compute Arr;
15   if(cabs(Arr) >= ALPHA * sigma){
```

```
16    swap columns k and r in G;
17    update Pcol;
18    Akk = Arr;
19    goto PIVOT_1;
20  }
21
22  // ------------------ START OF PIVOT2 ------------------
23
24  if(r != k+1){
25    swap columns k+1 and r in G;
26    update Pcol;
27  }
28
29  make the 1st column in G[k:M, k:N] real;
30
31  find the first idx >= k so that G[idx + M*k] != 0;
32  if(idx != k){
33    swap rows k and idx;
34    update Prow;
35    swap diagonal els k and idx in J;
36  }
37  do plane rotations with G[k + M*k] on all els with sign J[k];
38
39  find the first idx so that J[idx] = -J[k] and G[idx + M*k] != 0;
40  if(idx != k+1){
41    swap rows k+1 and idx;
42    update Prow;
43    swap diagonal els k+1 and idx in J;
44  }
45  do plane rotations with G[k+1 + M*k] on all els with sign J[k+1];
46
47
48  // keep track of forms, so that
49  // we can determine the cases A1, A2, A3, B1, B2
50  int kth_nonzeros = number of nonzero els in G[k:M, k];
51
52  make the 1ts column in G[(k+kth_nonzeros):M, (k+1):N] real;
53
54  find the first idx >= (k+kth_nonzeros) so that G[idx + M*(k+1)] != 0;
55  if(idx != k + kth_nonzeros){
56    swap rows k+kth_nonzeros and idx;
57    update Prow;
58    swap diagonal els k+kth_nonzeros and idx in J;
59  }
60  do plane rotations with G[(k+kth_nonzeros) + M*(k+1)] on all els with
61    sign J[k+kth_nonzeros];
62
63  find the first idx > (k+kth_nonzeros) so that
64    J[idx] = -J[k+kth_nonzeros] and G[idx + M*(k+1)] != 0;
65
66  if(idx != k+kth_nonzeros+1){
67    swap rows k+kth_nonzeros+1 and idx;
68    update Prow;
69    swap diagonal els k+kth_nonzeros+1 and idx in J;
70  }
71  do plane rotations with G[(k+kth_nonzeros+1) + M*(k+1)] on all els with
72    sign J[k+kth_nonzeros+1];
73
74  // keep track of forms
75  int kkth_nonzeros = number of nonzero els in
```

```
76      G[(k+kth_nonzeros):M, k+1];
77
78    // condition A3
79    if(kth_nonzeros == 2 && kkth_nonzeros == 0) goto LOOP_END;
80
81    // condition B2
82    if(kth_nonzeros == 1 && kkth_nonzeros == 1) goto LOOP_END;
83
84    // handle the A1 form
85    if(kth_nonzeros == 2 && kkth_nonzeros == 2){
86
87      // check if its a proper form
88      if( cabs(det(G[k:k+1, k:k+1])) < EPSILON){
89        swap columns k and k+1 in G;
90        update Pcol;
91
92        make the 1st and the 2nd row in G[k:M, k:N] real;
93        do a plane rotation with G[k + M*k] on an el with sign J[k];
94        do a plane rotation with G[k+1 + M*k] on an el with
95          sign J[k+1];
96      }
97
98
99      // do the final reduction
100     compute a, z, r;
101     fill a 4x4 matrix U^{-1};
102     G[k:k+4, k:N] = U^{-1} * G[k:k+4, k:N];
103
104     k = k+1;
105     goto LOOP_END;
106   }// end of case A1
107
108
109   // handle forms A2 and B1
110   else if(kth_nonzeros == 2 && kkth_nonzeros == 1 ||
111           kth_nonzeros == 1 && kkth_nonzeros == 2){
112
113     // check if its a proper form, B1 i already in proper!
114     if( cabs(det(G[k:k+1, k:k+1])) < EPSILON ){
115       swap columns k and k+1 in G;
116       update Pcol;
117
118       make the 1st and the 2nd row in G[k:M, k:N] real;
119       do plane rotations with one of the first two rows in G[k:M, k:N]
120         to eliminate G[k+2 + M*k];
121     }
122
123     // do the final reduction
124     compute a, z, r;
125     fill a 3x3 matrix U^{-1};
126     G[k:k+3, k:N] = U^{-1} * G[k:k+3, k:N];
127
128     k = k+1;
129     goto LOOP_END;
130   }//end of cases A2 and B1
131
132   // if here A is close to singular
133   exit(-4);
134
135   // ------------------- START OF PIVOT1 -------------------
```

```
136   PIVOT_1:
137
138   if( Akk > 0 && J[k] < 0){
139     find the first idx > k so that J[idx] = 1;
140     swap rows k and idx in G;
141     update Prow;
142     swap diagonal elemnts k and idx in J;
143   }
144
145   else if( Akk < 0 && J[k] > 0){
146     find the first idx > k so that J[idx] = -1;
147     swap rows k and idx in G;
148     update Prow;
149     swap diagonal elemnts k and idx in J;
150   }
151
152   // compute reflector constant H_sigma
153   double complex H_sigma = 1;
154   if(cabs(G[k+M*k]) >= EPSILON) H_sigma = -G[k+M*k] / cabs(G[k+M*k]);
155
156   fill f[k:M] = (csqrt(cabs(Akk)) * H_sigma, 0, ... , 0);
157   tempf = f;
158   f[k:M] = f[k:M] - G[k, k:M];
159
160   double complex wJw = Akk + J[k] * (cabs(Akk) + 2 * csqrt(cabs(Akk)) * cabs(G[k + M*k]));
161
162   // make the reflector H[k:M, k:M]
163   for(j = k; j < M; ++j)
164     for(i = k; i < M; ++i)
165       H[i + M*j] = -2 * f[i] * conj(f[j]) * J[j] / wJw;
166
167   for(i = k; i < M; ++i) H[i + M*i] += 1;
168
169   //apply the reflector on a submatrix
170   G[k:M, (k+1):M] = H[k:M, k:M] * G[k:M, (k+1):M];
171   G[k:M, k] = tempf;
172
173   LOOP_END: continue;
174 }
```

Because we are storing matrices in a column order, it is wise to fill matrix H as we did in lines 160–162. Accessing array memory consecutively boosts performance. Note that in the case of column or row swaps, just the nonzero pieces of the vectors should be swapped. If the corresponding elements in vectors are zero, we gain nothing from swapping them.

6.2 Parallelisation

Almost everything can be parallelised in this algorithm. Parallelisation means that multiple processes are executed simultaneously.

For parallelisation we used OpenMP, an application programming interface that supports shared memory multiprocessing in C [14]. Shared memory is a part of memory which can be accessed by multiple threads at the same time and for OpenMP it is the L3 cache. The temporary view of memory allows the thread to cache variables and thereby to avoid

going to memory for every reference to a variable. Each thread also has access to another type of memory that must not be accessed by other threads, called the private memory known as L1 cache. L1 is the 'fast' memory, but with very limited storage, while L3 is the largest and the 'slowest' (if we do not include RAM).

Cache optimization is important for performance when using OpenMP. For example, when a parallel region starts, the shared variables are stored into L3 cache. If they do not fit in L3, since the memory is much smaller than RAM, a part of it is stored. If a thread constantly needs a call to the L3 cache, then, first a request to L2 cache is sent, then if data is not there, a request to the L3 cache is sent. Then data is sent to L1 cache over L2 cache, which takes more time. Therefore, we should avoid cache misses as much as possible. Optimizing cache performance in our case means that a single thread should not 'jump' too far across memory.

For boosting BLAS and LAPACK, an Intel Math kernel library was used [12]. Basically, when a program is linked to the MKL library, a routine gets a team of threads.

OpenMP and MKL allow a user to specify the number of threads. Since we used nested parallelism (parallel MKL routines in parallel OpenMP regions), we have to be aware of overheads. Overheads occur when we have too many threads with no work, an opposite of doing useful work.

When using OpenMP, overheads can occur when two threads update individual elements in the shared memory which are close to each other, more precisely in the same cache line (for more information see [9]). This situation is called false sharing. Since we have to update our matrix G frequently, and G will be a shared variable in parallel regions, we should not send intertwined chunks of G to threads if we can send it as a compact block. Mainly, MKL will handle the division into blocks and we just need to make sure we do not assign to many threads to a specific region.

Let us now take a look on how to boost performance of our sequential code.

Reduction

First, all sums should be performed using the reduction technique.

```
1 int nthreads = (M-k)/D > omp_get_max_threads() ? (M-k)/D : omp_get_max_threads();
2 if ((M-k)/D == 0) nthreads = 1;
3
4 #pragma omp parallel for reduction(+:Akk) num_threads( nthreads )
5 for(i = k; i < M; ++i) Akk += conj(G[i+M*k]) * J[i] * G[i+M*k];
```

For example, in figure 6.1, ideally the first four threads would form their 'mini' sums: 40, 29, 35, 30. After that, other two threads would compute 69 and 65 from the previous 'mini' sums, and at last, one thread would sum it all up into 134.

Reduction was also implemented in annihilation of elements using Givens rotations. One process had a job of annihilating elements of class 1 with a team of threads for reduc-

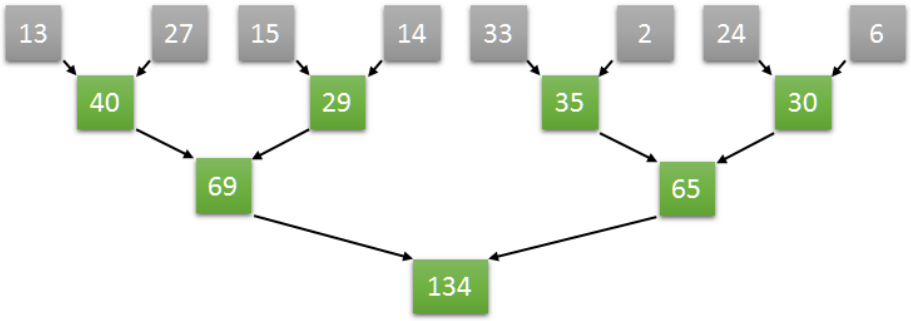


Figure 6.1: Example of reduction on a sum.

tion and the other did the same thing on elements of class -1. This way, an annihilation of one column has the complexity $\mathcal{O}(\log M)$. While performing reduction, it is handy for OpenMP to use variables loaded into cache as much as possible, so we made the kth column real and performed reduction in the same parallel region. An example of code using a reduction technique with Givens rotations is given below.

```
#pragma omp parallel num_threads(2)
{
// first thread kills positives
if(omp_get_thread_num() == 0){
  int offset;
  for(offset = 1; offset < np; offset *= 2){

    int nthreads_loc = np/(2*offset);
    if(nthreads_loc == 0) nthreads_loc = 1;
    else if ( nthreads_loc > omp_get_max_threads()/2)
                nthreads_loc = omp_get_max_threads()/2;

    #pragma omp parallel for num_threads( nthreads_loc )
    for(i = 0; i < np - offset; i += 2*offset){

      int mkl_nthreads = (N-k)/D > mkl_get_max_threads()/2 ?
                (N-k)/D : mkl_get_max_threads()/2;
      if((N-k)/D == 0) mkl_nthreads = 1;
      mkl_set_num_threads(mkl_nthreads);

      // G[p[i], k] destroys G[p[i+offset], k]
      // first if kth column isnt real, make it real

      if( cimag(G[p[i] + M*k]) != 0){
        double complex scal = conj(G[p[i] + M*k]) / cabs(G[p[i] + M*k]);
        G[p[i] + M*k] = cabs(G[p[i] + M*k]);
        int Nk = N - k - 1;
        zscal(&Nk, &scal, &G[p[i] + M*(k+1)], &M);
      }

      if( cimag(G[p[i+offset] + M*k]) != 0){
        double complex scal = conj(G[p[i+offset] + M*k]) / cabs(G[p[i+offset] + M*k]);
        G[p[i+offset] + M*k] = cabs(G[p[i+offset] + M*k]);
```

```
34                int Nk = N - k - 1;
35                zscal(&Nk, &scal, &G[p[i+offset] + M*(k+1)], &M);
36          }
37          double c;
38          double complex s;
39          double complex eliminator = G[p[i] + M*k];
40          double complex eliminated = G[p[i + offset] + M*k];
41          zrotg(&eliminator, &eliminated, &c, &s);
42
43          // apply the rotation
44          int Nk = N-k;
45          zrot(&Nk, &G[p[i] + M*k], &M, &G[p[i + offset] + M*k], &M, &c, &s);
46          G[p[i + offset] + M*k] = 0;
47        }
48      }
49    }//end of if
50    // second thread kills negatives
51    else{
52      int offset;
53      for(offset = 1; offset < nn; offset *= 2){
54
55        int nthreads_loc = nn/(2*offset);
56        if(nthreads_loc == 0) nthreads_loc = 1;
57        else if ( nthreads_loc > omp_get_max_threads()/2)
58                  nthreads_loc = omp_get_max_threads()/2;
59
60        #pragma omp parallel for num_threads( nthreads_loc )
61        for(i = 0; i < nn - offset; i += 2*offset){
62
63          int mkl_nthreads = (N-k)/D > mkl_get_max_threads()/2 ?
64                    (N-k)/D : mkl_get_max_threads()/2;
65          if((N-k)/D == 0) mkl_nthreads = 1;
66          mkl_set_num_threads(mkl_nthreads);
67
68          // G[n[i], k] destroys G[n[i+offset], k]
69          // make them real
70
71          if( cimag(G[n[i] + M*k]) != 0){
72            double complex scal = conj(G[n[i] + M*k]) / cabs(G[n[i] + M*k]);
73            G[n[i] + M*k] = cabs(G[n[i] + M*k]);
74            int Nk = N - k - 1;
75            zscal(&Nk, &scal, &G[n[i] + M*(k+1)], &M);
76          }
77
78          if( cimag(G[n[i+offset] + M*k]) != 0){
79            double complex scal = conj(G[n[i+offset] + M*k]) / cabs(G[n[i+offset] + M*k]);
80            int Nk = N - k - 1;
81            zscal(&Nk, &scal, &G[n[i+offset] + M*(k+1)], &M);
82          }
83
84          double c;
85          double complex s;
86          double complex eliminator = G[n[i] + M*k];
87          double complex eliminated = G[n[i + offset] + M*k];
88          zrotg(&eliminator, &eliminated, &c, &s);
89
90          // apply the rotation
91          int Nk = N-k;
92          zrot(&Nk, &G[n[i] + M*k], &M, &G[n[i + offset] + M*k], &M, &c, &s);
93          G[n[i + offset] + M*k] = 0;
```

```
94      }
95    }
96 }//end of else
97 }//end of parallel region
```

As we can see, an additional array n was needed for mapping which row had signs -1 in J and an additional array p for mapping positive signs. The predefined variable D was used to specify a chunk size. It helps us calculate the number of threads, so that each thread does work on a chunk of size D.

Division into blocks

We can also boost performance when copying arrays. Since we have a lot of row and column permutations, therefore a lot of column swapping. This is done by slicing a vector in chunks of size D and then calling a team of threads so that every chunk is copied simultaneously. Fortunately, MKL cares for such things.

```
1 int mkl_nthreads = M/D > mkl_get_max_threads() ? M/D : mkl_get_max_threads();
2 if(M/D == 0) mkl_nthreads = 1;
3 mkl_set_num_threads(mkl_nthreads);
4 zswap(&M, &G[M*pivot_r], &inc, &G[M*k], &inc);
```

The algorithm also has a multiplication of two matrices, the reflector H and the submatrix in G. Jobs here are also divided between threads in a way that each thread gets a chunk of memory, also handled with MKL. The block-like division is true as long as OMP_SCHEDULE is set to static and it is static by default.

For details, see full code of both implementations on github [2].

6.3 Intel Xeon Phi 7210

The program was tested on a Intel's Xeon Phi 7210 [10]. This processor has characteristic memory structure: it has a high-bandwidth memory with a data transfer rate over 400 GB/s (memory bandwidth is the rate at which data can be read and stored). HBM can be used either as a last-level cache, or as addressable memory. The *Flat* mode uses the entirety of HBM as addressable memory, whereas *Cache* mode uses the entirety of HBM as cache. With *Hybrid* mode, a portion of the HBM is used as addressable memory and the rest is used as cache. In the *Flat* mode, all memory allocated with mkl_malloc() is allocated as HBM by default (if it fits). In this case, the shared memory is the L2 cache of size 32 MB, whereas the HBM is 16 GB, in our case, meant for array allocation.

We used HDM in the *Flat* mode. Each core has 512 KB of L2 cache, out of total 64 cores, and each core can have up to 4 threads. Binding memory allocation on HMB is done during runtime as

```
1 numactl --membind 1 make ./...
```

which binds the memory of our application to node number 1, which is in our case a node with HBM.

MKL can have a pre-set number of threads or it can be left to the system to decide how many threads it will be given, but while using MKL in nested parallelism with OpenMP, the MKL_DYNAMICS variable needs to be set to false, otherwise, MKL would perform sequentially in parallel regions (we are using that in the reduction process with Givens). We also set the maximum level of active parallel levels to 3 (again because of the reduction process with Givens). MKL_DYNAMICS was true for regions when we were not in OpenMP parallel regions, since then the system optimizes the workload for BLAS and LAPACK routines by itself. In nested areas, a mkl_get_max_threads() function was used. This function returns the number of OpenMP threads available for Intel MKL to use in internal parallel regions, so if all OpenMP threads are busy, the MKL regions will perform singlethreadedly. For more information on the variables and environment see [1, 13]

Results

The following numbers are the results of our implementation on Xeon Phi 7210. Matrices were randomly generated. Variable $pivot_i$ stands for the count of pivots of order i, where t_i is the time needed for them. Time t_{strat} is the time spent on choosing a strategy. The relative error is rel. err. $= \|A - \widetilde{G}^* J' \widetilde{G}\|_2 / \|A\|_2$. Time spent in each part of the algorithm for matrices is 6.3. The best choice for chunk size D turned out to be $D = 64$.

Table 6.1: Table of results for different size problems.

M	N	t(s)	$pivot_1$	t_1(s)	$pivot_2$	t_2(s)	t_{strat}(s)	rel. err. $\cdot 10^{-12}$
4000	1000	88.12	254	39.06	373	14.28	34.78	2.52039
5000	2000	303.28	568	129.65	716	36.31	137.32	6.95133
7000	3000	1097.29	732	320.90	1134	466.81	309.57	1.02424
9000	6000	6444.14	1418	1181.70	2291	4014.95	1246.9	2.45308

Further work

One way to improve our implementation is to try computing Akk not with reduction, but with BLAS' *zdot* function which computes the dot product x^*y. The rule of thumb is: BLAS with MKL usually does it better. We should then build a temporary vector with each element scaled with $+1$ or -1 depending on the corresponding sign in J and then use the *zdot* routine. Since we also need an index of the maximal element by absolute value r, we can use OpneMP's user defined reduction to do so. In this approach we do not need critical regions which were necessary to use in order to store the maximum value for

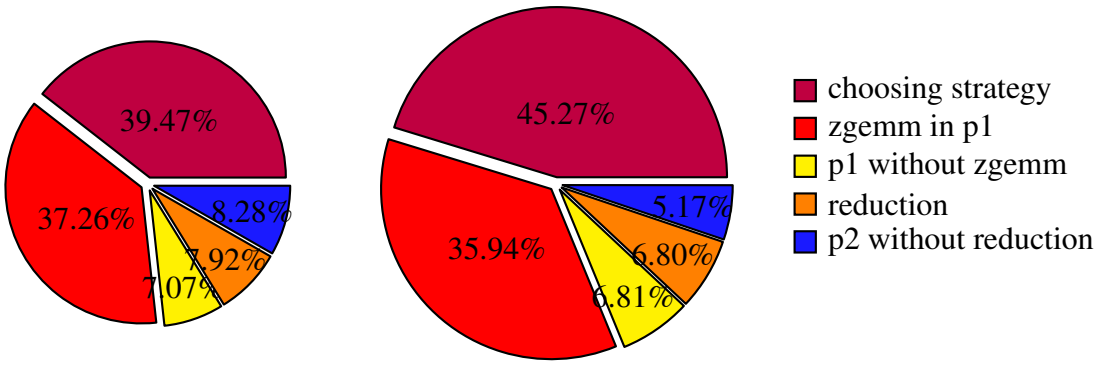


Figure 6.2: Portion of time spent for matrices in 6.1 of sizes 4000×1000 and 5000×2000, respectively.

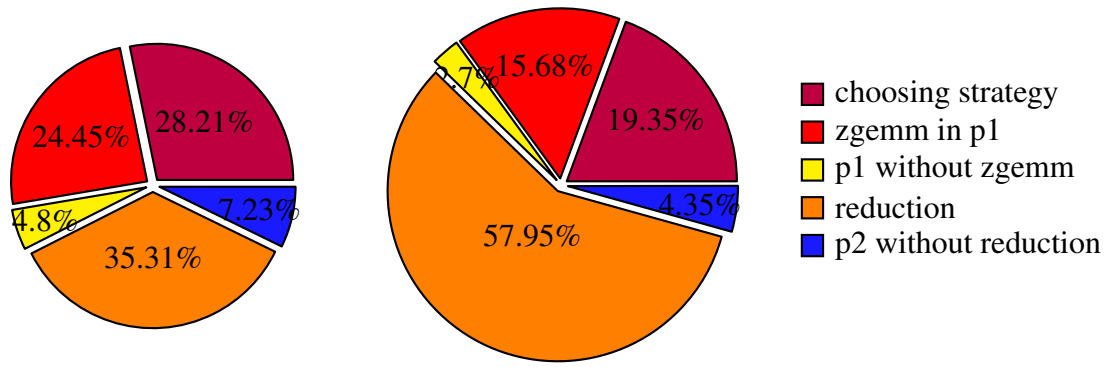
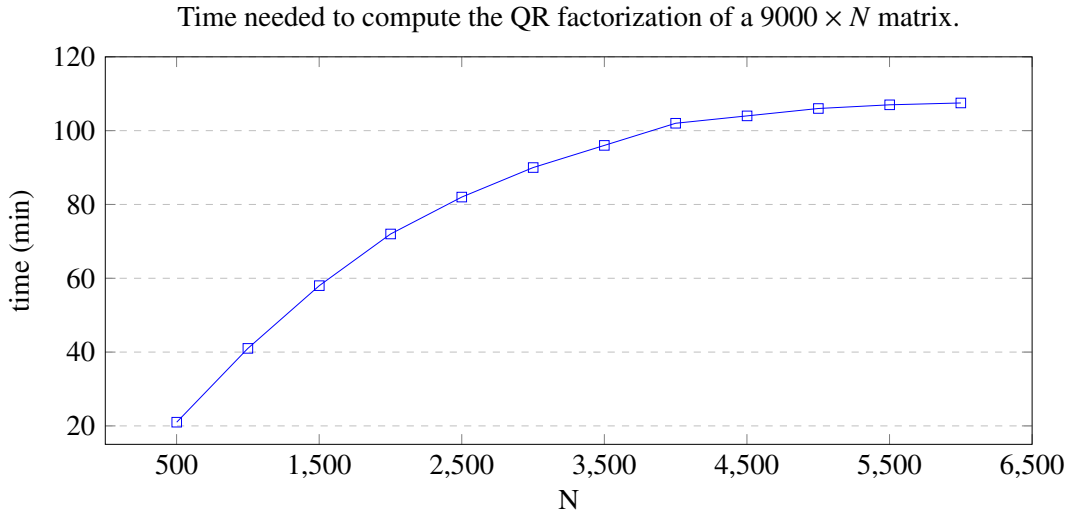


Figure 6.3: Portion of time spent for matrices in 6.1 of sizes 7000×3000 and 9000×6000, respectively.

finding λ in the partial pivoting strategy and its index. The segment of the code as it is in this moment, containing critical regions, is listed below.

```
1  // find pivot_lambda
2
3  #pragma omp parallel for num_threads((int) csqrt(nthreads) )
4  for(i = k+1; i < N; ++i){
5    double complex Aik = 0;    //Aik = gi* J gk, but on a submatrix G[k:M, k:N]
6
7    #pragma omp parallel for reduction(+:Aik) num_threads( (int)csqrt(nthreads) )
8    for(j = k; j < M; ++j) Aik += conj(G[j+M*i]) * J[j] * G[j+M*k];
9
10   #pragma omp critical
11   if(pivot_lambda < cabs(Aik)){
12     pivot_lambda = cabs(Aik);
13     pivot_r = i;
14   }
```




```
15 }
16
17 if(cabs(Akk) >= ALPHA * pivot_lambda) goto PIVOT_1;
```

The critical regions do not allow threads to change the value of *pivot_lambda* at the same time, otherwise it would be a thread-race. Critical regions usually cause overheads (threads need to synchronize and do stuff one at a time).

An even better approach wold be to combine steps 1 and 2 from the partial pivoting strategy, immediately. If $|a_{11}^{(k)}| > \alpha_0 |a_{j1}^{(k)}|$ for some index j, we can stop and proceed to pivot 1, since λ is not needed afterwards. This way, not every $a_{j1}^{(k)}$ is computed. This can be implemented without critical regions with the user defined reduction, but only if we compute all the $a_{j1}^{(k)}$ in advance with *zdot*. It can also be implemented with critical regions, as it is now, just with a small modification. We should then have one shared variable so that the threads can atomically check if the search is finished earlier. The same goes for combining steps 3 and 4 in the partial pivoting strategy.

The performance of the application should then be checked with any tool that allows an insight in memory leaks, cache misses, instructions per second, etc. This analysis can help determine which part of the application spends too much time and why.

Bibliography

[1] *A list of OpenMP environment variables*, https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.cbcux01/ruomprun.htm, Accessed: 2018-07-01.

[2] *Code on GitHub*, https://github.com/caklovicka/diplomski, Accessed: 2018-07-01.

[3] *BLAS (Basic Linear Algebra Subprograms)*, http://www.netlib.org/blas/, Accessed: 2018-07-01.

[4] James R. Bunch, *Analysis of the diagonal pivoting method*, SIAM J. Numer. Anal. **8** (1971), no. 4, 656–680.

[5] ———, *Partial pivoting strategies for symmetric matrices*, SIAM J. Numer. Anal. **11** (1974), no. 3, 521–528.

[6] James R. Bunch and Linda C. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comp. **31** (1977), no. 137, 163–179.

[7] James R. Bunch, Linda C. Kaufman, and Beresford N. Parlett, *Decomposition of a symmetric matrix*, Numer. Math. **27** (1976), no. 1, 95–109.

[8] James R. Bunch and Beresford N. Parlett, *Direct methods for solving symmetric indefinite systems of linear equations*, SIAM J. Numer. Anal. **8** (1971), no. 4, 639–655.

[9] *False sharing*, https://docs.oracle.com/cd/E19205-01/819-5270/aewcx/index.html, Accessed: 2018-07-01.

[10] *Intel Xeon Phi* 7210, https://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors/7210.html, Accessed: 2018-07-01.

[11] *LAPACK–Linear Algebra PACKage*, http://www.netlib.org/lapack/, Accessed: 2018-07-01.

[12] *Intel Math Kernel Library*, https://software.intel.com/en-us/mkl, Accessed: 2018-07-01.

[13] *Process and thread affinity for Intel Xeon Phi processors*, https://software.intel.com/en-us/articles/process-and-thread-affinity-for-intel-xeon-phi-processors-x200, Accessed: 2018-07-01.

[14] *OpenMP*, https://www.openmp.org/, Accessed: 2018-07-01.

[15] Sanja Singer, *Indefinite QR factorization*, BIT **46** (2006), no. 1, 141–161.

[16] Sanja Singer and Saša Singer, *Orthosymmetric block reflectors*, Linear Algebra Appl. **429** (2008), no. 5–6, 1354–1385.

[17] ———, *Orthosymmetric Scalar Products — Theory and Algorithms (in Croatian)*, 2017, https://www.fsb.unizg.hr/ssinger/Ortosimetricni_skalarni_produkti/pred.pdf, University of Zagreb, Faculty of Science, Department of Mathematics, online course material, accessed: 2018-07-01.

[18] Krešimir Veselić, *On a new class of elementary matrices*, Numer. Math. **33** (1979), no. 2, 173–180.

Sažetak

U ovom radu prezentirali smo kako računati hiperboličku QR J-faktorizaciju. Prvo je postavljena teorija koja nam daje dva načine redukcije matrice $G \in \mathbb{C}^{m\times n}$, $m \geq n$, na blok gornjetrokutastu formu. Jedan način je redukcija jednog stupca pomoću J-Householderovog reflektora. Razjašnjeni su nužni i dovoljni uvjeti postojanja takvih operatora. Drugi način je redukcija dva stupca koristeći Givensove rotacije. U tom poglavlju je obrađeno što sve zovemo pravilnom ('proper') formom, kako svesti matrice na pravilnu formu, te kako tu pravilnu formu do kraja reducirati J-unitarnim matricama manjih dimenzija.

Nadalje, indefinitni QR povezali smo sa još jednom faktorizacijom, hermitskom indefinitnom faktorizacijom. Pokazali smo kako su te dvije faktorizacije povezane i koja je optimalna strategija pivotiranja u hermitskoj indefinitnoj faktorizaciji (ona koja ima najmanji pivotni rast u svakom slučaju izbora strategije, nebitno biraju li se dva ili jedan stupac za redukciju). Ista pivotna strategija primijenjena je i na QR faktorizaciju.

Naposljetku, prezentiran je sekvencijalni algoritam redukcije matrice G na gorenje blok trokutastu formu, kao i njegovi dijelovi koji su paralelizirani. Pri optimizaciji koda, u obzir je uzeta i arhitektura memorije računala, te način funkcioniranja biblioteka OpenMP i MKL koje smo koristili za paralelizaciju. Testiranja na umjetno generiranim matricama izvedena su na Intelovom Xeon Phi 7210 računalu, gdje je također u obzir uzeta posebna memorijska arhitektura računala.

Summary

In this thesis, a way of computing a J-unitary QR factorization was presented. The theoretical part was set first, in order to explain two possible ways of transforming a matrix $G \in \mathbb{C}^{m\times n}$, $m \geq n$, into a block upper triangular matrix. One way to do this is with J-unitary Householder like reflectors. The necessary and sufficient conditions for their existence were formed. The other way to do this is using Givens rotations. In that chapter, a term proper form was defined, how to transform matrices into proper forms and, in the end, how are those proper forms fully reduced with J-unitary matrices of smaller dimensions.

Furthermore, we showed how indefinite QR is connected to the Hermitian indefinite factorization. An optimal pivoting strategy for the Hermitian indefinite factorization was presented, based on minimizing the pivot growth (regardless of the fact that one or two columns were chosen as pivotal). The same strategy was then used in the QR factorization.

At last, a sequential version of the algorithm for reducing the matrix G to a block upper triangular form was presented, as well as with the parallelised segments of it. The memory architecture was took into account while optimizing the code as well as the optimal usage of OpenMP and MKL libraries. The tests on randomly generated matrices were performed on Intel's Xeon Phi 7210. The special architecture of Xeon Phi was also taken into account.

Curriculum Vitae

Gayatri Čaklović rođena je 31. svibnja 1994. u Zagrebu. Pohađala je osnovnu školu Antuna Gustava Matoša u Zagrebu, a nakon toga upisuje XV. gimnaziju, te maturira 2013. godine sa odličnim uspjehom. Iste godine upisuje preddiplomski studij matematike na Prirodoslovno-matematičkom fakultetu Sveučilišta u Zagrebu. Na trećoj godini preddiplomskog studija ostvaruje pravo na stipendiju Grada Zagreba. Istovremeno dobiva i priznanje od matematičkog odsjeka na PMF-u za izniman uspjeh na studiju u akademskoj godini 2015./2016. Dana 13. rujna 2016. završava preddiplomski studij matematike i postaje sveučilišna prvostupnica (baccalaurea) matematike. Iste godine u jesen, upisuje diplomski studij Primijenjene matematike na PMF-u u Zagrebu, a na ljeto 2018. pohađa tromjesečnu ljetnu školu u "Institute for Advanced Simulation, Jülich Supercomputing Centre" u Njemačkoj i istražuje konvergenciju GMRES metode. Na drugoj godini diplomskog studija ponovno dobiva priznanje matematičkog odsjeka, ali i Pohvalnicu od Prirodoslovno–matematičkog fakulteta za izuzetan uspjeh na studiju.

Gayatri Čaklović was born on 31. May 1994 in Zagreb. She attended elementary school of Antun Gustava Matoša in Zagreb, then enrolled in XV. high-school and graduated in 2013 with great success. That same year she enrolled in the undergraduate study in mathematics at the Faculty of Science, University of Zagreb. On the third year of undergraduate studies, she is entitled to a scholarship of the City of Zagreb. At the same time, she is also awarded from the mathematics department at PMF for her outstanding academic success in the academic year 2015/2013. On September 13, 2016, she finished her undergraduate studies in mathematics and became a university baccalaurea mathematician. In the fall of the same year, she enrolled in a graduate study of Applied Mathematics at the Faculty of Science in Zagreb, and in the summer 2018, she attended a three-month summer school at the "Advanced Computing Simulation Institute, Jülich Supercomputing Centre" in Germany and investigated the convergence of the GMRES method. On her second year of graduate studies, she receives recognition from the Mathematics Department, as well as the Praise from the Faculty of Science and Mathematics for outstanding academic success.