Ein orthogonal symplektischer Multishift Algorithmus zur Lösung der algebraischen Riccatigleichung

Peter Benner
Rehmplatz 9
5100 Aachen

27. März 1993

Zusammenfassung

Die Lösung der algebraischen Riccatigleichung kann mit Hilfe eines bestimmten invarianten Teilraums der zugehörigen Hamiltonischen Matrix gefunden werden. Für die in vielen Anwendungen benötigte stabilisierende Lösung ist dies der Lagrange–Teilraum zu den Eigenwerten mit negativem Realteil dieser Matrix. G. Ammar und V. Mehrmann konnten einen Algorithmus zur Berechnung von Lagrange–Teilräumen angeben, der die Hamiltonische Struktur durch ausschließliche Verwendung orthogonal symplektischer Ähnlichkeitstransformationen ausnutzt. Auf der Basis dieser Methode wird nach Einführung der zur Lösung der algebraischen Riccatigleichung nötigen theoretischen Ergebnisse und einiger orthogonal symplektischer Basisalgorithmen ein Algorithmus dargestellt, der die stabilisierende Lösung der algebraischen Riccatigleichung bestimmt. Dieses Verfahren kann noch mit einer Defektkorrekturmethode gekoppelt werden, mit der die erzielten Ergebnisse auf einfache Weise verbessert werden können. Es werden die Implementierung des Algorithmus beschrieben, die numerischen Eigenschaften aufgezeigt und anhand einiger Beispiele mit denen anderer Methoden verglichen.

Inhaltsverzeichnis

1 Einleitung **3**

2 Die Lösung der algebraischen Riccatigleichung **9**

2.1 Notation und Definitionen 9

2.2 Hamiltonische Matrizen und die Lösung der algebraischen Riccatigleichung 14

2.3 Bestimmung invarianter Teilräume Hamiltonischer Matrizen mit symplektischen Transformationen 21

2.4 Numerische Lösungsverfahren 27

2.5 Defektkorrektur 35

3 Orthogonal symplektische Algorithmen für Hamiltonische Matrizen **40**

3.1 Orthogonal symplektische Transformationsmatrizen 40

3.1.1 Householder–Matrizen 41

3.1.2 Givens–Matrizen 43

3.1.3 Eigenschaften der Gruppe $\mathcal{US}_{2n}$ 46

3.2 Orthogonal symplektische Transformation auf Hessenberg–ähnliche Gestalt 48

3.3 Bestimmung der Eigenwerte einer Hamiltonischen Matrix . . . 52

3.4 Symmetrisches Aufdatieren der Lösung der algebraischen Riccatigleichung 56

4 Ein orthogonal symplektischer Multishift–Algorithmus für die algebraische Riccatigleichung **62**

4.1 Orthogonal symplektische Berechnung invarianter Teilräume Hamiltonischer Matrizen 62

4.2 Anwendung auf die algebraische Riccatigleichung 73
4.3 Defektkorrektur mit dem Multishift–Algorithmus 79

5 Implementierung und numerische Ergebnisse 80
5.1 Fortran–Subroutinen 80
5.1.1 HAMSQR.F 84
5.1.2 SQREDE.F, SKWREF.F 86
5.1.3 MSHIFT.F 87
5.1.4 COLRED.F, COLRD2.F 89
5.1.5 PVLRED.F, PVLRD2.F, PVLUPD.F 90
5.1.6 OSMSTP.F, OSMST2.F, OSMSTU.F 91
5.1.7 XSOLVE.F 93
5.1.8 OSMARE.F, OSMAR2.F, OSMARU.F 95
5.1.9 DCORC.F 98
5.2 Matlab–Routinen 99
5.3 Beispiele 100

6 Schlußfolgerungen und Ausblick 110

Literaturverzeichnis 113

Kapitel 1

Einleitung

Linear quadratische Steuerungsprobleme, Differentialspiele und die Berechnung von Kalman–Filtern benötigen die Lösung von *algebraischen Riccatigleichungen*. Dabei handelt es sich um symmetrische quadratische Matrixgleichungen der Form

$$F + A^T X + XA - XGX = 0, \tag{1.1}$$

mit $A, F, G \in \mathbf{R}^{n,n}$, wobei F und G symmetrisch und positiv semidefinit sind.

Man betrachte z.B. das linear quadratische Steuerungsproblem

$$\min \mathcal{S}(y,u) = \frac{1}{2}\int_0^\infty \Big(y(t)^T Q y(t) + u(t)^T R u(t)\Big)\, dt \tag{1.2}$$

mit positiv definiten $Q \in \mathbf{R}^{p,p}$, $R \in \mathbf{R}^{m,m}$, wobei folgendes dynamisches System erfüllt sei:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \quad x(0) = x_0, \\ y(t) &= Cx(t) \end{aligned} \tag{1.3}$$

mit $A \in \mathbf{R}^{n,n},\ B \in \mathbf{R}^{n,m},\ C \in \mathbf{R}^{p,n}$.
Unter den Bedingungen, daß (A, B) stabilisierbar und (A, C^TQC) entdeckbar (vgl. Definition 2.8) sind, kann man eine eindeutige optimale Lösung von (1.2), (1.3) angeben, siehe z.B. Mehrmann [M 1], Casti [C 1], Knobloch/Kwakernaak [K 2]. Diese ist durch die Rückkopplung

$$u(t) = -R^{-1}B^T X x(t), \tag{1.4}$$

gegeben. Dabei ist X die eindeutige positiv semidefinite Lösung der algebraischen Riccatigleichung

$$0 = C^T QC + A^T X + XA - XBR^{-1}B^T X. \tag{1.5}$$

X ist *stabilisierend* in dem Sinne, daß alle Eigenwerte von $A - BR^{-1}B^T X$ negativen Realteil besitzen. Damit ist die zur optimalen Steuerung u gehörende Trajektorie x, also die Lösung des Anfangswertproblems

$$\dot{x}(t) = (A - BR^{-1}B^T)x(t), \quad x(0) = x_0, \tag{1.6}$$

asymptotisch stabil, d.h. $\lim_{t\to\infty} x(t) = 0$.
Die Gleichung (1.5) wird in der Steuerungstheorie als „stetige" symmetrische algebraische Riccatigleichung bezeichnet, zur Unterscheidung von der „diskreten" algebraischen Riccatigleichung, siehe Mehrmann [M 1].
Ein einfaches Steuerungsproblem findet man z.B. bei Mehrmann [M 1] oder Knobloch/Kwakernaak [K 2].

Beispiel 1.1: Betrachte die Steuerung eines Gleichstrommotors

$$u \longrightarrow \boxed{\text{VERSTÄRKER}} \longrightarrow \boxed{\text{GLEICHSTROMMOTOR}} \longrightarrow \theta$$

Der Gleichstrommotor werde durch die Eingangsspannung u mittels eines Verstärkers geregelt. Den Ausgang des Systems bilde der Drehwinkel θ der Motorachse.
Sei nun ω die Winkelgeschwindigkeit der Motorachse, dann ist der Zustand des Systems durch $x = \begin{bmatrix} \theta \\ \omega \end{bmatrix}$ gegeben. Linearisiert man die zugehörigen Bewegungsgleichungen, so erhält man

$$\begin{aligned} \dot{\theta} &= \omega \\ j\dot{\omega} &= -b\omega + ku. \end{aligned}$$

Dabei sei

- j das Trägheitsmoment des Motors,
- b der Reibungskoeffitient und
- k der Verstärkungsfaktor.

Unter der physikalisch sicher vernünftigen Annahme, daß das Trägheitsmoment des Motors nicht verschwindet, erhält man daraus das Anfangswertproblem

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{b}{j} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{k}{j} \end{bmatrix} u(t), \quad x(0) = 0. \tag{1.7}$$

Den Ausgang des Systems bildet

$$y(t) = [\,1 \quad 0\,]\, x(t) = \theta. \tag{1.8}$$

Betrachtet man das Kostenfunktional

$$\mathcal{S}(y, u) = \frac{1}{2} \int_0^\infty \left(y(t)^T y(t) + \rho u(t)^2 \right) \, dt, \tag{1.9}$$

so erhält man folgende Systemmatrizen:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{b}{j} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{k}{j} \end{bmatrix}, \quad C = [\,1 \quad 0\,], \quad Q = [\,1\,], \quad R = [\,\rho\,].$$

Um die optimale Steuerung u nach (1.4) angeben zu können, benötigt man also die stabilisierende Lösung der algebraischen Riccatigleichung, vgl. (1.5),

(1.10)

$$\begin{aligned} 0 \;=\; & \begin{bmatrix} 1 \\ 0 \end{bmatrix} [1 \quad 0] + \begin{bmatrix} 0 & 0 \\ 1 & -\frac{b}{j} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} + \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & -\frac{b}{j} \end{bmatrix} \\ & - \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \begin{bmatrix} 0 \\ \frac{k}{j} \end{bmatrix} \frac{1}{\rho} \begin{bmatrix} 0 & \frac{k}{j} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}. \end{aligned}$$

Damit ergibt sich die optimale Steuerung des Systems gemäß

$$u(t) = -\frac{1}{\rho} \begin{bmatrix} 0 & \frac{k}{j} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \begin{bmatrix} \theta(t) \\ \omega(t) \end{bmatrix}. \;\Box \tag{1.11}$$

Setzt man $G := BR^{-1}B^T$ und $F := C^TQC$, so erhält man aus (1.5) wieder die Darstellung (1.1), da aus der Symmetrie und positiven Definitheit von Q und R sofort folgt, daß G und F symmetrisch und positiv semidefinit sind. Daher wird im Folgenden für die algebraische Riccatigleichung (kurz: ARG) immer die Form (1.1) benutzt.

Gesucht ist also die symmetrische positiv semidefinite Lösung $X \in \mathbf{R}^{n,n}$ von (1.1). Da die ARG i.a. viele unterschiedliche Lösungen besitzt, muß man somit sicherstellen, daß man sich für die „Richtige“ entscheidet.
Als wesentlich bei der Berechnung der Lösung der ARG erweist sich der Zusammenhang mit den invarianten Teilräumen der zur ARG (1.1) gehörenden *Hamiltonischen* Matrix

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \in \mathbf{R}^{2n,2n} \tag{1.12}$$

Dieser Zusammenhang wird in Kapitel 2.2 dargestellt, insbesondere ergibt sich daraus auch die Existenz und Eindeutigkeit der gesuchten stabilisierenden Lösung unter den oben an das Steuerungsproblem gestellten Bedingungen.

Die numerischen Verfahren zur Berechnung dieser Lösung lassen sich im Wesentlichen in zwei Klassen unterteilen. Dies sind einerseits die Methoden, die auf der Berechnung eines invarianten Teilraums der Hamiltonischen Matrix H aus (1.12) beruhen. Dabei wird mittels Ähnlichkeitstransformationen eine Schur–ähnliche Form von H berechnet. Vertreter dieser Klasse sind u.a. die auf dem QR-Algorithmus für unsymmetrische Matrizen (siehe z.B. [G 1]) basierende Schurvektor–Methode nach A.J. Laub [L 1], der Hamiltonische SR–Algorithmus [M 1] und der Hamiltonische QR–Algorithmus von R. Byers [B 4]. Letzterer ist allerdings auf den Fall $\mathrm{rg}(G) = 1$ oder $\mathrm{rg}(F) = 1$ beschränkt, d.h. in der Sprache der Steuerungstheorie auf Systeme mit nur einem Eingang oder einem Ausgang.

Die andere Möglichkeit besteht darin, die ARG direkt zu lösen. Wichtigster Vertreter dieser Kategorie ist das Newton–Verfahren [K 1]. Ein weiteres Verfahren, die Matrix–Signum–Funktion Methode [B 5], läßt sich nicht direkt in diese Klassifizierung einordnen. Die Grundlage besteht zwar in der Kenntnis eines invarianten Teilraums von (1.12), es wird jedoch keine Schur–ähnliche Form von H berechnet.

Von einem numerischen Verfahren verlangt man i.a., daß es

- die Struktur des Problems ausnutzt,
- numerisch stabil und
- effizient

ist. Das bedeutet für Algorithmen, die auf eine Schur–ähnliche Form von (1.12) abzielen, daß die benutzten Ähnlichkeitstransformationen *symplektisch* (siehe Definition 2.4) sein sollten, da diese die Hamiltonische Struktur erhalten.
Andererseits erfordert die numerische Stabilität die Orthogonalität der Ähnlichkeitstransformationen, da diese optimal in Hinsicht auf Rundefehler sind, siehe dazu z.B. Golub/Van Loan [G 1]. Paige/Van Loan [P 1] konnten zeigen, daß unter gewissen Voraussetzungen eine orthogonal symplektische Matrix existiert, die H aus (1.12) auf eine *Hamiltonische Schur–Form* transformiert. Den Zusammenhang zwischen Hamiltonischen und symplektischen Matrizen sowie den Rahmen für einen Algorithmus, der obige Forderungen erfüllt, behandelt Kapitel 2.3.
Der Hamiltonische QR–Algorithmus besitzt zwar alle obigen Eigenschaften, unterliegt jedoch der oben erwähnten Einschränkung, daß er nur für bestimmte Spezialfälle einsetzbar ist. Die anderen erwähnten Verfahren erfüllen nicht alle oben aufgeführten Forderungen. Die genannten Verfahren und ihre Eigenschaften werden in Kapitel 2.4 näher erläutert.

Benutzt man ein numerisch nicht stabiles Verfahren zur Lösung der ARG, so benötigt man eine Methode, die so gewonnene Näherungslösung zu verbessern. Das in Kapitel 2.5 vorgestellte Defektkorrekturverfahren bietet eine solche Möglichkeit.

In Kapitel 3.1 werden die orthogonal symplektischen Matrizen genauer untersucht. Daraus läßt sich (Kapitel 3.2, vgl. [P 1]) ein Reduktionsalgorithmus für Hamiltonische Matrizen angeben, mit dessen Hilfe ein orthogonal symplektischer Algorithmus zur Berechnung der Eigenwerte — nicht aber der Eigenvektoren — Hamiltonischer Matrizen hergeleitet werden kann, siehe Kapitel 3.3 bzw. [V 1].
Benutzt man orthogonal symplektische Transformationen, um einen invarianten Teilraum der zur ARG gehörenden Hamiltonischen Matrix aus (1.12) zu berechnen, so kann man nach Byers/Mehrmann [B 6] die Lösung X der ARG direkt aufdatieren, ohne die Transformationen zu akkumulieren. Dieses Vorgehen wird in Kapitel 3.4 erläutert.

Keines der bisher erwähnten Verfahren zur Lösung von (1.1) ist vollkommen befriedigend im Sinne der oben aufgestellten Forderungen. Ammar/Mehrmann konnten in [A 1] aufzeigen, wo das Problem, einen allgemeinen QR–ähnlichen Algorithmus für Hamiltonische Matrizen anzugeben, zu suchen ist.

Wie in Kapitel 4.1 dargelegt, gelang es ihnen jedoch basierend auf diesem Ergebnis und unter Kenntnis der Eigenwerte einer Hamiltonischen Matrix $H \in \mathbf{R}^{2n,2n}$ einen *Multishift*–Algorithmus zur Berechnung bestimmter n–dimensionaler H–invarianter Teilräume anzugeben. Mit Hilfe dieses Algorithmus kann man nun einen orthogonal symplektischen Algorithmus zur Lösung der ARG (1.1) angeben, siehe dazu Kapitel 4.2. Auf der Basis von Kapitel 2.5 wird in Kapitel 4.3 ein Defektkorrekturverfahren unter Ausnutzung der Eigenschaften des Multishift–Algorithmus vorgestellt.

In Kapitel 5 wird dann näher auf die Implementierung in FORTRAN 77 und MATLAB des im 4. Kapitel vorgestellten Multishift–Algorithmus sowie der dazu erforderlichen Routinen eingegangen. Außerdem werden die numerischen Eigenschaften anhand einiger Beispiele erläutert.

Kapitel 2

Die Lösung der algebraischen Riccatigleichung

In diesem Kapitel werden die für das Verständnis dieser Arbeit notwendigen theoretischen Ergebnisse über die algebraische Riccatigleichung (1.1) vorgestellt. Außerdem wird ein Überblick über die wichtigsten bekannten numerischen Verfahren zur Berechnung einer Lösung der ARG gegeben.

2.1 Notation und Definitionen

In diesem Abschnitt werden einige wesentliche Schreibweisen und grundlegende Definitionen eingeführt.

Zunächst einige Bezeichnungen.

Übersicht 2.1 Im Folgenden bezeichne mit

$\mathbf{R}$ ($\mathbf{C}$)	— die reellen (komplexen) Zahlen,
$\mathbf{R}^{n,m}$ ($\mathbf{C}^{n,m}$)	— die reellen (komplexen) $n \times m$– Matrizen, $\mathbf{R}^n := \mathbf{R}^{n,1}$, ($\mathbf{C}^n := \mathbf{C}^{n,1}$)
$Re(z), Im(z)$	— den Real- bzw. Imaginärteil einer komplexen Zahl $z \in \mathbf{C}$,
$\bar{z}$	— die konjugiert Komplexe von $z = x + \imath y \in \mathbf{C}$, also $\bar{z} = x - \imath y$,
e_j	— den j–ten Einheitsvektor,
I_n/I	— die Einheitsmatrix in $\mathbf{R}^{n,n}$, falls die Dimension klar ist, wird der Index weggelassen.

Für Operationen mit Vektoren und Matrizen bzw. deren Eigenschaften werden die folgenden Schreibweisen benutzt.

Übersicht 2.2 Es sei $A = [a_{ij}] \in \{\mathbf{R}^{n,n}, \mathbf{C}^{n,n}\}$ und $a \in \{\mathbf{R}^n, \mathbf{C}^n\}$, dann bezeichne mit

A^T,a^T	— die Transponierte von A bzw. a,
A^*,a^*	— die kongugiert komplex Transponierte von A bzw. a, d.h. $A^* = \bar{A}^T$, $a^* = \bar{a}^T$,
A^{-1}	— die Inverse von A, falls A invertierbar ist,
a_{j*}, a_{*j}	— die j–te Zeile bzw. Spalte von A,
$\mathrm{rg}(A)$	— den Rang von A,
$\det(A)$	— die Determinante von A,
$\mathrm{diag}\,(A_1, \ldots, A_k)$	— die Blockdiagonalmatrix mit Blöcken $A_1, \ldots, A_k$ auf der Diagonalen,
$\sigma(A)$	— das Spektrum von A, d.h. die Menge der Eigenwerte von A,
$\mathrm{span}\,\{a_1, \ldots, a_k\}$	— den k–dimensionalen Teilraum des $\mathbf{R}^n$, der durch die k linear unabhängigen Vektoren $a_1, \ldots, a_k \in \mathbf{R}^n$ aufgespannt wird,
$\|A\|, \|a\|$	— eine beliebige Matrix-/Vektornorm,
$\|a\|_2$	— die Euklidische Norm des Vektors a, d.h. $\|a\|_2 = \sqrt{a_1^2 + \cdots + a_n^2}$,
$\|A\|_2$	— die 2–Norm für Matrizen, d.h. $\|A\|_2 = \sqrt{\max\{\lvert\lambda\rvert \mid \lambda \in \sigma(A^TA)\}}$,
$\|A\|_1$	— die Spaltensummen–Maximumnorm für Matrizen, d.h. $\|A\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^n \lvert a_{ij}\rvert$,
$\|A\|_F$	— die Frobeniusnorm für Matrizen, d.h. $\|A\|_F = \sqrt{\sum_{i,j=1}^n \lvert a_{ij}\rvert^2}$.

Es werden folgende Matrixeigenschaften benutzt:

Definition 2.3 $A = [a_{ij}] \in \{\mathbf{R}^{n,n}, \mathbf{C}^{n,n}\}$ *heißt*

i) **symmetrisch**, *falls $A = A^T$ bzw. $a_{ij} = a_{ji}$ für $i, j = 1, \ldots, n$,*

ii) **Hermitesch**, *falls $A = A^*$ bzw. $a_{ij} = \bar{a}_{ji}$ für $i, j = 1, \ldots, n$,*

iii) **schiefsymmetrisch**, *falls $A = -A^T$ bzw. $a_{ij} = -a_{ji}$ für $i, j = 1, \ldots, n$,* *iv)* **positiv semidefinit**, *falls* $x^T Ax \geq 0$ *für alle* $x \in \mathbf{R}^n$ *(Schreibweise:* $A \geq 0$*),*

v) **positiv definit**, *falls* $x^T Ax > 0$ *für alle* $x \in \mathbf{R}^n \backslash \{0\}$ *(Schreibweise:* $A > 0$*),*

vi) **orthogonal**, *falls* $A^T A = I$*, d.h.* $A^{-1} = A^T$*,*

vii) **c–stabil**, *falls* $Re(\lambda) < 0$ *für alle* $\lambda \in \sigma(A)$*,*

viii) **diagonal**, *falls* $a_{ij} = 0$ *für* $i \neq j$*, d.h.* $A \triangleq \left[\ \diagdown\ \right]$*,*

ix) **obere Dreiecksmatrix**, *falls* $a_{ij} = 0$ *für* $i > j$*, d.h.* $A \triangleq \left[\ \triangledown\ \right]$*,*

x) **obere Hessenbergmatrix**, *falls* $a_{ij} = 0$ *für* $i > j+1$*, d.h.* $A \triangleq \left[\ \triangledown\!\!\diagdown\ \right]$*,*

xi) in **reeller Schurform**, *falls*

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1k} \\ & A_{22} & \dots & A_{2k} \\ \vdots & & \ddots & \vdots \\ 0 & & \dots & A_{kk} \end{bmatrix},$$

wobei $A_{ii} \in \left\{\mathbf{R}^{1,1}, \mathbf{R}^{2,2}\right\}$ *für* $i = 1, \dots, k$ *mit* $\lfloor \frac{n}{2} \rfloor \leq k \leq n$*. Dabei entsprechen die reellen Eigenwerte von* A *den* 1×1*–Blöcken und die konjugiert komplexen Eigenwerte von* A *den Eigenwerten der* 2×2*–Blöcke.*

Eine zentrale Rolle in der Behandlung von algebraischen Riccatigleichungen spielen Hamiltonische und symplektische Matrizen. Um diese zu definieren, führt man zunächst folgende schiefsymmetrische Matrix ein:

$$J = \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix} \in \mathbf{R}^{2n,2n}$$

Definition 2.4 *Sei* $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \in \mathbf{R}^{2n,2n}$*,* $A_{11}, A_{12}, A_{21}, A_{22} \in \mathbf{R}^{n,n}$*, dann heißt* A

i) **Hamiltonisch**, *falls* $AJ = (AJ)^T$*,*

ii) **symplektisch**, *falls* $AJA^T = J$*.*

Bemerkung: Wie man leicht nachrechnet, muß für Hamiltonische Matrizen gelten

$$A_{22} = -A_{11}^T,\ A_{12} = A_{12}^T,\ A_{21} = A_{21}^T.$$

Mit obigen Bezeichnungen kann man folgende Mengen von Matrizen definieren.

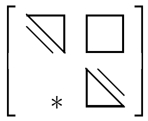
Definition 2.5 *Es bezeichnet*

$\mathcal{H}_{2n}$ — *die Menge der Hamiltonischen Matrizen in* $\mathbf{R}^{2n,2n}$,
$\mathcal{S}_{2n}$ — *die Gruppe der symplektischen Matrizen in* $\mathbf{R}^{2n,2n}$,
$\mathcal{U}_n$ — *die Gruppe der orthogonalen Matrizen in* $\mathbf{R}^{n,n}$,
$\mathcal{US}_{2n}$ — *die Gruppe der orthogonalen symplektischen Matrizen in* $\mathbf{R}^{2n,2n}$, *d.h.* $\mathcal{US}_{2n} = \mathcal{U}_{2n} \cap \mathcal{S}_{2n}$.

Analog zu den Matrixtypen aus Definition 2.3 gibt es Hamiltonische Matrizen mit spezieller Gestalt.

Definition 2.6 *Sei* $H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \in \mathbf{R}^{2n,2n}$ *Hamiltonisch.*

i) H ist in **Hamiltonischer Hessenbergform**, *falls A obere Hessenbergmatrix, G symmetrisch und* $F = f e_n e_n^T$, $f \in \mathbf{R}$, *ist, d.h.*

$A \triangleq$  .

H heißt in diesem Falle **unreduziert**, *falls* $a_{i+1,i} \neq 0$ *für* $i = 1, \ldots, n-1$ *und* $f \neq 0$.

ii) H ist in **Hamiltonischer Schurform**, *falls A in reeller Schurform, G symmetrisch und* $F = 0$ *ist.*

Wesentlich bei der Behandlung der ARG sind invariante Teilräume Hamiltonischer Matrizen und einige spezielle Eigenschaften dieser Räume.

Definition 2.7

i) Ein k–dimensionaler Teilraum χ *des* $\mathbf{R}^n$ *heißt* **invarianter Teilraum** *für* $A \in \mathbf{R}^{n,n}$ *(auch A-invarianter Teilraum), falls Matrizen* $P \in \mathbf{R}^{n,k}$ *und* $Q \in \mathbf{R}^{k,k}$ *existieren mit*

$$AP = PQ$$

und die Spalten von P χ *aufspannen, d.h.* $\chi = span\{p_{*1}, \ldots, p_{*k}\}$.

ii) Ein Teilraum $\mathcal{Q} \subset \mathbf{R}^{2n}$ heißt **isotrop**, *falls $x^T J y = 0$ für alle $x, y \in \mathbf{R}^{2n}$.*

iii) Ein maximaler isotroper Teilraum $\mathcal{Q} \subset \mathbf{R}^{2n}$ (d.h. $\mathcal{Q}$ ist nicht in einem größeren isotropen Teilraum enthalten) heißt **Lagrange–Teilraum**.

Bemerkung: Wie man leicht nachrechnet, gilt für einen Lagrange–Teilraum $\mathcal{Q} \subset \mathbf{R}^{2n}$, daß $\dim \mathcal{Q} \leq n$.

Aus der Steuerungstheorie werden zwei Begriffe benötigt, die Anforderungen an Paare von Matrizen stellen. In den nächsten Abschnitten werden diese Bedingungen benötigt, um Aussagen über die Existenz und Eindeutigkeit der stabilisierenden Lösung der ARG (1.1) treffen zu können.

Definition 2.8 *Sei $A \in \mathbf{R}^{n,n},\ B \in \mathbf{R}^{n,m},\ C \in \mathbf{R}^{k,n}$.*

i) (A, B) heißt **stabilisierbar**, *falls*

$$\mathrm{rg}\,[\lambda I - A, B] = n$$

für alle $\lambda \in \mathbf{C}$ mit $Re(\lambda) \geq 0$.

ii) (A, C) heißt **entdeckbar**, *falls*

$$\mathrm{rg} \begin{bmatrix} \lambda I - A \\ C \end{bmatrix} = n$$

für alle $\lambda \in \mathbf{C}$ mit $Re(\lambda) \geq 0$.

Zum Abschluß dieses Kapitels werden noch einige Bezeichnungen eingeführt, die zur Aufwands- und Fehleranalyse der vorgestellten Algorithmen dienen. Das Ergebnis einer Rechenoperation mit Fließkommazahlen wird mit $fl(.)$ bezeichnet, wobei der Punkt für einen beliebigen Term steht. Sind z.B. $A, B \in \mathbf{R}^{n,n}$, so ist

$$E := fl(AB) - AB$$

die Differenzmatrix zwischen dem numerischen und dem exakten Ergebnis der Matrixmultiplikation von A mit B.

Definition 2.9

i) Die einer Rechenoperation mit Fließkommazahlen zugrundegelegte **Maschinengenauigkeit**, *d.h. die kleinste positive darstellbare Zahl x, für die $fl(1 + x) \neq 1$ gilt, wird im Folgenden mit $\mathbf{u}$ bezeichnet.*

ii) Als **Flop** *(floating point operation) bezeichne eine der elementaren Rechenoperationen Addition, Subtraktion, Multiplikation, Division und Bilden der Quadratwurzel, wobei die jeweiligen Operanden Fließkommazahlen sind.*

Man beachte, daß es sich hierbei um die neue Definition eines Flops handelt, siehe z.B. [G 1],[T 1]. Es wird also jede elementare Rechenoperation gezählt, im Unterschied zur alten Definition, bei der eine Anweisung der Form $Y(i,k) = Y(i,k) + sX(i,k)$ als ein Flop betrachtet wurde.
Bei der Behandlung numerischer Verfahren stellt sich die Frage nach der Rundefehleranfälligkeit des zugrundeliegenden Algorithmus. Ein wesentliche Forderung wird z.B. bei Mehrmann [M 1] angegeben.

Definition 2.10 *Ein Algorithmus wird als* **numerisch rückwärts stabil** *bezeichnet, falls die vom Algorithmus gelieferte numerische Lösung des Problems eine exakte Lösung für dasselbe Problem mit leicht gestörten Eingabedaten darstellt. Der Algorithmus ist* **numerisch rückwärts stark stabil***, falls die berechnete Lösung die exakte Lösung eines leicht gestörten Problems aus der selben Problemklasse ist.*

2.2 Hamiltonische Matrizen und die Lösung der algebraischen Riccatigleichung

In der Einleitung wurde bereits angedeutet, daß die Lösung der ARG

$$0 = F + A^T X + AX - XGX \tag{2.1}$$

mit der Bestimmung invarianter Teilräume der Hamiltonischen Matrix

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \in \mathbf{R}^{2n,2n} \tag{2.2}$$

zusammenhängt. Dazu benötigt man zunächst einige Eigenschaften Hamiltonischer Matrizen, die man z.B. bei Kućera [K 3] findet.

Lemma 2.11 *Sei $H \in \mathcal{H}_{2n}$, $\lambda \in \sigma(H)$ mit zugehörigem Eigenvektor x_0 und Hauptvektoren $x_1, \dots, x_k$, d.h.*

$$\begin{aligned} Hx_0 &= \lambda x_0, \\ (H - \lambda I)x_j &= x_{j-1}. \end{aligned}$$

Ist $x_j = \begin{bmatrix} y_j \\ z_j \end{bmatrix}$ mit $y_j, z_j \in \mathbf{C}^n$ für $j = 0, \dots, k$ und $w_j = \begin{bmatrix} z_j \\ -y_j \end{bmatrix}$, dann gilt

$$\begin{aligned} w_0^* H &= -\bar{\lambda} w_0^*, \\ w_j^*(H + \bar{\lambda} I) &= (-1)^j w_{j-1}^*, \quad j = 1, \dots, k. \end{aligned}$$

Beweis: Aus $Hx_0 = \lambda x_0$ folgt durch Linksmultiplikation mit J aus Kapitel 2.1 und Definition 2.4

$$\lambda J x_0 = JHx_0 = (JH)^T x_0.$$

Durch Transponieren und komplex Konjugieren und unter Beachtung, daß H und J reell sind, erhält man

$$x_0^* J^T H = -\bar{\lambda} x_0^* J^T.$$

Mit $w_0 = Jx_0 = \begin{bmatrix} z_0 \\ -y_0 \end{bmatrix}$ folgt die Behauptung für den Eigenvektor x_0.

Der Beweis für die Hauptvektoren erfolgt analog. □

Ist also λ Eigenwert von H mit dem Rechts–Eigenvektor $\begin{bmatrix} y_0 \\ z_0 \end{bmatrix}$ und den Rechts–Hauptvektoren $\begin{bmatrix} y_1 \\ z_1 \end{bmatrix}, \dots, \begin{bmatrix} y_k \\ z_k \end{bmatrix}$, so ist $-\bar{\lambda}$ ebenfalls Eigenwert von H und besitzt den Links–Eigenvektor $[\, z_0^* \;\; -y_0^* \,]$ und die Links–Hauptvektoren $-[\, z_1^* \;\; -y_1^* \,], \dots, (-1)^k [\, z_k^* \;\; -y_k^* \,]$.

Folgerung 2.12 *Sei $H \in \mathcal{H}_{2n}$. Ist $\lambda \in \sigma(H) \cap \mathbf{R}$, dann ist $-\lambda \in \sigma(H)$, ist $\lambda \in \sigma(H) \cap (\mathbf{C} \setminus \mathbf{R})$, dann sind auch $\bar{\lambda}, -\lambda, -\bar{\lambda} \in \sigma(H)$.*

Seien nun $G = BB^T$ und $F = C^T C$ *Voll–Rang–Faktorisierungen* von F, G, d.h. $\mathrm{rg}(G) = \mathrm{rg}(B)$, $\mathrm{rg}(F) = \mathrm{rg}(C)$, dann kann man eine weitere Aussage über das Spektrum Hamiltonischer Matrizen treffen (siehe Bemerkung nach Theorem 1 in [K 3]).

Lemma 2.13 *Sind (A,B) stabilisierbar und (A,C) entdeckbar, dann gilt $Re(\lambda) \neq 0$ für alle $\lambda \in \sigma(H)$.*

Beweis: Angenommen, es existiert ein $\lambda \in \sigma(H)$ mit $\lambda = \imath\mu$, $\mu \in \mathbf{R}$. Sei nun $x = \begin{bmatrix} y \\ z \end{bmatrix}$ Eigenvektor zu λ, dann kann man nach Ausmultiplizieren von

$$\begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \imath\mu \begin{bmatrix} y \\ z \end{bmatrix}$$

und einigen Umformungen zeigen (vgl. Kućera [K 3], Beweis zu Lemma 8, „$\Longrightarrow$“), daß

$$z^* B = 0, \tag{2.3}$$
$$Cy = 0 \tag{2.4}$$

gelten muß.
Da $x = \begin{bmatrix} y \\ z \end{bmatrix}$ Eigenvektor zu λ ist, können y, z nicht gleichzeitig verschwinden.
Ist nun $z \neq 0$, so ist wegen (2.3)

$$z^* \left[\, \imath\mu I - A \quad B \,\right] = \left[(-\bar{\lambda}) z^* - z^* A \quad 0 \right]$$

Wegen Lemma 2.11 gilt

$$z^* A + \bar{\lambda} z^* = y^* F = y^* C^T C$$

und wegen (2.4) $y^* C^T = 0$. Es folgt also

$$z^* \left[\, \imath\mu I - A \quad B \,\right] = 0,$$

und damit

$$\mathrm{rg} \left[\, \gamma I - A \quad B \,\right] < n \quad \text{für} \quad Re(\gamma) = Re(\imath\mu) = 0,$$

was der Stabilisierbarkeit von (A,B) widerspricht.
Analog folgt für $y \neq 0$ ein Widerspruch zur Entdeckbarkeit von (A,C). □

Folgerung 2.14 *Sind (A, B) stabilisierbar und (A, C) entdeckbar, dann existieren genau n stabile Eigenwerte $\lambda_1, \ldots, \lambda_n$ von H, d.h. $Re(\lambda_i) < 0$ für $i = 1, \ldots, n$.*

Sei nun ein invarianter Teilraum $\mathcal{Q} \subset \mathbf{R}^{2n}$ von H aus (2.2) gegeben. Dieser werde aufgespannt durch die Spalten von $\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \in \mathbf{R}^{2n,n}$, also

$$H \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} Z \tag{2.5}$$

mit $Z \in \mathbf{R}^{n,n}$ und $\sigma(Z) = \{\lambda_1, \ldots, \lambda_n\}$.
Da $\mathcal{Q}$ reell ist, muß die Menge $\{\lambda_1, \ldots, \lambda_n\}$ abgeschlossen bezüglich komplexer Konjugation sein. Sei nun $S \in \mathbf{C}^{n,n}$ invertierbar, so daß

$$\tilde{Z} = S^{-1} Z S$$

in Jordan–Normalform ist. Dann ist mit $V := Q_1 S$, $W := Q_2 S$ (2.5) äquivalent zu

$$H \begin{bmatrix} V \\ W \end{bmatrix} = \begin{bmatrix} V \\ W \end{bmatrix} \tilde{Z}, \tag{2.6}$$

wobei die Spalten von $\begin{bmatrix} V \\ W \end{bmatrix}$ aus Eigen– und Hauptvektoren von H zu den Eigenwerten $\lambda_1, \ldots, \lambda_n$ bestehen.
Damit können folgende Aussagen benutzt werden, die auf Potter [P 3] und Kućera [K 3] zurückgehen.

Lemma 2.15 *Jede Lösung der ARG (2.1) hat die Form*

$$X = -WV^{-1},$$

wobei mit $V = [\, v_1, \ldots, v_n \,]$, $W = [\, w_1, \ldots, w_n \,]$, $v_i, w_i \in \mathbf{R}^n$, die Vektoren $\begin{bmatrix} v_i \\ w_i \end{bmatrix}$, $i = 1, \ldots, n$, Eigen- und Hauptvektoren von H zu $\lambda_1, \ldots, \lambda_n$ darstellen und V so gewählt sei, daß V^{-1} existiert.
Dabei ist mit einem Hauptvektor auch jeder Hauptvektor niedrigerer Stufe in $\begin{bmatrix} V \\ W \end{bmatrix}$ enthalten. Sind andererseits $\begin{bmatrix} v_i \\ w_i \end{bmatrix}$, $i = 1, \ldots, n$, Eigen- und Hauptvektoren von H und ist V invertierbar, dann löst X die ARG (2.1).

Beweis: Siehe [P 3], Theorem 1. □

Wegen Lemma 2.15 ist auch durch jeden H–invarianten Teilraum gemäß (2.5) mit invertierbarem Q_1 eine Lösung der ARG gegeben, denn dann gilt

$$-WV^{-1} = -Q_2SS^{-1}Q_1^{-1} = -Q_2Q_1^{-1}.$$

Um die Invertierbarkeit von V bzw. Q_1 zu gewährleisten, benötigt man einige weitere Aussagen über V und W.

Lemma 2.16 *Seien V, W w.o. Gilt zusätzlich $\lambda_i + \bar{\lambda}_j \neq 0$ für $1 \leq i, j \leq n$, dann ist V^*W Hermitesch.*

Beweis: Siehe [P 3], Beweis zu Theorem 2. □

Die Behauptung aus Lemma 2.16 gilt insbesondere, falls $\lambda_1, \ldots, \lambda_n$ negative Realteile besitzen.

Lemma 2.17 *Seien V, W w.o. und $Re(\lambda_i) < 0$ für $i = 1, \ldots, n$. Dann ist V invertierbar.*

Beweis: Siehe [K 3], Beweis zu Theorem 1, „$\Longrightarrow$“, 2). □

Lemma 2.18 *Unter den Voraussetzungen von Lemma 2.17 ist $-WV^{-1}$ positiv semidefinit.*

Beweis: Wegen Lemma 2.17 ist V nichtsingulär. Damit ist die Aussage von Lemma 2.18 äquivalent zu Theorem 3(b) aus [P 3]. Wegen

$$WV^{-1} = V^{-*}(V^*W)V^{-1},$$

reicht es zu zeigen, daß $V^*W \leq 0$ gilt. Dies ist aber äquivalent zum Beweis in [P 3]. □

Diese Ergebnisse können zu folgendem Satz zusammengefaßt werden.

Satz 2.19 *Es sei* $H \in \mathcal{H}_{2n}$ *mit Spektrum* $\sigma(H) = \{\lambda_1, -\lambda_1, \ldots, \lambda_n, -\lambda_n\}$ *und es sei* $Re(\lambda_i) < 0$ *für* $i = 1, \ldots, n$. *Durch die Spalten von* $\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \in \mathbf{R}^{2n,n}$ *werde ein* H*–invarianter Teilraum zu* $\lambda_1, \ldots, \lambda_n$ *aufgespannt. Dann ist* Q_1 *invertierbar, mit* $X := -Q_2 Q_1^{-1}$ *gilt*

$$X = X^T \geq 0$$

und X *löst die algebraische Riccatigleichung*

$$0 = F + A^T X + XA - XGX. \tag{2.7}$$

Beweis: Nach Voraussetzung gilt

$$H \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} Z \tag{2.8}$$

mit $\sigma(Z) = \{\lambda_1, \ldots, \lambda_n\}$. Wähle $S \in \mathbf{C}^{n,n}$ nichtsingulär, so daß $\tilde{Z} = S^{-1} Z S$ in Jordan–Normalform ist und setze $V := Q_1 S,\ W := Q_2 S$, dann ist (2.8) äquivalent zu

$$H \begin{bmatrix} V \\ W \end{bmatrix} = \begin{bmatrix} V \\ W \end{bmatrix} \tilde{Z} \tag{2.9}$$

mit $\sigma(\tilde{Z}) = \sigma(Z)$. Auf V, W können nun die Lemmata 2.16 – 2.18 angewendet werden.
Nach Lemma 2.17 ist V invertierbar und damit auch

$$Q_1 = V S^{-1}.$$

Da $Re(\lambda_i) < 0$ für $i = 1, \ldots, n$ vorausgesetzt wurde, ist Lemma 2.16 anwendbar und es folgt, daß $V^* W$ und daher auch

$$W V^{-1} = V^{-*} V^* W V^{-1}$$

Hermitesch sind. Da Q_1, Q_2 reell sind, erhält man aus

$$X = -Q_2 Q_1^{-1} = -W S^{-1} S V^{-1} = -W V^{-1}, \tag{2.10}$$

daß X symmetrisch ist.
Mit Lemma 2.18 und (2.10) ergibt sich sofort $X \geq 0$.
Setzt man $\hat{Z} = Q_1 Z Q_1^{-1}$, so ist (2.8) äquivalent zu

$$\begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \begin{bmatrix} I \\ -X \end{bmatrix} = \begin{bmatrix} I \\ -X \end{bmatrix} \hat{Z}. \tag{2.11}$$

Multipliziert man (2.11) aus, so erhält man

$$A - GX = \hat{Z} \tag{2.12}$$

$$F + A^T X = -X\hat{Z} = -X(A - GX). \tag{2.13}$$

Die letzte Gleichung ist äquivalent zu (2.7). □

Folgerung 2.20 *Mit den Voraussetzungen und Bezeichnungen aus Satz 2.19 folgt, daß $A - GX$ c–stabil, X also stabilisierend im eingangs geforderten Sinne ist.*

Beweis: Klar mit (2.12) und $\sigma(\hat{Z}) = \sigma(Z) = \{\lambda_1, \ldots, \lambda_n\}$. □

Unter den Voraussetzungen von Lemma 2.13 bzw. Folgerung 2.14 besitzt H genau n stabile Eigenwerte. Damit ist die gesuchte Lösung der ARG gefunden, sobald man einen H–invarianten Teilraum zu diesen Eigenwerten gefunden hat. Die Existenz eines solchen Teilraums folgt z.B. aus dem Satz über die reelle *Schurform* einer Matrix (Definition 2.3, siehe z.B. [M 1], [S 1]) und aus der Tatsache, daß $Re(\lambda) < 0$ auch $Re(\bar{\lambda}) < 0$ impliziert.

Die Aussage in Satz 2.19 ist zunächst nur hinreichend für die Existenz einer stabilisierenden Lösung $X = X^T \geq 0$ der ARG (2.7). Kućera zeigt in [K 3] aber auch, daß die Stabilisierbarkeit von (A, B) und $Re(\lambda) \neq 0$ für alle $\lambda \in \sigma(H)$ sowohl hinreichend als auch notwendig für die Existenz dieser Lösung ist. Weiter zeigt Kućera, daß unter diesen Voraussetzungen die Entdeckbarkeit von (A, C) notwendig und hinreichend für die Eindeutigkeit dieser Lösung ist.
Damit kann man folgenden Satz beweisen, den man als Hauptsatz über die Lösung der algebraischen Riccatigleichung ansehen kann und den man in ähnlicher Form u.a. bei Casti [C 1], Knobloch/Kwakernaak [K 2] findet.

Satz 2.21 (Existenz- und Eindeutigkeitssatz)
Die ARG (2.1) besitzt eine eindeutige stabilisierende Lösung $X = X^T \geq 0$ genau dann, wenn (A, B) stabilisierbar und (A, C) entdeckbar ist.

Beweis: Siehe Kućera [K 3], Theorem 3. □

Bemerkung: Eine Tabelle, in der die Zusammenhänge zwischen positiv semidefiniten Lösungen der ARG, der stabilisierenden Lösung der ARG, Stabilisierbarkeit von (A, B) und Entdeckbarkeit von (A, C) auf übersichtliche Weise dargestellt sind, findet man bei Casti [C 1], S. 212.

2.3 Bestimmung invarianter Teilräume Hamiltonischer Matrizen mit symplektischen Transformationen

Im vorigen Abschnitt hat man gesehen, daß die Lösung der ARG äquivalent zur Bestimmung eines invarianten Teilraums der zugehörigen Hamiltonischen Matrix ist. Da die Berechnung von invarianten Teilräumen oft mit Hilfe von Ähnlichkeitstransformationen geschieht und von einem Algorithmus die Ausnutzung bzw. Erhaltung der Struktur des zugrundeliegenden Problems gefordert wird, benötigt man im Falle Hamiltonischer Matrizen Transformationen, die die Hamiltonische Struktur erhalten. Dies führt auf die symplektischen Matrizen, wie das folgende Lemma (siehe z.B. Paige/Van Loan [P 1]) zeigt.

Lemma 2.22 *Es sei $H \in \mathcal{H}_{2n}$ und $S \in \mathcal{S}_{2n}$. Dann gilt*

$$\tilde{H} = S^{-1} H S \in \mathcal{H}_{2n}.$$

Beweis: Aus $S \in \mathcal{S}_{2n}$ folgt $S^T J S = J \Longleftrightarrow S^T J = J S^{-1}$ und damit

$$\begin{aligned} J\tilde{H} &= JS^{-1}HS = S^T JHS \\ &= S^T (JH)^T S = S^T H^T J^T S \\ &= S^T H^T S^{-T} J^T = (J\tilde{H})^T. \quad \square \end{aligned}$$

Sei nun

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \in \mathcal{H}_{2n}.$$

Für numerische Verfahren, die ausschließlich mit symplektischen Ähnlichkeitstransformationen arbeiten, genügt es, das $n \times n$–Feld A und die symmetrischen Matrizen F, G als Vektoren der Länge $n(n+1)/2$ abzuspeichern (zur Speicherung symmetrischer Matrizen siehe z.B. [G 1]). Man benötigt also nur ungefähr die Hälfte des Speicherbedarfs von H. Damit ist bei jeder Transformation nur ungefähr die Hälfte der Elemente von H zu berechnen, der Rest ergibt sich aus den vorhandenen Symmetrien, man spart also ca. die Hälfte der Rechenoperationen ein.
Falls die verwendeten Transformationsmatrizen zusätzlich orthogonal sind, ergeben sich daraus außerordentlich gute numerische Eigenschaften, was in

Kapitel 3.1 näher dargelegt wird (siehe dazu auch die Ausführungen nach Algorithmus 2.26 und [B 1]).

Hat man mit Hilfe symplektischer Ähnlichkeitstransformationen einen H–invarianten Teilraum gefunden, so hat dieser eine besondere Eigenschaft.

Lemma 2.23 *Es sei $H \in \mathcal{H}_{2n}$ und $S \in \mathcal{S}_{2n}$ so daß durch die ersten k, $k \leq n$, Spalten von S ein H–invarianter Teilraum aufgespannt werde, also*

$$H \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \tag{2.14}$$

mit $S_{11}, S_{21} \in \mathbf{R}^{n,k}$ und $T_{11} \in \mathbf{R}^{k,k}$. Dann ist $\chi = \operatorname{span}\{s_{*1}, \ldots, s_{*k}\}$ isotrop.

Beweis: Nach Definition 2.7 ist zu zeigen, daß $x^T J y = 0$ für alle $x, y \in \chi$ gilt. Wegen $S \in \mathcal{S}_{2n}$ gilt nach Definition 2.4 $S^T J S = J$. Daraus folgt sofort

$$s_{*i}^T J s_{*j} = 0 \quad \text{für} \quad 1 \leq i, j \leq n$$

und damit die Behauptung. □

Folgerung 2.24 *Gilt für χ aus Lemma 2.23 $\dim \chi = n$, d.h. $k = n$, dann ist χ ein Lagrange–Teilraum.*

Beweis: Mit Lemma 2.23 und der Bemerkung nach Definition 2.4. □

Die Berechnung invarianter Teilräume beliebiger Matrizen geschieht häufig über die Berechnung ihrer Schurform, vgl. Definition 2.3. Dies kann effizient und numerisch rückwärts stabil mit Hilfe des QR–Algorithmus (siehe z.B. [G 1], [S 1] oder [W 1]) geschehen.
Paige/Van Loan [P 1] konnten für den Spezialfall Hamiltonischer Matrizen zeigen, daß eine strukturierte Schurform existiert.

Satz 2.25 (Reelle Hamiltonische Schurzerlegung)
Sei

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \in \mathcal{H}_{2n}.$$

Falls für alle Eigenwerte $\lambda \neq 0$ von H $Re(\lambda) \neq 0$ gilt, dann existiert eine Matrix $Q \in \mathcal{US}_{2n}$, so daß

$$Q^T H Q = \begin{bmatrix} T & R \\ 0 & -T^T \end{bmatrix}, \quad T, R \in \mathbf{R}^{n,n}, \tag{2.15}$$

wobei T in reeller Schurform und R symmetrisch ist. Dabei kann Q so gewählt werden, daß alle Eigenwerte von T negativen Realteil besitzen.

Kann man die Matrix Q aus Satz 2.25 bestimmen, so hat man nach Satz 2.19 die gesuchte Lösung der ARG gefunden.

Da der QR–Algorithmus ein effizientes stabiles Verfahren zur Bestimmung der Schurform gewöhnlicher Matrizen liefert, liegt es nahe, diesen auf Hamiltonische Matrizen zu spezialisieren. Dies leistet der folgende Algorithmus, den man in [B 1] und [M 1] findet.

Algorithmus 2.26 (Hamiltonischer QR–Algorithmus)
Eingabe: $H \in \mathcal{H}_{2n}$.

Ausgabe: $Q \in \mathcal{US}_{2n}$, *so daß* $\tilde{H} = Q^T HQ$ *näherungsweise in Hamiltonischer Schurform ist.*

Setze $Q = I_{2n},\ H_0 = H$.

FOR $k = 1, 2, \ldots$ UNTIL H_k *näherungsweise in Hamiltonischer Schurform*

1. *Wähle eine holomorphe Funktion* $f_k(z)$ *und faktorisiere*

$$f_k(H_{k-1}) = Q_k R_k \tag{2.16}$$

   für ein $Q_k \in \mathcal{US}_{2n}$ *so daß* $R_k \in \mathbf{R}^{2n,2n}$ *„Hamiltonische Dreiecksmatrix“ ist.*

2. *Setze* $H_k = R_k Q_k$.
3. *Setze* $Q = QQ_k$.

END FOR.

Man kann zeigen, daß für alle k $HQ = QH_k$ gilt. Damit ist Algorithmus 2.26 eine Folge orthogonal symplektischer Ähnlichkeitstransformationen und die H_k sind Hamiltonisch.
Eine Hamiltonische Dreiecksmatrix stellt nur eine Permutation einer gewöhnlichen Dreiecksmatrix dar, somit ist obiger Algorithmus ein Spezialfall des unsymmetrischen QR–Algorithmus. Da dieser unter schwachen Voraussetzungen gegen die Schurform konvergiert, siehe z.B. [G 1], [W 1], konvergiert der Hamiltonische QR–Algorithmus gegen die Hamiltonische Schurform.

Die *Shift*–Funktion f_k ist dabei so zu wählen, daß die strukturierte QR–Zerlegung in (2.16) existiert. Eine Funktion, die dies leistet, ist der *Cayley–Shift*, siehe [B 4], [M 1],

$$f_k(z) = (z - \mu_k)(z + \bar{\mu}_k)^{-1}, \tag{2.17}$$

wobei μ_k eine Approximation eines Eigenwertes von H_{k-1} darstellen sollte. Die Wahl von μ_k wird bei Byers [B 4] diskutiert, dort findet man auch eine Shift–Funktion, die ähnliche Eigenschaften hat wie (2.17), jedoch komplexe Arithmetik vermeidet.
Wählt man nun f_k aus (2.17), dann kann man leicht verifizieren, daß $f_k(H_{k-1})$ symplektisch ist, falls $H_{k-1} \in \mathcal{H}_{2n}$. Für symplektische Matrizen existiert eine strukturierte QR–Zerlegung, siehe z.B. [B 4], [M 1], so daß Algorithmus 2.26 durchführbar ist.
Da alle Ähnlichkeitstransformationen im Hamiltonischen QR–Algorithmus orthogonal und symplektisch sind, erfüllt dieser Algorithmus bei sorgfältiger Implementierung die Forderung, numerisch rückwärts stark stabil zu sein. Insbesondere gilt für die mit Algorithmus 2.26 berechneten Matrizen Q und $\tilde{H}$

$$Q^T H Q = \tilde{H} + E.$$

Dabei ist E Hamiltonisch und es gilt

$$\|E\| \leq \phi(n)\mathbf{u}\|H\|$$

für ein Polynom ϕ niedrigen Grades, siehe dazu auch Kapitel 3.1 und [B 4].

Schwierigkeiten bereitet hier jedoch die dritte Forderung aus der Einleitung an ein numerisches Verfahren, nämlich die Effizienz, d.h. der Algorithmus sollte nicht mehr als $O(n^3)$ Flops benötigen. Betrachtet man aber einen Iterationsschritt, so benötigt dieser allein schon $O(n^3)$ Flops. Da i.d.R. mindestens ein Iterationsschritt pro Eigenwertpaar von H ausgeführt werden muß, ergibt sich ein Gesamtaufwand für Algorithmus 2.26 von $O(n^4)$. Damit ist der Algorithmus in dieser Form praktisch nicht anwendbar.
Beim gewöhnlichen QR–Algorithmus löst man dieses Problem, indem man die Eingabematrix zunächst auf Hessenbergform bringt, was eine $O(n^3)$–Operation darstellt. Wie man leicht nachprüft, bleibt diese Form unter der QR–Iteration invariant. Damit reduziert sich der Aufwand pro Iterationsschritt auf $O(n^2)$ und der des gesamten QR–Algorithmus auf $O(n^3)$, für eine

genauere Analyse und eine detailierte Beschreibung dieser Vorgehensweise siehe z.B. [G 1], [S 1], [W 1].
Für den Hamiltonischen QR–Algorithmus mit Cayley–Shift konnte Byers [B 4] zeigen, daß die Hamiltonische Hessenbergform aus Definition 2.6 das Gewünschte leistet. Das Problem, einen stabilen, strukturerhaltenden, effizienten Algorithmus zur Bestimmung des stabilen invarianten Teilraums einer Hamiltonischen Matrix und damit der Lösung der ARG anzugeben, reduziert sich also auf die orthogonal symplektische Berechnung der Hamiltonischen Hessenbergform einer Matrix

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \in \mathcal{H}_{2n}.$$

Für den Fall, daß $\mathrm{rg}(G) = 1$ oder $\mathrm{rg}(F) = 1$ erfüllt ist, konnte Byers [B 4] solch einen Algorithmus angeben, für den allgemeinen Fall ist dies jedoch nicht möglich. Ammar/Mehrmann [A 1] konnten zeigen, worin die Schwierigkeit besteht.

Satz 2.27 *Sei* $H \in \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix}$. *Dann existiert* $Q \in \mathcal{US}_{2n}$, *so daß* $Q^T H Q$ *in unreduzierter Hamiltonischer Hessenbergform ist genau dann, wenn das nichtlineare Gleichungssystem*

$$\begin{aligned} x^T J H^{2i-1} x &= 0, \quad i = 1, \ldots, n-1, \\ x^T x &= 1 \end{aligned} \tag{2.18}$$

einen Lösungsvektor x *besitzt, der nicht in einem* H*–invarianten Teilraum einer Dimension kleiner oder gleich* n *enthalten ist.*

Beweis: Siehe Ammar/Mehrmann, Theorem 2.6 i). □

Ist z.B. JH positiv definit, dann existiert kein Vektor, der (2.18) erfüllt.

Bemerkung: Raines/Watkins [R 1] konnten ein äquivalentes Ergebnis für symplektische, nicht notwendig orthogonale Transformationen zeigen.

Aus dem Beweis von Ammar/Mehrmann in [A 1] zu Satz 2.27 kann man ersehen, daß die erste Spalte der Transformationsmatrix $Q \in \mathcal{US}_{2n}$ im Wesentlichen durch den Lösungsvektor x von (2.18) bestimmt wird. Die eigentliche Schwierigkeit ergibt sich damit aus der Kenntnis des folgenden Satzes [A 1].

Satz 2.28 (Impliziter Q–Satz)
Es sei $H \in \mathcal{H}_{2n}$ und $U, V \in \mathcal{US}_{2n}$, so daß

$$H_1 = U^T H U, \quad H_2 = V^T H V$$

in unreduzierter Hamiltonischer Hessenbergform sind. Falls $u_{*1} = v_{*1}$ und die Elemente von H_1 und H_2 in den Positionen $(i+1, i)$, $i = 1, \ldots, n-1$ und $(2n, n)$ positiv sind, dann gilt $u_{*j} = v_{*j}$ für alle $j = 2, \ldots, 2n$ und $H_1 = H_2$.

Beweis: Es sei $M \in \mathcal{H}_{2n}$ in Hamiltonischer Hessenbergform, d.h.

$$M \triangleq \begin{bmatrix} \diagdown\!\!\!\!\lrcorner & \square \\ * & \llcorner\!\!\!\!\diagdown \end{bmatrix}$$

und

$$P = \begin{bmatrix} I & 0 \\ 0 & \begin{smallmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{smallmatrix} \end{bmatrix},$$

dann ist PMP in gewöhnlicher Hessenbergform, d.h. die Hamiltonische Hessenbergform ist orthogonal ähnlich zur gewöhnlichen Hessenbergform.
Damit kann der gewöhnliche Implizite Q–Satz (siehe z.B. Golub/Van Loan [G 1], Theorem 7.4.2) auf PH_iP, $i = 1, 2$, PHP, PUP, PVP angewendet werden und es folgt die Behauptung. □

Damit ist die orthogonal symplektische Matrix, die H auf Hamiltonische Hessenbergform reduziert, im Wesentlichen durch ihre erste Spalte bestimmt, d.h. nach obiger Beobachtung durch den Lösungsvektor von (2.18). Da der Lösungsraum von (2.18) i.a. klein ist, verdeutlicht dies die Schwierigkeit, ein allgemein gültiges Verfahren zur Reduktion einer Hamiltonischen Matrix auf Hamiltonische Hessenbergform anzugeben. Dies ist nur dann möglich, wenn man eine Lösung von (2.18) zur Verfügung hat. Eine hinreichende Bedingung für die Existenz eines Lösungsvektors von (2.18) konnten Ammar/Mehramnn angeben [A 1].

Lemma 2.29 *Sei $H \in \mathcal{H}_{2n}$ und $\mathcal{Q} \subset \mathbf{R}^{2n}$ ein isotroper H–invarianter Teilraum. Dann erfüllen alle $x \in \mathcal{Q}$ mit $\|x\|_2 = 1$ die Bedingung (2.18).*

Beweis: Da $\mathcal{Q}$ invariant ist, gilt $H^k = y \in \mathcal{Q}$ für alle $k = 1, \ldots, 2n$ und wegen $\mathcal{Q}$ isotrop folgt $x^T J y = 0$. Da $\|x\|_2 = 1$ vorausgesetzt war, folgt die Behauptung. $\square$

Bemerkung 1: Wegen $\dim \mathcal{Q} = k \leq n$ kann mit x aus Lemma 2.29 die Transformationsmatrix Q aus Satz 2.27 bestimmt werden. Betrachtet man den dem Beweis von Satz 2.27 zugrundeliegenden *Arnoldi–Prozeß* [A 1], so sieht man, daß dieser nach k Schritten abbricht. Also sind die ersten k Spalten der transformierten Hamiltonischen Matrix die einer Hamiltonischen Hessenbergmatrix.

Bemerkung 2: Hat man einen k–dimensionalen H–invarianten Teilraum mit symplektischen Transformationen bestimmt, so ist dieser wegen Lemma 2.23 isotrop. Sind diese Transformationen außerdem orthogonal, so gilt für die Spalten von $\begin{bmatrix} S_{11} \\ S_{21} \end{bmatrix}$ aus (2.14) $s_{*i}^T s_{*i} = 1, \ i = 1, \ldots, k$. Damit erfüllt jeder Vektor aus diesem Teilraum wegen Lemma 2.29 die Bedingung (2.18), insbesondere die erste Spalte von $\begin{bmatrix} S_{11} \\ S_{21} \end{bmatrix}$.

Mit Hilfe dieser Überlegungen gelang es Ammar/Mehrmann [A 1], einen Algorithmus zur direkten Bestimmung des gesuchten H–invarianten Teilraums anzugeben. Dieser wird in Kapitel 4.1 dargestellt.

2.4 Numerische Lösungsverfahren

In diesem Abschnitt werden die wichtigsten bekannten Verfahren zur numerischen Lösung der ARG vorgestellt.

Newton–Verfahren

Das Newton–Verfahren und seine verschiedenen Varianten werden häufig zur iterativen Lösung von nichtlinearen Gleichungssystemen eingesetzt (siehe z.B. [S 1]). Da die ARG eine quadratische Matrixgleichung darstellt, liegt es nahe, darauf das Newton–Verfahren anzuwenden.

Die Iterationsvorschrift für die ARG lautet wie folgt:

Gegeben sei $X_0 \in \mathbf{R}^{n,n}$. Definiere

$$
\begin{array}{llcll}
(2.19) & A_j & = & A - GX_{j-1}, & \multirow{2}{*}{$j = 1, 2, \ldots$} \\
(2.20) & W_j & = & F + X_{j-1}GX_{j-1}, &
\end{array}
$$

und bestimme X_j als Lösung der *Lyapunovgleichung*

$$A_j^T X_j + X_j A_j = -W_j. \tag{2.21}$$

Die Lösung dieser Lyapunovgleichung erhält man z.B. mit Hilfe des *Bartels–Stewart–Algorithmus*, siehe z.B. Mehrmann [M 1], Algorithmus 11.33. Bei Mehrmann findet man auch einen Satz, der die Eigenschaften des Newton–Verfahrens zusammenfaßt.

Satz 2.30 *Sei F positiv definit und X_0 stabilisierend, so daß $A - GX_0$ c–stabil ist. Weiter sei X_j, $j = 1, 2, \ldots$, die eindeutige positiv semidefinite Lösung von (2.21) und X die eindeutige positiv semidefinite Lösung der ARG (1.1). Dann gilt für alle $j = 1, 2, \ldots$, (mit $X \geq Y$ falls $X - Y \geq 0$)*

$$0 \leq X \leq X_{j+1} \leq X_j \leq \ldots \leq X_0, \tag{2.22}$$

$$A - GX_j \quad \textit{ist c–stabil}, \tag{2.23}$$

$$\lim_{j \to \infty} X_j = X. \tag{2.24}$$

Außerdem existiert eine Konstante $\gamma > 0$ so, daß

$$\|X_j - X\| \leq \gamma \|X_{j-1} - X\|^2, \tag{2.25}$$

d.h. das Newton–Verfahren für die ARG ist quadratisch konvergent.

Unter den Voraussetzungen des Satzes konvergiert das Newton–Verfahren bei geeignet gewählter Startmatrix monoton und quadratisch gegen die gesuchte Lösung der ARG. Dies ist eine sehr erstrebenswerte Konvergenzaussage, unterliegt aber der Forderung nach einer stabilisierenden Startmatrix. Da diese i.a. aber nicht bekannt ist, muß eine stabilisierende Approximation der Lösung der ARG mit einer anderen Methode berechnet werden.
Damit ist dieses Verfahren i.d.R. zur direkten Berechnung der Lösung der ARG nicht geeignet, andererseits eignet es sich hervorragend dazu, die mit einer beliebigen Methode gefundene Lösungsapproximation zu verbessern.

Damit kann dieses Verfahren insbesondere für das Defektkorrekturverfahren in Kapitel 2.5 benutzt werden.

Der Aufwand pro Iterationsschritt beträgt $9n^3$. Hat man eine gute Approximation der Lösung als Startwert zur Verfügung, werden wegen des guten Konvergenzverhaltens nur wenige Iterationsschritte benötigt, so daß dies insgesamt ein sehr schnelles Verfahren liefert. Dabei ist allerdings zu berücksichtigen, daß für die Bestimmung der Startmatrix meist ein Vielfaches des Aufwands für einen Iterationsschritt benötigt wird, wie man der Erläuterung der folgenden Verfahren entnehmen kann.

Das Newton–Verfahren liegt als FORTRAN–Routine z.B. [B 3] bei. Die Implementierung als MATLAB–Funktion *newtonar* erwies sich als besonders einfach, da in der Toolbox CONTROL [T 2] die Funktion *lyap* für die Lösung der Lyapunovgleichung zur Verfügung gestellt wird.

Matrix–Signumfunktion–Methode

Die Beschreibung der Matrix–Signumfunktion–Methode findet man z.B. bei Byers [B 5], Casti [C 1], Mehrmann [M 1]. Zunächst benötigt man die Definition der Matrix–Signumfunktion.

Definition 2.31 *Sei $A \in \mathbf{R}^{n,n}$ mit Jordan–Normalform $J = D + N = TAT^{-1}$. Dabei sei $D = \operatorname{diag}(d_1, \ldots, d_n)$ die Diagonale von J und N der Rest. Definiert man $\sigma_i := \operatorname{sign}(Re(d_i))$, $i = 1, \ldots, n$, falls $Re(d_i) \neq 0$ und $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$, dann definiere die* **Matrix–Signumfunktion** *für A durch*

$$\operatorname{sign}(A) = T^{-1}\Sigma T. \tag{2.26}$$

Man beachte, daß $\operatorname{sign}(A)$ nicht definiert ist, falls A Eigenwerte auf der imaginären Achse hat. Weiterhin folgt für jede nichtsinguläre Matrix S

$$\operatorname{sign}\left(S^{-1}AS\right) = S^{-1}\operatorname{sign}(A)\, S. \tag{2.27}$$

Falls nun $\operatorname{sign}(A)$ definiert ist, kann man zeigen, daß die Newton–Iteration für Wurzeln der Einheitsmatrix,

$$\begin{aligned} Z_0 &= A, \\ Z_{j+1} &= Z_j - \frac{Z_j - Z_j^{-1}}{2}, \end{aligned} \tag{2.28}$$

gegen $\operatorname{sign}(A)$ konvergiert.

Betrachtet man nun die ARG (1.1), so sieht man leicht, daß diese äquivalent zu

$$
\begin{aligned}
H &= \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \\
&= \begin{bmatrix} I & 0 \\ -X & I \end{bmatrix} \begin{bmatrix} A - GX & G \\ 0 & -(A - GX)^T \end{bmatrix} \begin{bmatrix} I & 0 \\ -X & I \end{bmatrix}^{-1}
\end{aligned} \tag{2.29}
$$

ist. Ist X die stabilisierende Lösung der ARG, so ist $A - GX$ c–stabil und die Signumfuktion von H definiert. Berechnet man diese, so gilt wegen (2.27)

(2.30)

$$
W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} := \operatorname{sign}(H) = \begin{bmatrix} I & 0 \\ -X & I \end{bmatrix} \begin{bmatrix} -I & Z \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ -X & I \end{bmatrix}^{-1},
$$

wobei Z die Lyapunovgleichung

$$
(A - GX)Z + Z(A - GX)^T = -2G
$$

erfüllt. Damit ist X Lösung des überbestimmten, konsistenten linearen Gleichungssystems

$$
\begin{bmatrix} W_{12} \\ W_{22} + I \end{bmatrix} X = \begin{bmatrix} W_{11} + I \\ W_{21} \end{bmatrix}. \tag{2.31}
$$

Mit Hilfe der QR–Zerlegung von $\begin{bmatrix} W_{12} \\ W_{22} + I \end{bmatrix}$ kann dann X berechnet werden.

Wendet man die Iteration (2.28) auf eine Hamiltonische Matrix an, so ist jede Iterierte wieder Hamiltonisch. Daher kann man die Matrixinvertierung Z_j^{-1} durch die symmetrische Invertierung $(JZ_j)^{-1}J$ ersetzen, womit sich der Rechen- und Speicheraufwand für diese Operation nahezu halbiert. Diese Beobachtung sowie eine Methode zur Beschleunigung der Konvergenz der Signumfunktion–Iteration findet man bei Byers [B 5]. Auf Basis der Darstellung in [B 5] wurde die MATLAB–Funktion *signf* implementiert, siehe auch Kapitel 5.2.

I.a. ist der aus obiger Vorgehensweise resultierende Algorithmus zur Lösung der ARG nicht numerisch stabil. Daher sollte ihm mindestens ein Schritt mit

einem Defektkorrekturverfahren (siehe Kapitel 2.5) folgen. Den kompletten Algorithmus findet man bei Byers [B 5].
Der Aufwand des Verfahrens ist schwierig anzugeben, da dies von der Anzahl der Iterationsschritte in (2.28) abhängt. Erfahrungsgemäß werden meist 7 – 10 Schritte durchgeführt, wovon jeder $8n^3$ Flops benötigt, vgl. [M 1]. Die Lösung des überbestimmten Systems mit QR–Zerlegung ergibt einen Aufwand von $7n^3$ Flops.

Schurform–Methoden

Nach Lemma 2.15 ist jede Lösung der ARG durch eine bestimmte Auswahl von n Eigen- und Hauptvektoren der zugehörigen Hamiltonischen Matrix H bestimmt. Für die folgenden Verfahren sei zur Vereinfachung vorausgesetzt, daß $Re(\lambda) \neq 0$ für alle Eigenwerte von H gilt. (Diese Verfahren können auch auf Hamiltonische Matrizen mit rein imaginären Eigenwerten gerader Vielfachheit angewendet werden, da auch dann nach Lin [L 2] die Hamiltonische Schurform existiert.) Demnach ist die stabilisierende Lösung der ARG durch die zu den Eigenwerten mit negativem Realteil gehörenden Eigen- und Hauptvektoren bestimmt. Diese können mit Hilfe von Standardsoftware wie der EISPACK–Routine RG [S 2] oder der MATLAB–Funktion *eig* [T 1], die beide auf dem unsymmetrischen QR–Algorithmus beruhen, bestimmt werden. Sind V, W wie in Kapitel 2.2 durch die obere bzw. untere Hälfte dieser Vektoren gegeben, so ist V nach Lemma 2.17 invertierbar und die Lösung der ARG ist gegeben durch

$$X = -WV^{-1}.$$

Damit hat man ein erstes Verfahren zur Lösung der ARG, das in MATLAB so in den Funktionen *lqr, lqr2* [T 2] benutzt wird, um die Lösung des linear quadratischen Steuerungsproblems gemäß (1.4) zu berechnen.

Schurvektormethode nach Laub

Obwohl die Berechnung der Eigenwerte einer Matrix mit dem QR–Algorithmus numerisch rückwärts stabil ist, kann die Bestimmung der Eigen- und Hauptvektoren sehr ungenau werden, vgl. [G 1]. Diese Überlegung und eine Variante von Satz 2.19 führten Laub [L 1] auf eine Modifikation obigen Verfahrens. Berechnet man die Schurform von H mit dem QR–Algorithmus und ordnet anschließend die Eigenwerte, so daß die n stabilen Eigenwerte im $(1,1)$–Block stehen, so erhält man $Q \in \mathrm{U}_{2n}$ mit

$$(2.32) \qquad \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}^T \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix},$$

wobei $T_{11}, T_{22} \in \mathbf{R}^{n,n}$ in reeller Schurform sind und es ist $Re(\lambda) < 0$ für alle $\lambda \in \sigma(T_{11})$. (Für die Berechnung der Schurform mit dem QR–Algorithmus mit anschließendem Ordnen der Eigenwerte siehe Golub/Van Loan [G 1].)
Damit ist nach Satz 2.19 und Folgerung 2.20 die stabilisierende Lösung der ARG durch $X = -Q_{21}Q_{11}^{-1}$ gegeben. Die Spalten von $\begin{bmatrix} Q_{11} \\ Q_{21} \end{bmatrix}$ bezeichnet Laub als *Schurvektoren*, woraus sich der Name des Verfahrens ableitet.

Bemerkung: Unter der Annahme, daß Q^THQ in (2.32) die exakte Schurform von H darstellt, erhält man die Hamiltonische Schurform, indem man

$$\tilde{Q} := \begin{bmatrix} Q_{11} & -Q_{21} \\ Q_{21} & Q_{11} \end{bmatrix}$$

setzt, denn dann gilt

$$\tilde{Q}^T H \tilde{Q} = \begin{bmatrix} T_{11} & T_{12} \\ 0 & -T_{11}^T \end{bmatrix}$$

mit $T_{12} = T_{12}^T$ und $\tilde{Q} \in \mathcal{US}_{2n}$, siehe Mehrmann [M 1], Theorem 14.1.

Damit hat man ein numerisch rückwärts stabiles Verfahren zur Lösung der ARG zur Verfügung. Da der QR–Algorithmus die Hamiltonische Struktur nicht berücksichtigt, ist diese Methode jedoch nicht rückwärts stark stabil. Insbesondere wird durch Rundefehler die $(+/-)$–Struktur der Eigenwerte (vgl. Folgerung 2.12) zerstört. Bei Eigenwerten, die nahe der imaginären Achse liegen, kann dies dazu führen, daß in der berechneten Schurform nicht mehr genau n Eigenwerte stabil sind, so daß es schwierig ist, den richtigen invarianten Teilraum auszuwählen.
Clements/Glover beschreiben in [C 2] ein Deflationsverfahren (siehe auch Mehrmann [M 1]), durch das dieser Effekt vermieden wird, d.h. man erhält damit die Eigenwerte in der gewünschten Anordnung. Außerdem halbiert diese Vorgehensweise die Anzahl der QR–Schritte pro Eigenwert, so daß man für den Algorithmus nach Laub mit Clements/Glover–Deflation (unter den

üblichen Konvergenzannahmen für den QR–Algorithmus, [G 1], [S 1]) auf eine Aufwandsschätzung von $200n^3$ kommt.

In der CONTROL Toolbox [T 2] liegt dieser Algorithmus ohne den oben erwähnten Deflationsprozeß als MATLAB–Funktion *are* vor.

Hamiltonischer QR–Algorithmus

Der Rahmen für den Hamiltonischen QR–Algorithmus wurde mit Algorithmus 2.26 vorgestellt. Die Schwierigkeit dabei war, daß man eine geeignete Anfangsreduktion der Hamiltonischen Matrix H benötigt, um diesen Algorithmus effizient durchführen zu können.

In [B 4] konnte Byers für den Spezialfall, daß einer der Blöcke auf der Antidiagonalen von H Rang 1 besitzt, einen Algorithmus zur Reduktion von H auf Hamiltonische Hessenbergform angeben (vgl. beiliegende MATLAB–Funktion *hamhes*). Wie bereits in Kapitel 2.3 beschrieben, kann der Hamiltonische QR–Algorithmus mit Cayley–Shift (bzw. der bei Byers beschriebenen Shift–Funktion, mit der komplexe Arithmetik vermieden werden kann) dann effizient durchgeführt werden.

Da alle verwendeten Transformationen orthogonal und symplektisch sind und dieser Algorithmus einen Spezialfall des gewöhnlichen unsymmetrischen QR–Algorithmus darstellt, erhält man in diesem Sonderfall ein effizientes, numerisch rückwärts stark stabiles Verfahren zur Bestimmung des stabilen H–invarianten Teilraums und somit zur Lösung der ARG.

Für eine detailierte Beschreibung des Hamiltonischen QR–Algorithmus sei auf Byers [B 4] verwiesen. Unter der Annahme, daß pro Eigenwert ein Iterationsschritt benötigt wird, beträgt der geschätzte Aufwand für den gesamten Algorithmus $80n^3$.

Hamiltonischer SR–Algorithmus

Dieser Algorithmus hat die gleiche Struktur wie der QR–Algorithmus, verzichtet jedoch auf die Orthogonalität der Transformationsmatrizen. Statt der QR–Zerlegung einer Matrix bildet hier die *SR–Zerlegung* die Grundlage des Iterationsschrittes. Darunter versteht man eine Zerlegung der Form $A = SR$

mit $S \in \mathcal{S}_{2n}$ und $R \in \mathbf{R}^{2n,2n}$, wobei R in *J–Dreiecksform ist*, d.h.

$$R \triangleq \begin{bmatrix} \diagdown\!\rule{0pt}{1em} & \diagdown\!\rule{0pt}{1em} \\ {}^{\circ}\!\ddots\!{}_{\circ} & \diagdown\!\rule{0pt}{1em} \end{bmatrix}.$$

Neben orthogonal symplektischen Matrizen wie in Kapitel 3.1 beschrieben, werden hier auch *optimal konditionierte Eliminationsmatrizen vom Typ G_0* benutzt. Dies sind nichtorthogonale symplektische Matrizen, mit denen einzelne Einträge in der oberen Hälfte eines $2n$–Vektors eliminiert werden können, siehe dazu Mehrmann [M 1].
Analog zur Transformation auf Hessenbergform vor Beginn der QR–Iteration wird hier $H \in \mathcal{H}_{2n}$ zunächst auf *J–Tridiagonalform* transformiert, d.h. es wird eine Matrix $S \in \mathcal{S}_{2n}$ bestimmt, so daß

$$S^{-1}HS \triangleq \begin{bmatrix} \diagdown & \ddots \\ \diagdown & \diagdown \end{bmatrix}.$$

Unter Benutzung einer speziellen Shiftfunktion wird diese Form in der *SR–Iteration* erhalten. Für eine Beschreibung des gesamten Algorithmus siehe Mehrmann [M 1].
Insgesamt erhält man mit diesem Algorithmus eine symplektische Matrix, die eine Hamiltonische Matrix auf Hamiltonische Schurform transformiert, so daß die stabilen Eigenwerte im (1,1)–Block stehen. Damit hat man nach Satz 2.19 die gesuchte Lösung der ARG gefunden.
Durch die Verwendung nichtorthogonaler Ähnlichkeitstransformationen kann es bei schlechter Kondition dieser Matrizen zu großen Rundefehlern kommen. Dieser Effekt wird nach Mehrmann [M 1] durch die Wahl der dort angegebenen Shiftfunktion und der oben erwähnten G_0–Eliminationsmatrizen als einzigen nichtorthogonalen Transformationen zwar möglichst gering gehalten, Mehrmann gibt jedoch an, daß die erzielten Resultate oft ungenauer sind als die mit Laub's Schurvektormethode Gewonnenen. Insbesondere ist der SR–Algorithmus nicht numerisch rückwärts stabil. Daher sollte sich an den SR–Algorithmus mindestens ein Defektkorrekturschritt anschließen.
Mit einem geschätzten Aufwand von $90n^3$ Flops (ein Iterationsschritt pro Eigenwert) ist der SR–Algorithmus im Vergleich zu Laub's Verfahren relativ schnell und eignet sich damit z.B. zur Bestimmung einer Startmatrix für das

Newton–Verfahren. Da ein erheblicher Anteil des Aufwands von der Akkumulation der Transformationsmatrizen stammt, kann der SR–Algorithmus auch zur schnellen Bestimmung der Eigenwerte einer Hamiltonischen Matrix eingesetzt werden, insofern nicht höchst mögliche Genauigkeit gefordert wird.

2.5 Defektkorrektur

Berechnet man numerisch eine Lösung $\tilde{X}$ der algebraischen Riccatigleichung

$$0 = F + A^T X + XA - XGX, \tag{2.33}$$

so unterliegt diese Lösung Rundefehlern. $\tilde{X}$ ist also nur eine Approximation der Lösung X von (2.33). Zudem sind einige der in Kapitel 2.4 vorgestellten Algorithmen nicht numerisch rückwärts stabil im Sinne von Definition 2.10. Daher benötigt man ein Verfahren zur iterativen Verbesserung der Lösung. Ein solches Verfahren für die ARG stellt die *Defektkorrektur* dar, deren Grundlagen in diesem Abschnitt behandelt werden sollen.
Die Defektkorrektur beruht auf einem Prinzip, daß z.B. beim Lösen linearer Gleichungssysteme erfolgreich angewandt wird (dort *Nachiteration* genannt). Ausgehend von einer Näherungslösung $\tilde{x}$ des zugrundeliegenden Problems $Ax = b$ bestimmt man das *Residuum* $r = b - A\tilde{x}$, berechnet dann eine Lösung δ des Gleichungssystems $A\delta = r$ und setzt $x^+ = \tilde{x} + \delta$. Dieses Verfahren kann man iterieren, bis ein vorgegebenes Abbruchkriterium erfüllt ist, i.d.R. $\|r\| < \varepsilon$ oder $\|\delta\| < \varepsilon$.
Dieses Prinzip läßt sich auch auf die ARG anwenden. Dabei bildet der folgende Satz von Mehrmann/Tan [M 2] die Grundlage für ein Defektkorrekturverfahren.

Satz 2.32 *Sei $X = X^T$ eine Lösung von (2.33) und $\tilde{X}$ eine symmetrische Approximation dieser Lösung. Weiter sei $P = X - \tilde{X}$, $\tilde{A} = A - G\tilde{X}$ und $\tilde{F}$ das beim Einsetzen von $\tilde{X}$ in (2.33) entstehende Residuum der ARG, also*

$$\tilde{F} = F + A^T\tilde{X} + \tilde{X}A - \tilde{X}G\tilde{X}.$$

Dann erfüllt P die algebraische Riccatigleichung

$$0 = \tilde{F} + P\tilde{A} + \tilde{A}^T P - PGP. \tag{2.34}$$

Beweis: Setze $X = \tilde{X} + P$ in (2.33) ein, dann folgt

$$\begin{aligned}
0 &= F + A^T(\tilde{X} + P) + (\tilde{X} + P)A - (\tilde{X} + P)G(\tilde{X} + P) \\
&= F + A^T\tilde{X} + A^TP + \tilde{X}A + PA - \tilde{X}G\tilde{X} - \tilde{X}GP - PG\tilde{X} - PGP \\
&= \tilde{F} + (A^T - \tilde{X}G)P + P(A - G\tilde{X}) - \tilde{X}G\tilde{X} \\
&= \tilde{F} + \tilde{A}^TP + P\tilde{A} - \tilde{X}G\tilde{X}. \quad \square
\end{aligned}$$

Da die ARG (2.34) viele Lösungen haben kann, man aber an der eindeutigen symmetrischen, positiv semidefiniten Lösung von (2.33) interessiert ist, muß sichergestellt werden, daß man die richtige Lösung von (2.34) berechnet.

Zunächst kann man zeigen, daß sich die Eigenschaft der Stabilisierbarkeit (vgl. Def. 2.8) aus dem Problem (2.33) auf (2.34) überträgt.

Lemma 2.33 *Gegeben sei die ARG (2.33) und eine Voll–Rang–Faktorisierung $G = BB^T$ mit $\mathrm{rg}(B) = \mathrm{rg}(G)$. Ist (A, B) stabilisierbar, dann ist mit $\tilde{A} = A - G\tilde{X}$ wie in Satz 2.32 auch $(\tilde{A}, B)$ stabilisierbar.*

Beweis: Sei $K = B^T\tilde{X}$, also $\tilde{A} = A - BK$. Mit beliebigem $\lambda \in \mathbf{C}$, $Re(\lambda) \geq 0$ gilt dann

$$\begin{aligned}
\mathrm{rg}[\lambda I_n - \tilde{A}, B] &= \mathrm{rg}[\lambda I_n - A + BK, B] \\
&= \mathrm{rg}\left[[\lambda I_n - A, B]\begin{bmatrix} I_n & 0 \\ K & I_m \end{bmatrix}\right] \\
&= \mathrm{rg}[\lambda I_n - A, B].
\end{aligned}$$

Die Behauptung folgt mit Definition 2.8. $\square$

Wie in Kapitel 2.2 gesehen, kann man die positiv semdefinite Lösung der ARG (2.33) mittels des stabilen invarianten Teilraums der Hamiltonischen Matrix

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix}$$

bestimmen. Bildet man die zu der ARG für den Defekt (2.34) gehörende Hamiltonische Matrix

$$\tilde{H} = \begin{bmatrix} \tilde{A} & G \\ \tilde{F} & -\tilde{A}^T \end{bmatrix},$$

so gilt $\tilde{H} = Q^{-1}HQ$ mit

$$Q = \begin{bmatrix} I & 0 \\ -\tilde{X} & I \end{bmatrix}.$$

Da H nach Vorraussetzung genau n stabile Eigenwerte hat (vgl. Folgerung 2.12) und Q regulär ist, also H ähnlich zu $\tilde{H}$ ist, folgt sofort, daß auch $\tilde{H}$ genau n stabile Eigenwerte hat, genauer sogar $\sigma(H) = \sigma(\tilde{H})$.
Betrachte nun den $\tilde{H}$–invarianten Teilraum $\tilde{\mathcal{Q}}$ zu diesen Eigenwerten. Dieser sei aufgespannt durch die Spalten von $\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} \in \mathbf{R}^{2n,n}$, also

$$\begin{bmatrix} \tilde{A} & G \\ \tilde{F} & -\tilde{A}^T \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} Z, \tag{2.35}$$

wobei alle Eigenwerte von $Z \in \mathbf{R}^{n,n}$ negativen Realteil haben.
Nach Lemma 2.33 ist $(\tilde{A}, B)$ stabilisierbar. Wendet man Lemma 2.17 auf (2.35) an, so folgt, daß Z_1 invertierbar ist. Setzt man nun $P := -Z_2 Z_1^{-1}$ in (2.35) ein, so ergibt sich

$$\begin{bmatrix} \tilde{A} & G \\ \tilde{F} & -\tilde{A}^T \end{bmatrix} \begin{bmatrix} I \\ -P \end{bmatrix} = \begin{bmatrix} I \\ -P \end{bmatrix} \tilde{Z}, \tag{2.36}$$

wobei $\tilde{Z} = Z_1 Z Z_1^{-1}$ wieder c–stabil und P nach Lemma 2.16 symmetrisch ist.
Multipliziert man die Terme auf der linken und rechten Seite von (2.36) aus, so folgt aus der ersten Zeile, daß

$$\tilde{A} - GP = \tilde{Z} \tag{2.37}$$

c–stabil ist. Aus der zweiten Zeile von (2.36) erhält man mit (2.37)

$$\tilde{F} + \tilde{A}^T P = -P\tilde{Z} = -P\tilde{A} + PGP, \tag{2.38}$$

d.h. P löst die Defektgleichung (2.34).
Mit $\tilde{A} = A - G\tilde{X}$ folgt aus (2.37), daß auch

$$A - G(\tilde{X} + P) = \tilde{Z} \tag{2.39}$$

c–stabil ist, insbesondere gilt $\sigma(\tilde{Z}) = \sigma(Z)$ und das Spektrum von Z bildeten gerade die stabilen Eigenwerte von $\tilde{H}$, also von H. Somit ist P die richtige stabilisierende Lösung der Defektgleichung, um $\tilde{X}$ zu modifizieren.
Man erhält somit daß folgende Ergebnis (siehe [M 1]).

Satz 2.34 *Sei der invariante Teilraum zu den stabilen Eigenwerten von*

$$\tilde{H} = \begin{bmatrix} \tilde{A} & G \\ \tilde{F} & \tilde{A}^T \end{bmatrix}$$

aufgespannt durch die Spalten von $\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} \in \mathbf{R}^{2n,n}$ *mit* $\tilde{A}, \tilde{F}$ *wie in Satz 2.32 und* $P = -Z_2 Z_1^{-1}$*, dann ist* $\tilde{X} + P$ *die eindeutige symmetrische, positiv semidefinite Lösung von (2.33).*

Mit den Sätzen 2.32 und 2.34 kann man nun einen Defektkorrekturalgorithmus für die ARG formulieren.

Algorithmus 2.35 (DCORC)

Eingabe: $A, G, F \in \mathbf{R}^{n,n}$ *aus der ARG (2.33) sowie eine Toleranz* $\varepsilon > 0$ *für das Residuum.*

Ausgabe: *Eine Appoximation* $\tilde{X}$ *der Lösung von (2.33) und eine Fehlerschätzung* $P \in \mathbf{R}^{n,n}$ *mit* $\|P\| \leq \varepsilon$.

1.Schritt: *Berechne mit einer beliebigen Methode eine stabilisierende Näherungslösung* $\tilde{X}$ *von (2.33).*

2.Schritt: *Setze* $P := \tilde{X}, \quad \tilde{X} := 0.$

3.Schritt:

WHILE $\|P\| > \varepsilon$

Setze

$$\begin{aligned} \tilde{X} &:= \tilde{X} + P \\ \tilde{F} &:= F + A^T\tilde{X} + \tilde{X}A - \tilde{X}G\tilde{X} \\ \tilde{A} &:= A - G\tilde{X} \end{aligned}$$

Berechne mit einer beliebigen Methode eine stabilisierende Lösung der ARG für den Defekt

$$0 = \tilde{F} + \tilde{A}^T P + P\tilde{A} - PGP \tag{2.40}$$

END WHILE

END.

Da in Schritt 1 und Schritt 3 keine Bedingungen an die zu verwendende Methode gestellt werden (außer daß eine „stabilisierende" Lösung der jeweiligen ARG berechnet werden soll), ist es möglich, verschiedene Methoden zur Lösung von (2.33) und (2.40) zu verwenden. Da bei Konvergenz des Defektkorrekturalgorithmus $\|P\|$ klein, der quadratische Term PGP damit vernachlässigbar wird, eignet sich insbesondere das in Kapitel 2.4 beschriebene Newton–Verfahren zur Lösung von (2.40).
Die Kosten des Algorithmus bestehen im Wesentlichen aus den Kosten der in Schritt 1 und 3 verwendeten Methoden. Diese Beobachtung macht den Multishift–Algorithmus für die Defektkorrektur besonders attraktiv, falls man (2.33) und (2.40) damit löst, da die Kosten für Schritt 3 durch Ausnutzen von Information aus Schritt 1 verringert werden können, für eine genauere Analyse siehe Abschnitt 4.3.

Der Algorithmus DCORC ist numerisch rückwärts stabil im Sinne von Definition 2.10, denn $\tilde{X}$ ist die exakte Lösung der ARG mit gestörten Anfangsdaten,

$$0 = F - \tilde{F} + A^T X + XA - XGX,$$

wobei $\tilde{F}$ das Residum aus Satz 2.32 ist.
Werden die ersten signifikanten Stellen des Defekts P in Schritt 3 exakt berechnet, so konvergiert die Näherungslösung $\tilde{X}$ gegen die exakte Lösung X von (2.33). In der Praxis führt dies dazu, daß die Genauigkeit des Residuums $\tilde{F}$ durch Auslöschung bei der Subtraktion des letzten Terms abnimmt, da $\tilde{F}$ bei fortschreitender Konvergenz des Algorithmus nur noch eine Störung von F in der Größenordnung der Maschinengenauigkeit darstellt. Dadurch können dann auch die führenden signifikanten Stellen des Defekts P beeinträchtigt werden.
Aufgrund von numerischen Erfahrungen gibt Mehrmann in [M 1] jedoch an, daß auch bei Erreichen dieser Genauigkeitsgrenze in den meisten Fällen $\|P\| \leq 10\|X - \tilde{X}\|$ gilt. Bei den in Kapitel 5.3 untersuchten Beispielen wird diese Genauigkeit (bei Konvergenz der Defektkorrektur) oft bereits nach ein oder zwei Schritten von DCORC erreicht.

Kapitel 3

Orthogonal symplektische Algorithmen für Hamiltonische Matrizen

In diesem Kapitel werden zunächst die bereits in Kapitel 2.3 eingeführten orthogonal symplektischen Matrizen näher betrachtet. Mit deren Hilfe ist es möglich, einen strukturerhaltenden Algorithmus anzugeben, der Hamiltonische Matrizen auf eine Hessenberg–ähnliche Gestalt transformiert. In Abschnitt 3.3 wird ein Algorithmus vorgestellt, mit dem die Eigenwerte einer Hamiltonischen Matrix unter Ausnutzung der Struktur berechnet werden können. Im letzten Abschnitt dieses Kapitels wird ein effektives Aufdatierungsverfahren für die Lösung der ARG beschrieben.
Diese Algorithmen bilden die Grundlage für einen orthogonal symplektischen Algorithmus zur Lösung der ARG, der in Kapitel 4 konstruiert wird.

3.1 Orthogonal symplektische Transformationsmatrizen

In Lemma 2.22 wurde gezeigt, daß Ähnlichkeitstransformationen mit symplektischen Matrizen die Hamiltonische Struktur einer Matrix erhalten. Will man Hamiltonische Matrizen in eine spezielle Form Hamiltonischer Struktur bringen, so sollten die dazu benutzten Transformationsmatrizen demnach symplektisch sein.

Andererseits sollen bei numerischen Berechnungen möglichst geringe Rundefehler gemacht werden. Daher verwendet man für Ähnlichkeitstransformationen bevorzugt orthogonale Matrizen.
Insgesamt ist es daher wünschenswert, bei numerischen Verfahren für Hamiltonische Matrizen orthogonale, symplektische Transformationen zu benutzen. Die Menge dieser Matrizen wurde in Definition 2.5 mit $\mathcal{US}_{2n}$ bezeichnet. Über deren Elemente kann man folgende Aussage treffen (vgl. [P 1],[V 1]).

Lemma 3.1 *Sei* $\mathcal{US}_{2n} = \mathcal{S}_{2n} \cap \mathcal{U}_{2n}$. *Dann gilt für alle*

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \in \mathcal{US}_{2n}$$

$Q_{11} = Q_{22}$ *und* $Q_{21} = -Q_{12}$ *bzw.*

$$\mathcal{US}_{2n} = \{Q \in \mathbf{R}^{2n,2n} \mid Q^T Q = I,\ Q = \begin{bmatrix} Q_1 & Q_2 \\ -Q_2 & Q_1 \end{bmatrix},\ Q_1, Q_2 \in \mathbf{R}^{n,n}\}.$$

Beweis: Aus $Q \in \mathcal{S}_{2n}$ folgt

$$J = Q^T J Q \Longleftrightarrow I = Q^T J Q J^T$$

und wegen $Q \in \mathcal{U}_{2n}$ gilt damit

$$Q = JQJ^T \Longleftrightarrow \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} = \begin{bmatrix} Q_{22} & -Q_{21} \\ -Q_{12} & Q_{11} \end{bmatrix}. \quad \square$$

In den beiden folgenden Abschnitten werden zwei Typen orthogonal symplektischer Matrizen genauer betrachtet. Im dritten Abschnitt wird noch einmal allgemeiner auf diese Transformationen eingegangen und deren Rundefehlerverhalten analysiert.

3.1.1 Householder–Matrizen

Eine *Householder*–Matrix wird definiert durch

$$(3.1)\quad H(k,j,v) = I_n - 2vv^T/v^T v, \quad v = [0,\ldots,0,v_k,\ldots,v_j,0,\ldots,0]^T.$$

Dabei wird v als *Householder*–Vektor bezeichnet (z.B. [G 1]). $H(k,j,v)$ ist eine Modifizierung der Einheitsmatrix mit der Rang–1–Matrix $2vv^T/v^Tv$. Wie man leicht nachrechnet, ist $H(k,j,v)$ orthogonal und symmetrisch, also

$$H(k,j,v) = H^T(k,j,v) = H^{-1}(k,j,v).$$

Eine wesentliche Anwendung von Householdermatrizen ist es, bestimmte Einträge eines Vektors zu eliminieren. Der folgende Algorithmus HHGEN (von engl. HouseHolder elimination matrix GENeration) bestimmt v so, daß die Komponenten $k+1,\ldots,j$ von $H(k,j,v)x$ Null sind.

Algorithmus 3.2 (HHGEN)
Eingabe: $x \in \mathbf{R}^n$, $k, j \in \{1,\ldots,n\}$ *mit* $k \leq j$.
Ausgabe: $v \in \mathbf{R}$, $\beta = 2/v^Tv$, *so daß mit* $H(k,j,v) = I - \beta vv^T$ *gilt*
$H(k,j,v)x = [x_1,\ldots,x_{k-1},\tilde{x}_k,0,\ldots,,0,x_{j+1},\ldots,x_n]^T$.

$$\begin{aligned}
&v = 0\\
&\mu = \max\{|x_k|,\ldots,|x_j|\}\\
&\alpha = 0\\
&\text{FOR} \quad i = k,\ldots,j\\
&\qquad v_i = x_i/\mu\\
&\qquad \alpha = \alpha + v_i^2\\
&\text{END FOR}\\
&\alpha = \sqrt{\alpha}\\
&\beta = 1/(\alpha(\alpha + |v_k|))\\
&v_k = v_k + \operatorname{sign}(v_k)\alpha
\end{aligned}$$

END.

Ein Aufruf von HHGEN wird durch $[v,\beta] = \text{HHGEN}(k,j,x)$ zitiert. Obiger Algorithmus benötigt $3(j-k)$ Flops.
Wendet man Householdermatrizen an, um bestimmte Einträge einer Matrix $A \in \mathbf{R}^{n,n}$ zu eliminieren, so benötigt man die Produkte $H(k,j,v)A$ bzw. $AH(k,j,v)$. Dabei ist es nicht notwendig, die Matrix $H(k,j,v)$ explizit zu bilden. Setzt man $w = \beta A^Tv$, so gilt

$$H(k,j,v)A = A - vw^T \tag{3.2}$$

bzw. mit $w = \beta Av$

$$AH(k,j,v) = A - wv^T. \tag{3.3}$$

Hat man v und β mit HHGEN berechnet, so lassen sich die Links- bzw. Rechtsmultiplikation von $H(k,j,v)$ mit $A \in \mathbf{R}^{n,n}$ mit den Formeln (3.2) und (3.3) realisieren. Nutzt man die Struktur von v aus, so können die Produkte $w = \beta A^T v$ und $w = \beta A v$ mit je $2n(j-k)$ Flops gebildet werden. Die Rang–1–Aufdatierungen (3.2) und (3.3) benötigen jeweils $2n(j-k)$ Flops, so daß der Aufwand für die Berechnung der Produkte $H(k,j,v)A$ und $AH(k,j,v)$ jeweils $4n(j-k)$ Flops beträgt.
Als *symplektische* Householdermatrizen bezeichnet man Matrizen der Form

$$(3.4) \qquad H_S(k,j,v) = \begin{bmatrix} H(k,j,v) & 0 \\ 0 & H(k,j,v) \end{bmatrix},$$

wobei $H(k,j,v)$ gemäß (3.1) definiert ist. Offensichtlich erfüllt $H_S(k,j,v)$ die Bedingungen aus Lemma 3.1, es gilt somit $H_S(k,j,v) \in \mathcal{US}_{2n}$.
Bildet man für $H \in \mathbf{R}^{2n,2n}$ die Produkte $H_S(k,j,v)H$ bzw. $HH_S(k,j,v)$, so entspricht dies der Rechts- bzw. Linksmultiplikation der $n \times n$–Blöcke von H mit $H(k,j,v)$ wie oben beschrieben.

3.1.2 Givens–Matrizen

Eine reelle *Givens*–Matrix hat die Form

$$(3.5) \qquad G(k,j,c,s) = \begin{array}{cc} \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & c & \dots & -s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & s & \dots & c & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix} & \begin{array}{c} \\ \\ k \\ \\ j \\ \\ \\ \end{array} \\ \begin{array}{ccccccc} & & k & & j & & \end{array} & \end{array}$$

oder ausgedrückt durch äußere Produkte

$$(3.6) \qquad G(k,j,c,s) = I + (c-1)(e_k e_k^T + e_j e_j^T) + s(e_j e_k^T - e_k e_j^T)$$

mit $c, s \in \mathbf{R}$ und $c^2 + s^2 = 1$. Offensichtlich ist $G(k,j,c,s)$ orthogonal.

Givens–Matrizen (oder Givens–Rotationen) werden dazu benutzt, einen bestimmten Eintrag eines Vektors oder einer Matrix zu eliminieren, vgl. Golub/Van Loan [G 1]. Der folgende Algorithmus GIVGEN (von engl. GIVens elimination matrix GENeration) berechnet c, s so, daß die j–te Komponente von $G^T(k, j, c, s)x$ Null ist.

Algorithmus 3.3 (GIVGEN)
Eingabe: $x \in \mathbf{R}^n$, $k, j \in \{1 \ldots, n\}$ *mit* $k \leq j$.
Ausgabe: $c, s \in \mathbf{R}$ *so, daß* $G^T(k, j, c, s)x = y$ *mit* $y_i = x_i$ *für* $i \neq j, k$,
$y_k = \pm\sqrt{x_k^2 + x_j^2}$ *und* $y_j = 0$.

```
  IF x_j = 0 THEN
     c = 1,   s = 0
  ELSE
     IF |x_j| ≥ |x_k| THEN
        t  =  x_k/x_j
        s  =  sqrt(1/(1 + t^2))
        c  =  st
     ELSE
        t  =  x_j/x_k
        c  =  sqrt(1/(1 + t^2))
        s  =  ct
     END IF
  END IF
END.
```

Ein Aufruf von GIVGEN wird im Folgenden mit $[c, s] = \text{GIVGEN}(k, j, x)$ zitiert. Dieser Algorithmus benötigt 6 Flops.
Betrachtet man wieder die Produkte $G^T(k, j, c, s)A$ und $AG(k, j, c, s)$ für $A \in \mathbf{R}^{n,n}$, so sieht man leicht, daß im ersten Fall nur die Zeilen k und j bzw. im zweiten Fall die Spalten k und j von A verändert werden. Algorithmisch läßt sich dies wie folgt formulieren.
Für $A \leftarrow G^T(k, j, c, s)A$

```
FOR  i = 1, ..., n
   α       =  cA(k,i)   +  sA(j,i)
   A(j,i)  =  -sA(k,i)  +  cA(j,i)
   A(k,i)  =  α
END
```

und für $A \leftarrow AG(k,j,c,s)$

$$
\begin{array}{llll}
\text{FOR} & i = 1, \ldots, n & & \\
\quad \alpha & = & cA(i,k) & + \; sA(i,j) \\
\quad A(i,j) & = & -sA(i,k) & + \; cA(i,j) \\
\quad A(i,k) & = & \alpha & \\
\text{END.} & & &
\end{array}
$$

Beide Algorithmen benötigen $6n$ Flops.

Soll eine Givens–Matrix zusätzlich symplektisch sein, so folgt aus Lemma 3.1, daß $j = n + k$ gelten muß. Eine *symplektische* Givens–Matrix hat somit folgende Gestalt

$$
(3.7) \qquad G_S(k,c,s) = \left[\begin{array}{ccccc|ccccc}
1 & \ldots & & & 0 & 0 & \ldots & & & 0 \\
\vdots & \ddots & & & \vdots & \vdots & \ddots & & & \vdots \\
& & c & & & & & -s & & \\
& & & \ddots & & & & & \ddots & \\
0 & \ldots & & & 1 & 0 & \ldots & & & 0 \\
\hline
0 & \ldots & & & 0 & 1 & \ldots & & & 0 \\
\vdots & \ddots & & & \vdots & \vdots & \ddots & & & \vdots \\
& & s & & & & & c & & \\
& & & \ddots & & & & & \ddots & \\
0 & \ldots & & & 0 & 0 & \ldots & & & 1
\end{array}\right]
$$

bzw. $G_S(k,c,s) = \begin{bmatrix} C & -S \\ S & C \end{bmatrix}$ mit $C = I_n - (c-1)e_k e_k^T$ und $S = s e_k e_k^T$. Man beachte, daß symplektische Givens–Matrizen z.B. bei Paige/Van Loan [P 1] als symplektische *Jacobi*–Matrizen bezeichnet werden.

Die Produkte $HG_S(k,c,s)$ und $G_S^T(k,c,s)H$ mit $H \in \mathbf{R}^{2n,2n}$ können analog zu den oben beschriebenen Produkten mit gewöhnlichen Givens–Matrizen gebildet werden. Bei Ähnlichkeitstransformationen $G_S^T(k,c,s)HG_S(k,c,s)$ für Hamiltonischen Matrizen H kann man zudem noch deren Struktur ausnutzen, was den Aufwand nahzu halbiert.

Bemerkung: Die Tatsache, daß Elemente eines Vektors oder einer Matrix nur durch Rotationen in der $(k, n+k)$–Ebene eliminiert werden können, schränkt die Anwendbarkeit der symplektischen Givens–Matrizen ein. Insbesondere ist es daher i.a. nicht möglich, den (2,1)–Block Hamiltonischer Matrizen mit orthogonal symplektischen Ähnlichkeitstransformationen zu eliminieren (vgl. Abschnitt 3.2).

3.1.3 Eigenschaften der Gruppe $\mathcal{US}_{2n}$

In den beiden vorangegangenen Abschnitten wurden spezielle orthogonal symplektische Matrizen vorgestellt. Paige und Van Loan geben in [P 1] ein Ergebnis an, das erklärt, warum man sich bei der Betrachtung orthogonal symplektischer Matrizen auf diese beiden Typen beschränken kann.

Lemma 3.4 *Sei $Q \in \mathcal{US}_{2n}$. Dann ist Q als Produkt symplektischer Householder– und Givens–Matrizen darstellbar.*

Beweis: Paige/Van Loan [P 1], Corollary 2.2. □

Aussagen über die Fehlerentwicklung bei Multiplikationen und Ähnlichkeitstransformationen mit orthogonal symplektischen Matrizen lassen sich leicht auf die Standardtheorie für Householder– und Givens–Matrizen nach Wilkinson [W 1] zurückführen. Zunächst gilt für die von den Prozeduren HHGEN und GIVGEN erzeugten Matrizen $fl(H(k,j,v))$ oder $fl(G(k,j,c,s))$

$$fl(H(k,j,v)) - H(k,j,v) = E$$

bzw.

$$fl(G(k,j,c,s)) - G(k,j,c,s) = E$$

mit $\|E\| = O(\mathbf{u})$.
Sei nun P eine beliebige Householder– oder Givens–Matrix und $\tilde{P} := fl(P)$. Dann gilt für Multiplikationen beliebiger $n \times n$–Matrizen A mit P

$$fl(A\tilde{P}) = (A+E)P$$

bzw.

$$fl(\tilde{P}^T A) = P^T(A+E)$$

mit $\|E\| \approx \mathbf{u}\|A\|$. Damit gilt für Ähnlichkeitstransformationen

$$fl(\tilde{P}^T A \tilde{P}) = P^T(A+E)P$$

mit $\|E\| \approx \mathbf{u}\|A\|$. Da eine symplektische Givens–Matrix eine Givens–Matrix im üblichen Sinne ist, gelten obige Betrachtungen analog für $P = G_S(k, c, s)$. Bei Multiplikationen und Ähnlichkeitstransformationen mit symplektischen Householdermatrizen kann man sich auf die Betrachtung der Blockmatrizen beschränken, denn mit $P = H(k, j, v)$ und $M \in \mathbf{R}^{2n,2n}$ gilt

$$H_S(k,j,v)MH_S(k,j,v) = \begin{bmatrix} PM_{11}P & PM_{12}P \\ PM_{21}P & PM_{22}P \end{bmatrix}.$$

Daher gelten für die Blockmatrizen obige Betrachtungen und damit offensichtlich auch

$$fl(\tilde{H}_S(k,j,v)M\tilde{H}_S(k,j,v)) = H_S(k,j,v)(M+E)H_S(k,j,v)$$

mit $E = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix}$ und $\|E_{ij}\| \approx \mathbf{u}\|M_{ij}\|$, $i,j = 1,2$ und insgesamt $\|E\| \approx \mathbf{u}\|M\|$.
Ist M zusätzlich Hamiltonisch, so sind Ähnlichkeitstransformationen mit Matrizen $Q \in \mathcal{US}_{2n}$ strukturerhaltend und wird dies bei der Implementierung beachtet, so ist auch E Hamiltonisch. Also ist die Matrix $\tilde{M} = fl(\tilde{Q}^T M \tilde{Q})$ das Ergebnis einer orthogonal symplektischen Ähnlichkeitstransformation einer Matrix gleicher Struktur mit leicht gestörten Anfangsdaten. Damit sind solche Ähnlichkeitstransformationen numerisch rückwärts stark stabil im Sinne von Definition 2.10.

Anwendungen orthogonal symplektischer Matrizen sind die von Paige/Van Loan [P 1] vorgestellte Reduktion einer Hamiltonischen Matrix auf eine Hessenberg–ähnliche Gestalt (siehe Abschnitt 3.2) und die Bestimmung der Eigenwerte einer Hamiltonischen Matrix nach Van Loan [V 1] (siehe Abschnitt 3.3). Weitere Anwendungen findet man bei Bunse–Gerstner/Byers/Mehrmann [B 2].
An dieser Stelle soll noch kurz beschrieben werden, wie mit symplektischen Householder– und Givens–Matrizen ein Vektor $z \in \mathbf{R}^{2n}$ auf ein Vielfaches des ersten Einheitsvektors im $\mathbf{R}^{2n}$ abgebildet werden kann.

Algorithmus 3.5 (COLRED [COLumn REDuction])
Eingabe: $z \in \mathbf{R}^{2n}$
Ausgabe: $v_1, v_2 \in \mathbf{R}^n$, $\beta_1, \beta_2, c, s \in \mathbf{R}$, $\quad z = [\alpha, 0, \ldots, 0]^T$, *wobei*
$z = H_S(1,n,v_2)G_S^T(1,c,s)H_S(1,n,v_1)z$, $\alpha = \pm\|z\|_2$.

1.Schritt:

$$\begin{array}{lcl} x & = & [z_{n+1}, \ldots, z_{2n}]^T \\ [v_1, \beta_1] & = & \mathrm{HHGEN}(1, n, x) \\ z & = & H_S(1, n, v_1) z \end{array}$$

2.Schritt:

$$\begin{array}{lcl} x & = & [z_1, z_{n+1}]^T \\ [c, s] & = & \mathrm{GIVGEN}(1, 2, x) \\ z & = & G_S^T(1, c, s) z \end{array}$$

3.Schritt:

$$\begin{array}{lcl} x & = & [z_1, \ldots, z_n]^T \\ [v_2, \beta_2] & = & \mathrm{HHGEN}(1, n, x) \\ z & = & H_S(1, n, v_2) z \end{array}$$

END.

Mit den Aufwandsbetrachtungen für HHGEN und GIVGEN folgt, daß Algorithmus 3.5 $6n$ Flops zur Erzeugung der Transformationen benötigt. Die Produkte der Householder– und Givens–Matrizen mit z lassen sich analog zu den in den Abschnitten 3.1.1 und 3.1.2 beschriebenen Produkten bilden. Berechnet man dabei nur die Nicht–Null–Elemente von z und berücksichtigt, daß z_{n+1} im ersten Schritt und z_1 im dritten Schritt im Prinzip bereits von HHGEN geliefert werden, so ergibt sich dafür ein Aufwand von $4n$ Flops, also insgesamt für COLRED $10n$ Flops.
Die Householdervektoren v_1, v_2 können in den eliminierten Komponenten von z gespeichert, c, s in z_{n+1} codiert werden. Zur Implementierung von COLRED siehe Kapitel 5.1.4.

3.2 Orthogonal symplektische Transformation auf Hessenberg–ähnliche Gestalt

Paige/Van Loan geben in [P 1] eine Methode an, die eine beliebige Matrix $M \in \mathbf{R}^{2n,2n}$ mit einer orthogonal symplektischen Matrix Q so transformiert, daß

$$Q^T M Q = \tilde{M} = \begin{bmatrix} \tilde{M}_{11} & \tilde{M}_{12} \\ \tilde{M}_{21} & \tilde{M}_{22} \end{bmatrix} \triangleq \begin{bmatrix} \diagdown & \square \\ \diagdown & \square \end{bmatrix}, \tag{3.8}$$

wobei $\tilde{M}_{11}$ eine obere Hessenbergmatrix und $\tilde{M}_{21}$ eine obere Dreiecksmatrix ist. Ist M zusätzlich Hamiltonisch, so folgt wegen $Q \in \mathcal{US}_{2n}$ und Lemma 2.22, daß $\tilde{M} \in \mathcal{H}_{2n}$. Damit gilt $\tilde{M}_{22} = -\tilde{M}_{11}^T$ und $\tilde{M}_{21} = \tilde{M}_{21}^T$, d.h. $\tilde{M}_{21}$ ist diagonal.
Der grundlegende Bestandteil dieser Transformation ist ein Algorithmus, der einen Vektor $x = \begin{bmatrix} y \\ z \end{bmatrix} \in \mathbf{R}^{2n}$ mit symplektischen Householder- und Givens–Transformationen auf $\begin{bmatrix} v \\ w \end{bmatrix}$ so abbildet, daß $y_i = v_i$, $z_i = w_i$ für $1 \leq i \leq k-2$, $v_i = 0$, $k+1 \leq i \leq n$ und $w_i = 0$ für $k \leq i \leq n$.

Algorithmus 3.6
Eingabe: $x = \begin{bmatrix} y \\ z \end{bmatrix} \in \mathbf{R}^{2n}, \quad k \in \{1, \ldots, n\}$.

Ausgabe: $Q \in \mathcal{US}_{2n}$ *mit* $Q^T \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} v \\ w \end{bmatrix}$ *mit* v, w *wie oben beschrieben.*

Setze $Q = I_{2n}$

1.Schritt:

$$\begin{aligned} [v_1, \beta_1] &= \text{HHGEN}(k, n, z) \\ \begin{bmatrix} y \\ z \end{bmatrix} &= H_S(k, n, v_1) \begin{bmatrix} y \\ z \end{bmatrix} \\ Q &= Q H_S(k, n, v_1) \end{aligned}$$

2. Schritt:

$$\begin{aligned} [c, s] &= \text{GIVGEN}(k, n+k, \begin{bmatrix} y \\ z \end{bmatrix}) \\ \begin{bmatrix} y \\ z \end{bmatrix} &= G_S^T(k, c, s) \begin{bmatrix} y \\ z \end{bmatrix} \\ Q &= Q G_S(k, c, s) \end{aligned}$$

3.Schritt:

$$\begin{aligned} [v_2, \beta_2] &= \text{HHGEN}(k, n, y) \\ \begin{bmatrix} y \\ z \end{bmatrix} &= H_S(k, n, v_2) \begin{bmatrix} y \\ z \end{bmatrix} \\ Q &= Q H_S(k, n, v_2) \end{aligned}$$

END.

Damit läßt sich die in (3.8) beschriebene Reduktion für Matrizen $M \in \mathbf{R}^{2n,2n}$ herleiten.
Dazu nehme an, daß $Q_1, \ldots, Q_{k-1} \in \mathcal{US}_{2n}$ gemäß Algorithmus 3.6 bestimmt worden sind, so daß (mit $M_0 := M$)

$$\begin{aligned} M_{k-1} &= (Q_1 \cdots Q_{k-1})^T M (Q_1 \cdots Q_{k-1}) \\ &= \left[\begin{array}{cc|cc} M_{11} & M_{12} & M_{13} & M_{14} \\ ue_k^T & M_{22} & M_{23} & M_{24} \\ \hline M_{31} & M_{32} & M_{33} & M_{34} \\ ve_k^T & M_{42} & M_{43} & M_{44} \end{array}\right] \begin{array}{l} \}\, k \\ \}\, n-k \\ \}\, k \\ \}\, n-k \end{array}, \end{aligned}$$

wobei M_{11} obere Hessenbergmatrix, M_{31} obere Dreiecksmatrix und e_k der k–te Einheitsvektor in $\mathbf{R}^k$ ist. Mit dem Algorithmus 3.6 kann man eine orthogonal symplektische Matrix

$$Q_k = H_S(k+1, n, v_1) G_S(k+1, c, s) H_S(k+1, n, v_2)$$

bestimmen, so daß

$$Q_k^T \begin{bmatrix} 0 \\ u \\ 0 \\ v \end{bmatrix} = \left[\begin{array}{cc|cc} I_k & 0 & 0 & 0 \\ 0 & Q_{11}^{(k)^T} & 0 & Q_{12}^{(k)^T} \\ \hline 0 & 0 & I_k & 0 \\ 0 & -Q_{12}^{(k)^T} & 0 & Q_{11}^{(k)^T} \end{array}\right] \begin{bmatrix} 0 \\ u \\ 0 \\ v \end{bmatrix} = \alpha \begin{bmatrix} 0 \\ e_k \\ 0 \\ 0 \end{bmatrix}.$$

Daraus folgt

$$M_k = Q_k^T M_{k-1} Q_k = \left[\begin{array}{cc|cc} M_{11} & \tilde{M}_{12} & M_{13} & \tilde{M}_{14} \\ \alpha e_1 e_k^T & \tilde{M}_{22} & \tilde{M}_{23} & \tilde{M}_{24} \\ \hline M_{31} & \tilde{M}_{32} & M_{33} & \tilde{M}_{34} \\ 0 & \tilde{M}_{42} & \tilde{M}_{43} & \tilde{M}_{44} \end{array}\right],$$

wobei die Tilde die Blöcke bezeichnet, die durch die Ähnlichkeitstransformation verändert werden.
Führt man diese Prozedur für $k = 1, \ldots, n-1$ durch, so hat M_{n-1} offensichtlich die in (3.8) beschriebene Form. Dabei ist im $(n-1)$–ten Schritt $u = \begin{bmatrix} m_{n-1,n-1}^{(n-2)} \\ m_{n,n-1}^{(n-2)} \end{bmatrix}$ und $v = \begin{bmatrix} m_{2n-1,n-1}^{(n-2)} \\ m_{2n,n-1}^{(n-2)} \end{bmatrix}$, es ist also nur noch eine Givens–Rotation zur Eliminierung von $m_{2n,n-1}^{(n-2)}$ nötig.

Damit lässt sich der Algorithmus PVLRED (von engl. Paige/Van Loan REDuction) formulieren, der eine beliebige Matrix $H \in \mathbf{R}^{2n,2n}$ auf die Hessenberg– ähnliche Gestalt (3.8) bringt.

Algorithmus 3.7 (PVLRED)
Eingabe: $M \in \mathbf{R}^{2n,2n}$

Ausgabe: $Q \in \mathcal{US}_{2n}$, *so daß* $Q^T M Q = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$, *wobei* M_{11} *obere Hessenbergmatrix und* M_{21} *obere Dreiecksmatrix ist.* M *wird mit der transformierten Matrix überschrieben.*

Setze $Q = I_{2n}$

FOR $k = 1, \ldots, n-1$

IF $k \leq n-2$ THEN

$$\begin{aligned}
\begin{bmatrix} y \\ z \end{bmatrix} &= Me_k \\
[v, \beta] &= \text{HHGEN}(k+1, n, z) \\
M &= H_S(k+1, n, v) M H_S(k+1, n, v) \\
Q &= Q H_S(k+1, n, v)
\end{aligned}$$

END IF

$$\begin{aligned}
x &= Me_k \\
[c, s] &= \text{GIVGEN}(k+1, n+k+1, x) \\
M &= G_S^T(k+1, c, s) M G_S(k+1, c, s) \\
Q &= Q G_S(k+1, c, s)
\end{aligned}$$

IF $k \leq n-2$ THEN

$$\begin{aligned}
\begin{bmatrix} y \\ z \end{bmatrix} &= Me_k \\
[v, \beta] &= \text{HHGEN}(k+1, n, y) \\
M &= H_S(k+1, n, v) M H_S(k+1, n, v) \\
Q &= Q H_S(k+1, n, v)
\end{aligned}$$

END IF

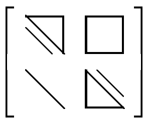
END FOR

END.

Dabei sollten die Ähnlichkeitstransformationen von M mit den Householder– und Givens–Matrizen entsprechend den in den Abschnitten 3.1.1 und 3.1.2 beschriebenen Verfahren gebildet werden.
Zur Erzeugung der Transformationen benötigt man dann $5n^2$ Flops, für die Ähnlichkeitstransformationen mit M $\frac{88}{3}n^3$ Flops und für die Aufdatierungen von Q $16n^3$ Flops. Berücksichtigt man, daß wegen $Q = \begin{bmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{bmatrix}$ nur Q_1 und Q_2 berechnet werden müssen, so halbiert sich der Aufwand für die Aufdatierung von Q.
Ist die Eingabematrix M Hamiltonisch (oder eine quadrierte Hamiltonische Matrix, vgl. den folgenden Abschnitt), so kann der Aufwand für die Ähnlichkeitstransformationen durch Ausnutzung der Symmetrien auf $14n^3$ verringert werden, siehe dazu auch Abschnitt 5.1.5.

Die mit PVLRED berechnete Hessenberg–ähnliche Form einer Hamiltonischen Matrix, im Folgenden als **PVL–Form** bezeichnet,

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \triangleq \qquad (3.9)$$



mit A obere Hessenbergmatrix, $F = \operatorname{diag}(f_1, \ldots, f_n)$ und $G = G^T$, genügt noch nicht den Bedingungen, die zur Eingabe in den Hamiltonischen QR–Algorithmus 2.26 nötig sind, es sei denn, es gilt $f_1 = \ldots = f_{n-1} = 0$. Dann ist M eine Hamiltonische Hessenbergmatrix, und der Hamiltonische QR–Algorithmus ist durchführbar, vgl. Kapitel 2.3. Wegen der in der Bemerkung am Ende von Abschnitt 3.1.2 erwähnten Einschränkung ist es i.a. aber nicht möglich, die f_i mit endlich vielen orthogonal symplektischen Transformationen zu eliminieren.
In Kapitel 4 wird jedoch gezeigt, daß mit Hilfe von PVLRED der invariante Teilraum zu den stabilen Eigenwerten einer Hamiltonischen Matrix berechnet werden kann.

3.3 Bestimmung der Eigenwerte einer Hamiltonischen Matrix

Die Berechnung der Eigenwerte einer Hamiltonischen Matrix kann über den unsymmetrischen QR–Algorithmus (vgl. z.B. [G 1]) erfolgen, wobei aber die

Hamiltonische Struktur der Matrix nicht berücksichtigt wird. Dabei kann es vorkommen, daß ein numerisch berechneter Eigenwert mit „kleinem“ Imaginärteil durch Rundefehler einem exakten Eigenwert auf der anderen Seite der imaginären Achse entspricht, d.h. die $(+/-)$–Struktur des Spektrums einer Hamiltonischen Matrix (vgl. Folgerung 2.12) bleibt nicht unbedingt erhalten. Van Loan [V 1] erwähnt Beispiele, bei denen $n+1$ der mit dem unsymmetrischen QR–Algorithmus berechneten Eigenwerte einer Hamiltonischen Matrix in der offenen linken Halbebene liegen.
Da man bei der Lösung der ARG (1.1) an dem eindeutigen stabilen invarianten Teilraum der zugehörigen Hamiltonischen Matrix interessiert ist, muß man sich für die n „richtigen“ Eigenwerte entscheiden. Außerdem erhält man die Eigenwerte in einer nicht vorhersehbaren Reihenfolge, d.h. die Eigenwerte müssen zusätzlich noch geordnet werden.

Ein Algorithmus, der diese Nachteile beseitigt, wurde von Van Loan [V 1] eingeführt. Die Hamiltonische Struktur der Matrix wird ausgenutzt, indem symplektische Transformationsmatrizen benutzt werden und man erhält die Eigenwerte in der gewünschten Anordnung, d.h. man berechnet die n Eigenwerte mit positivem Realteil. Bezeichnet man diese mit $\lambda_1, \ldots, \lambda_n$, so sind die n stabilen Eigenwerte nach Folgerung 2.12 $-\lambda_1, \ldots, -\lambda_n$. Dieser Algorithmus basiert auf den folgenden Überlegungen:

Es sei $\mathcal{H}^2 = \left\{ N \in \mathbf{R}^{2n,2n} | N = H^2, H \in \mathcal{H}_{2n} \right\}$ die Menge der quadrierten Hamiltonischen Matrizen in $\mathbf{R}^{2n,2n}$. Aus $\sigma(H) = \{\lambda_1, -\lambda_1, \ldots, \lambda_n, -\lambda_n\}$ folgt sofort $\sigma(N) = \{\lambda_1^2, \lambda_1^2, \ldots, \lambda_n^2, \lambda_n^2\}$. Damit lassen sich die Eigenwerte von H berechnen, sobald man die Eigenwerte von H^2 kennt. Ist $N \in \mathcal{H}^2$ und $S \in \mathcal{S}_{2n}$, dann gilt $S^{-1}NS \in \mathcal{H}^2$, denn

$$N = H^2 \Longrightarrow S^{-1}NS = S^{-1}H^2S = S^{-1}HSS^{-1}HS = (S^{-1}HS)^2$$

und $S^{-1}HS \in \mathcal{H}_{2n}$ nach Lemma 2.22.
Orthogonale, symplektische Transformationen lassen also $\mathcal{H}^2$ invariant. Ist

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \in \mathcal{H}_{2n},$$

dann folgt

$$N = H^2 = \begin{bmatrix} A^2 + GF & AG - GA^T \\ FA - A^TF & FG + (A^T)^2 \end{bmatrix}. \tag{3.10}$$

Da $FA - A^T F$ schiefsymmetrisch ist, also $N_{n+i,i} = 0$ für $i = 1, \ldots, n$, kann dieser Block eliminiert werden, indem man PVLRED auf N anwendet. Diese Beobachtung bildet die Grundlage für den Algorithmus SQRED (von engl. 'SQare REDuced') von Van Loan.

Algorithmus 3.8 (SQRED)
Eingabe:

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \in \mathcal{H}_{2n}$$

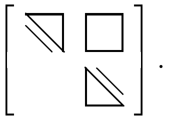
Ausgabe: *Die Eigenwerte* $\lambda_1, \ldots, \lambda_{2n}$ *von* H.

1.Schritt: *Bilde*

$$N = H^2 = \begin{bmatrix} A^2 + GF & AG - GA^T \\ FA - A^T F & FG + (A^T)^2 \end{bmatrix}.$$

2.Schritt: *Wende* PVLRED *auf* N *an, d.h. bestimme* $Q \in \mathcal{US}_{2n}$ *mit*

$$Q^T N Q = \begin{bmatrix} N_1 & N_2 \\ 0 & N_1^T \end{bmatrix} \triangleq$$



3.Schritt: *Bestimme die Eigenwerte* $\mu_1, \ldots, \mu_n$ *von* N_1 *mit dem unsymmetrischen QR–Algorithmus.*

4.Schritt: *Setze* $\lambda_i = \sqrt{\mu_i}$ *und* $\lambda_{i+n} = -\lambda_i$ *für* $i = 1, \ldots, n$.

Dabei ist es nicht nötig, die Matrix Q abzuspeichern, man erspart sich also die Aufdatierungen von Q in PVLRED. Damit benötigt der 2.Schritt $14n^3$ Flops. Der 1.Schritt kann mit $8n^3$ Flops durchgeführt werden, für den unsymmetrischen QR–Algorithmus ohne Akkumulation der Transformationen geben Golub/Van Loan [G 1] eine Aufwandsschätzung von $10n^3$ Flops an. Insgesamt kann SQRED daher mit einem Aufwand von $32n^3$ Flops ausgeführt werden, was ca. 40% des Aufwandes für den unsymmetrischen QR–Algorithmus, angewendet auf $H \in \mathcal{H}_{2n}$, entspricht (vgl. auch Kapitel 5.1.2).
Zur Speicherung benötigt man (zusätzlich zu H) ein n^2-Feld für $A^2 + GF$ und zwei Vektoren der Länge $n(n-1)/2$ für die schiefsymmetrischen Matrizen $FA - A^T F$ und $AG - GA^T$. Es besteht natürlich die Möglichkeit, den Speicherplatz von H zu überschreiben (vgl. auch implizite Version des Algorithmus in [V 1]), da hier H bzw. A, G, F aber unverändert weiter benötigt

werden, muß dieser Speicherplatz zusätzlich zur Verfügung gestellt werden. Außerdem wird für den in Kapitel 4 ausgeführten Algorithmus zur Lösung der ARG ohnehin H^2 benötigt, siehe dazu auch Kapitel 4.1 und Abschnitt 5.1.3.

Die Schritte 2 und 3 in Algorithmus 3.8 sind, falls sorgfältig implementiert, numerisch rückwärts stark stabil im Sinne von Definition 2.10. Probleme treten im ersten Schritt auf, da das Quadrieren einer Matrix zu großen Rundefehlern führen kann. Benutzt man eine auf inneren Produkten basierende Matrixmultiplikation auf einem Rechner mit der Maschinengenauigkeit $\mathbf{u}$, so gilt nach [G 1]

$$\|fl(H^2) - H^2\|_1 \leq 2n\mathbf{u}\|H\|_1^2 + O(\mathbf{u}^2),$$

der Fehler hängt also von der Norm von H quadratisch ab. Das führt bei Matrizen mit großer Norm offensichtlich zu Problemen. Nutzt man die Struktur von H aus, so läßt sich eine etwas bessere Fehlerabschätzung angeben, siehe dazu Abschnitt 5.1.1.

In [V 1] wird eine Fehlerschätzung für die mit SQRED berechneten Eigenwerte einer Hamiltonischen Matrix angegeben, die zum Teil auf heuristischen Überlegungen beruht. Dazu bezeichne mit $1/s(\lambda)$ die *Kondition* des Eigenwertes λ, wobei $s(\lambda) = y^T x$, falls y und x normalisierte Links– bzw. Rechtseigenvektoren zu λ sind. Falls λ ein Eigenwert von H ist, dann gilt für den entsprechenden mit SQRED berechneten Eigenwert λ_S

$$|\lambda_S - \lambda| \approx \min\left\{\frac{\mathbf{u}\|H\|_2^2}{s(\lambda)|\lambda|}, \frac{\sqrt{\mathbf{u}}\|H\|_2}{s(\lambda)}\right\}. \tag{3.11}$$

Für den Fehler eines mit dem unsymmetrischen QR–Algorithmus berechneten Eigenwertes λ_{QR} gilt nach Wilkinson [W 1]

$$|\lambda_{QR} - \lambda| \approx \frac{\mathbf{u}\|H\|_2}{s(\lambda)}. \tag{3.12}$$

Vergleicht man diese beiden Formeln, so kann man Folgendes schließen:

1. Ist $\|H\|/|\lambda| \approx 1$, dann ist λ_S im Wesentlichen so genau wie λ_{QR}.
2. Ist $\|H\|/|\lambda| > 1/\sqrt{\mathbf{u}}$, dann kann der Fehler in λ_S bis zu $1/\sqrt{\mathbf{u}}$ mal so groß sein wie der in λ_{QR}.

Man erkennt also, daß Probleme für Hamiltonische Matrizen mit Eigenwerten stark unterschiedlicher Größenordnungen auftreten können. In solchen Fällen ist der unsymmetrische QR–Algorithmus gegenüber SQRED zu bevorzugen.

3.4 Symmetrisches Aufdatieren der Lösung der algebraischen Riccatigleichung

Bei der Berechnung der Lösung der ARG (1.1) über den stabilen invarianten Teilraum für die zugehörige Hamiltonische Matrix

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix}$$

versucht man, eine Transformation der Form

$$\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}^{-1} \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} = \begin{bmatrix} \tilde{A} & \tilde{G} \\ 0 & \tilde{B} \end{bmatrix} \tag{3.13}$$

mit einem c–stabilen (1,1)–Block $\tilde{A}$ zu finden. Damit spannen nach Definition 2.7 die Spalten von $\begin{bmatrix} Q_{11} \\ Q_{21} \end{bmatrix}$ einen H–invarianten Teilraum zu den stabilen Eigenwerten von H auf und mit Satz 2.19 ist die gesuchte Lösung der ARG

$$X = -Q_{21}Q_{11}^{-1}. \tag{3.14}$$

Dabei ist bei jeder Ähnlichkeitstransformation von H mit einer Matrix S die Aufdatierung $Q = QS$ zu bilden und nachdem man die Form (3.13) gefunden hat, muß (3.14) berechnet werden (bei numerischen Verfahren wird man nicht die Inverse von Q_{11} bilden, sondern die lineare Matrixgleichung $XQ_{11} = -Q_{21}$ mit geeigneten Methoden wie QR– oder LR–Zerlegung lösen, siehe Kapitel 5.1.7). In (3.14) berechnet man die symmetrische Matrix X als Produkt zweier i.a. unsymmetrischer Matrizen. Die numerisch berechnete Matrix $\tilde{X}$ ist daher aufgrund von Rundefehlern i.a. nicht mehr symmetrisch, das berechnete Ergebnis hat also nicht die Struktur der exakten Lösung. Dies widerspricht aber der Forderung nach starker numerischer Rückwärtsstabilität aus Definition 2.10.

Eine Möglichkeit, die oben beschriebenen Schwierigkeiten zu vermeiden, besteht darin, die symmetrische Lösung X der ARG bei jeder Ähnlichkeitstransformation (symmetrisch) aufzudatieren, statt die Transformationsmatrix Q zu akkumulieren. Dazu geben Byers/Mehrmann [B 6] ein symmetrisches Aufdatierungsverfahren für symplektische Transformationen an. Da hier ein strukturerhaltender und stabiler Algorithmus für die Lösung der ARG konstruiert werden soll, sind die dabei zu benutzenden Transformationsmatrizen orthogonal und symplektisch. Daher werden hier nur Aufdatierungen mit Matrizen

$$Q = \begin{bmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{bmatrix} \in \mathcal{US}_{2n}$$

betrachtet.
Dazu seien $H_S(k,n,v)$ und $G_S(k,c,s)$ die in Kapitel 3.1 beschriebenen symplektischen Householder– und Givens–Matrizen. Nach Lemma 3.4 reicht es aus, Aufdatierungen mit diesen beiden Matrizentypen zu betrachten.
Statt nun bei jeder Ähnlichkeitstransformation $Q = QS$ zu bilden, reicht es aus, die Matrizen $Z = Q_1^{-1}, X = -Q_2Q_1^{-1}$ und $Y = Q_1^{-1}(-Q_2)$ aufzudatieren. Dabei beachte, daß X und Y symmetrisch sind und daher für die Speicherung von X, Y, Z nur ein Vektor der Länge n mehr als für die Speicherung von Q_1, Q_2 nötig ist.
Setze zu Beginn

$$Q = \begin{bmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}, \quad Z = I,\ X = 0,\ Y = 0.$$

Dann lassen sich die Aufdatierungen wie folgt durchführen.

Aufdatierung mit symplektischen Householder–Matrizen

Setze abkürzend $P := H(k,n,v)$, also

$$H_S(k,n,v) = \begin{bmatrix} P & 0 \\ 0 & P \end{bmatrix}.$$

Dann gilt mit

$$\tilde{Q} = QH_S(k,n,v) = \begin{bmatrix} Q_1P & -Q_2P \\ Q_2P & Q_1P \end{bmatrix}$$

und unter Ausnutzung von $P = P^{-1}$

$$\begin{aligned} (3.15) \qquad \tilde{X} &= -\tilde{Q}_2\tilde{Q}_1^{-1} = -Q_2P(Q_1P)^{-1} = -Q_2Q_1^{-1} = X \\ (3.16) \qquad \tilde{Z} &= \tilde{Q}_1^{-1} = PZ \\ (3.17) \qquad \tilde{Y} &= \tilde{Q}_1^{-1}(-\tilde{Q}_2) = (Q_1P)^{-1}(-Q_2P) = PYP. \end{aligned}$$

Aufdatierung mit symplektischen Givens–Matrizen

Sei dazu

$$G_S(k,c,s) = \begin{bmatrix} C & -S \\ S & C \end{bmatrix}, \quad C = I - (c-1)e_k e_k^T,\ S = s e_k e_k^T,\ c^2 + s^2 = 1.$$

Dann gilt

$$\tilde{Q} = Q G_S(k,c,s) = \begin{bmatrix} Q_1 C - Q_2 S & -Q_1 S - Q_2 C \\ Q_2 C + Q_1 S & -Q_2 S + Q_1 C \end{bmatrix}$$

und daher mit $\alpha = c + s y_{kk}$

$$\begin{aligned}
&(3.18) \qquad & \tilde{Z} &= Z - \frac{1}{\alpha}(c-1)e_k e_k^T Z - \frac{s}{\alpha} Y e_k e_k^T Z \\
&(3.19) \qquad & \tilde{X} &= X - \frac{s}{\alpha} Z^T e_k e_k^T Z \\
&(3.20) \qquad & \tilde{Y} &= Y + \frac{1}{\alpha}\{e_k e_k^T(-s + 2(c-1)y_{kk}) \\
& & &\quad + (1-c)(Y e_k e_k^T + e_k e_k^T Y) - s Y e_k e_k^T Y\}.
\end{aligned}$$

Zur Verifikation von (3.18)–(3.20) wird die *Sherman–Morrison–Woodbury–* Formel (SMW) für $A \in \mathbf{R}^{n,n},\ u, v \in \mathbf{R}^n$ benötigt (siehe z.B. [G 1])

$$(3.21) \qquad (A + uv^T)^{-1} = A^{-1} - \frac{A^{-1} u v^T A^{-1}}{1 + v^T A^{-1} u},$$

falls $v^T A^{-1} u \neq -1$ (sonst ist $A + uv^T$ singulär). Die SMW–Formel wird hier angewendet auf $A = Q_1$, $u = [(c-1)Q_1 - sQ_2]e_k$ und $v = e_k$. Der Term im Nenner ergibt mit diesen Setzungen

$$(3.22) \qquad \begin{aligned}
1 + v^T A^{-1} u &= 1 + (c-1)e_k^T Q_1^{-1} Q_1 e_k - s e_k^T Q_1^{-1} Q_2 e_k \\
&= 1 + (c-1) + s e_k^T Y e_k \\
&= c + s y_{kk} \ =: \alpha
\end{aligned}$$

und insgesamt

$$(3.23) \quad (A + uv^T)^{-1} = Q_1^{-1} - \frac{1}{\alpha}(c-1)e_k e_k^T Q_1^{-1} + \frac{s}{\alpha} Q_1^{-1} Q_2 e_k e_k^T Q_1^{-1}.$$

Damit ist (3.18) sofort klar, denn

$$\tilde{Z} = (Q_1C - Q_2S)^{-1} = (Q_1 + (c-1)Q_1e_ke_k^T - sQ_2e_ke_k^T)^{-1}.$$

Die Verifikation der Formeln (3.19) und (3.20) ist etwas aufwendiger, unter Benutzung von (3.23) jedoch ebenfalls relativ einfach durchzuführen.

Zu (3.19):

$$\begin{aligned}
\tilde{X} &= -(Q_2C + Q_1S)(Q_1C - Q_2S)^{-1} \\
&= -(Q_2 + (c-1)Q_2e_ke_k^T + sQ_1e_ke_k^T)(Q_1 + [(c-1)Q_1 - sQ_2]e_ke_k^T)^{-1} \\
&\overset{\text{SMW}}{=} -(Q_2 + (c-1)Q_2e_ke_k^T + sQ_1e_ke_k^T) \\
&\quad (Q_1^{-1} - \frac{1}{\alpha}(c-1)e_ke_k^TQ_1^{-1} + \frac{s}{\alpha}Q_1^{-1}Q_2e_ke_k^TQ_1^{-1}).
\end{aligned}$$

Multipliziert man diese Formel aus und ordnet die Terme um, so ergibt sich unter Ausnutzung von $-e_ke_k^TQ_1^{-1}Q_2e_ke_k^T = y_{kk}e_ke_k^T$

$$\begin{aligned}
\tilde{X} &= -Q_2Q_1^{-1} - \frac{s}{\alpha}Q_2Q_1^{-1}Q_2e_ke_k^TQ_1^{-1} - \frac{s}{\alpha}Q_1e_ke_k^TQ_1^{-1} \\
&\quad + \left\{\frac{1}{\alpha}(c-1) - (c-1) + \frac{1}{\alpha}(c-1)^2 + \frac{s}{\alpha}(c-1)y_{kk}\right\} Q_2e_ke_k^TQ_1^{-1} \\
&\quad + \left\{-s + \frac{s}{\alpha}c + \frac{s^2}{\alpha}y_{kk}\right\} Q_1e_ke_k^TQ_1^{-1}.
\end{aligned}$$

Wie man leicht nachrechnet, addieren sich die Terme in den geschweiften Klammern jeweils zu 0. Damit gilt

$$\tilde{X} = X - \frac{s}{\alpha}\left(Q_2Q_1^{-1}Q_2 + Q_1\right) e_ke_k^TQ_1^{-1}$$

Daraus ergibt sich (3.19), denn wegen $Q \in \mathcal{S}_{2n}$ gilt $Q_2Q_1^{-1} = (Q_2Q_1^{-1})^T$ und $I = Q_1^TQ_1 + Q_2^TQ_2$. Damit folgt

$$Q_1 = Q_1^{-T} - Q_2Q_1^{-1}Q_2. \quad \square$$

Zu (3.20):

$$\begin{aligned}
\tilde{Y} &= (Q_1C - Q_2S)^{-1}(-Q_1S - Q_2C) \\
&\overset{\text{SMW}}{=} (Q_1^{-1} - \frac{1}{\alpha}(c-1)e_ke_k^TQ_1^{-1} + \frac{s}{\alpha}Q_1^{-1}Q_2e_ke_k^TQ_1^{-1}) \\
&\quad (-sQ_1e_ke_k^T - Q_2 - (c-1)Q_2e_ke_k^T)
\end{aligned}$$

Multipliziert man die letzte Formel aus und nutzt wie bei der Verifikation von (3.19) die Beziehung $-e_k e_k^T Q_1^{-1} Q_2 e_k e_k^T = y_{kk} e_k e_k^T$, so folgt

$$\begin{aligned}
\tilde{Y} &= -Q_1^{-1}Q_2 + \left\{ \frac{s}{\alpha}(c-1) - s - \frac{(c-1)^2}{\alpha} y_{kk} - \frac{s}{\alpha}(c-1) y_{kk} \right\} e_k e_k^T \\
&\quad - \left\{ (c-1) + \frac{s^2}{\alpha} - \frac{s}{\alpha}(c-1) y_{kk} \right\} Q_1^{-1} Q_2 e_k e_k^T \\
&\quad + \frac{1}{\alpha}(c-1) e_k e_k^T Q_1^{-1} Q_2 - \frac{s}{\alpha} Q_1^{-1} Q_2 e_k e_k^T Q_1^{-1} Q_2 \\
&= Y + \frac{1}{\alpha}\{(-s + 2(c-1)y_{kk}) e_k e_k^T + (1-c)(Y e_k e_k^T + e_k e_k^T Y) \\
&\quad - \frac{1}{\alpha}(c-1) e_k e_k^T Y\}. \quad \square
\end{aligned}$$

Man sieht also, daß die Symmetrie der Matrix X bei dieser Form der Aufdatierung erhalten bleibt, denn jede Modifizierung von X besteht darin, eine symmetrische Matrix zu X zu addieren.
Das einzige Problem bei dieser Form der Aufdatierung entsteht, wenn $|\alpha|$ klein bzw. $\left|\frac{s}{\alpha}\right|$ groß wird, denn für $\left|\frac{s}{\alpha}\right| = O(10^k)$ verliert man beim Aufdatieren der Lösung X k signifikante Stellen.
Daher muß man bei der Implementierung obiger Formeln die Größe von $\left|\frac{s}{\alpha}\right|$ überwachen und gegebenenfalls anders vorgehen, falls diese Größe eine vorgegebene Toleranz überschreitet.
Bei den Beispielrechnungen mit dem Multishift–Algorithmus trat dieses Problem jedoch nur bei Beispiel 5.6 auf, bei dem allerdings auch die Kondition der Blöcke Q_1 und Q_2 groß ist, und damit die Lösung der linearen Matrixgleichung $XQ_1 = -Q_2$ schlecht konditioniert ist. Dabei war zu beobachten, daß der größte auftretende Wert $\left|\frac{s}{\alpha}\right|$ von der gleichen Größenordnung wie die Kondition von Q_1 ist. Benutzt man trotz dieses Problemes obiges Aufdatierungsverfahren, so erhält man u.U. vier signifikante Stellen mehr als mit der gewöhnlichen Vorgehensweise, vgl. dazu Kapitel 5.3.

Der Aufwand für die Aufdatierung mit einer symplektischen Householder–Matrix $H(k,n,v)$ ergibt sich nach Abschnitt 3.1.1 zu $4n^2 - 4nk$ Flops für (3.16). Eine Ähnlichkeitstransformation einer symmetrischen Matrix mit der gleichen Householdermatrix kann effektiv mit dem von Wilkinson [W 1] beschriebenen Verfahren durchgeführt werden (vgl. auch Fortran–Subroutine

SYMREF, [B 3]). Dann benötigt (3.17) $4n^2 - 4nk$ Flops, insgesamt also $8n^2 - 8nk$ Flops für den Formelsatz (3.15) – (3.17).
Unter Ausnutzung der Symmetrien benötigt eine Aufdatierung mit einer symplektischen Givensmatrix $G_S(k, c, s)$ für

- (3.18) $3n^2 + 2n$ Flops,
- (3.19) $(3n^2 + 3n)/2$ Flops,
- (3.20) $(3n^2 + 7n)/2$ Flops.

Zählt man die wesentlichen Faktoren, so ergibt sich für (3.18)–(3.20) ein Aufwand von $6n^2 + O(n)$ Flops.
Für die entsprechenden Aufdatierungen von Q_1 und Q_2 mit Householder–Matrizen benötigt man nach Abschnitt 3.1.1 $8n^2 - 8nk$ Flops und mit Givens–Matrizen nach Abschnitt 3.1.2 $12n$ Flops. Vergleicht man z.B. den Aufwand, der bei der einmaligen Durchführung des Algorithmus 3.6 für die Aufdatierungen von Q_1, Q_2 oder aber X, Y, Z nötig ist, d.h. zwei Aufdatierungen mit Householder–Matrizen $H(k, n, v)$ und eine mit einer Givens–Matrix $G(k, n + k, c, s)$, so sieht man, daß das hier beschriebene Aufdatierungsverfahren im Wesentlichen $6n^2$ Flops mehr benötigt, die auf das aufwendigere Aufdatieren der Givens–Rotation zurückgehen.
Der bei einer sukzessiven Durchführung von solchen Aufdatierungen entstehende Mehraufwand wird um den für die Lösung von $XQ_1 = -Q_2$ benötigten Aufwand vermindert. Verwendet man dazu die LR-Zerlegung mit Vorwärts– und Rückwärtseinsetzen aus Standard–Subroutinen wie z.B. die LINPACK–Routinen DGEFA und DGESL [D 2], so werden dafür $8n^3/3 - n^2/2 - n$ Flops benötigt.
I.a. wird eine der hier beschriebenen Aufdatierungen mit einer Ähnlichkeitstransformation einer Hamiltonischen Matrix mit der entsprechenden orthogonal symplektischen Matrix einhergehen. Der Aufwand dafür überwiegt den für die Aufdatierungen von Q_1, Q_2 bzw. X, Y, Z, daher fällt der hier entstehende Mehraufwand nicht allzu stark ins Gewicht.

Kapitel 4

Ein orthogonal symplektischer Multishift–Algorithmus für die algebraische Riccatigleichung

4.1 Orthogonal symplektische Berechnung invarianter Teilräume Hamiltonischer Matrizen

Am Ende von Kapitel 2.3 wurde erwähnt, wo die Schwierigkeit liegt, die Hamiltonische Hessenbergform einer Matrix $H \in \mathcal{H}_{2n}$ zu berechnen. Die erste Spalte einer dafür in Frage kommenden Matrix mußte das nichtlineare Gleichungssystem

$$\begin{array}{rcl} x^T J H^{2i-1} x &=& 0, \quad i = 1, \ldots, n-1, \\ x^T x &=& 1 \end{array} \tag{4.1}$$

erfüllen. Zudem durfte dieser Vektor nicht in einem H–invarianten Teilraum $\mathcal{Q}$ mit $\dim \mathcal{Q} \leq n$ enthalten sein.

Andererseits erfüllt nach Lemma 2.29 jeder in einem isotropen H–invarianten Teilraum enthaltene Vektor mit Euklidischer Norm 1 (4.1). Es sei nun v_1 solch ein Vektor. Dabei sei $\mathcal{Q}$ der minimale isotrope H–invariante Teilraum, in dem v_1 enthalten ist, d.h. v_1 habe eine Komponente in jede Richtung dieses

Teilraums. Ist nun $V = [v_1, \ldots, v_{2n}] \in \mathcal{US}_{2n}$ und $\tilde{H} = \left[\tilde{h}_{i,j}\right]_{i,j=1}^{2n} = V^T H V$, dann gilt

$$\tilde{h}_{n+1,1} = 0, \tag{4.2}$$

denn wegen Lemma 3.1 ist $v_{n+1} = J^T v_1$ und damit

$$\tilde{h}_{n+1,1} = v_{n+1}^T H v_1 = v_1^T J H v_1.$$

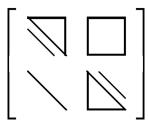
Da v_1 (4.1) erfüllt, folgt (4.2).
Wendet man die Paige/Van Loan–Reduktion (Algorithmus 3.7) auf $\tilde{H}$ an, so bleibt die Null in $\tilde{h}_{n+1,1}$ erhalten. Dazu betrachte die Matrix $U \in \mathcal{US}_{2n}$, die $\tilde{H}$ auf PVL–Form transformiert. U besitzt folgende Gestalt,

$$U = [u_1, \ldots, u_{2n}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & U_1 & 0 & -U_2 \\ 0 & 0 & 1 & 0 \\ 0 & U_2 & 0 & U_1 \end{bmatrix}.$$

Damit gilt

$$u_{n+1}^T \tilde{H} u_1 = e_{n+1}^T \tilde{H} e_1 = \tilde{h}_{n+1,1} = 0. \tag{4.3}$$

Sei nun $Q = VU$. Dann gilt $q_1 = v_1$ und

(4.4) $\hat{H} = Q^T H Q =$ 

ist in PVL–Form mit $\hat{h}_{n+1,1} = 0$.

Lemma 4.1 *Es seien H, $\hat{H}$, Q und $\mathcal{Q}$ wie oben mit* $\dim \mathcal{Q} = k$. *Dann gilt*

$$\begin{aligned} \hat{h}_{n+j,j} &= 0, \quad j = 1, \ldots, k, \\ \hat{h}_{j+1,j} &\neq 0, \quad j = 1, \ldots, k-1 \end{aligned}$$

und $\hat{h}_{k+1,k} = 0$.

Beweis: Benutze das Prinzip der vollständigen Induktion. Wegen (4.3) gilt $\hat{h}_{n+1,1} = 0$. Daher folgt aus (4.4)

$$H q_1 = \hat{h}_{11} q_1 + \hat{h}_{21} q_2.$$

Falls $\hat{h}_{21} = 0$, so würde q_1 einen 1–dimensionalen H–invarianten Teilraum aufspannen, d.h. es wäre $k = 1$ und alles gezeigt. Ist $\hat{h}_{21} \neq 0$, so folgt $q_2 \in \mathcal{Q}$ und $\dim \mathcal{Q} > 1$.
Sei nun $1 < l \leq k$ und es gelte $\hat{h}_{n+j-1,j-1} = 0$, $\hat{h}_{j,j-1} \neq 0$ und $q_j \in \mathcal{Q}$ für $j = 1, \ldots, l$. Da $\mathcal{Q}$ isotrop ist, folgt $q_i^T J q_j = 0$ für $1 \leq i, j \leq l$. Wegen $Q \in \mathcal{US}_{2n}$ ist $q_{n+j} = J^T q_j$ und mit (4.4) gilt

$$\hat{h}_{n+j,j} = q_{n+j}^T H q_j = q_j^T J \sum_{i=1}^{j+1} \hat{h}_{ij} q_i = \hat{h}_{j+1,j} q_j^T J q_{j+1}. \tag{4.5}$$

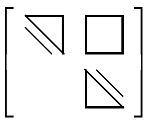
Ist jetzt $\hat{h}_{j+1,j} \neq 0$, dann gilt wegen (4.4)

$$q_{j+1} = \frac{1}{\hat{h}_{j+1,j}} \left[(H - \hat{h}_{j,j} I_{2n}) q_j - \sum_{i=1}^{j-1} \hat{h}_{ij} q_i \right] \tag{4.6}$$

und daher $q_{j+1} \in \mathcal{Q}$, also mit (4.5) $\hat{h}_{n+j,j} = 0$ und $\dim \mathcal{Q} = k > j$.
Ist andererseits $\hat{h}_{j+1,j} = 0$, so folgt aus (4.5) sofort $\hat{h}_{n+j,j} = 0$. Dann ist aber $\operatorname{span}\{q_1, \ldots, q_j\}$ der isotrope H–invariante Teilraum $\mathcal{Q}$ mit $\dim \mathcal{Q} = k = j$.
Gilt andererseits $k = j$, so muß $\hat{h}_{k+1,k} = 0$ gelten, da sonst wegen (4.6) $q_{k+1} \in \mathcal{Q}$ wäre und damit $\dim \mathcal{Q} > k$ sein müßte, was der Voraussetzung widerspricht. □

Folgerung 4.2 *Gilt* $\dim \mathcal{Q} = n$, *dann folgt mit obigen Bezeichnungen*

$$\hat{H} = Q^T H Q = \begin{bmatrix} \hat{H}_{11} & \hat{H}_{12} \\ 0 & -\hat{H}_{11}^T \end{bmatrix} \triangleq$$



,

wobei $\hat{H}_{11}$ *eine unreduzierte Hessenbergmatrix ist.*

Bemerkung 1: Unter der Annahme, daß q_1 die erste Spalte einer Matrix ist, die H auf unreduzierte Hamiltonische Hessenbergform transformiert, muß wegen Satz 2.29 $\dim \mathcal{Q} > n$ gelten, d.h. $\mathcal{Q}$ kann nicht isotrop sein. Setzt man (für solch ein q_1) $q_{n+j} := J^T q_j$ für $j = 1, \ldots, n$, so liefert die Vorgehensweise im Beweis zu Lemma 4.1 einen *Arnoldiprozess* zur Berechnung der Hamiltonischen Hessenbergform, siehe dazu [A 1].

Bemerkung 2: Raines/Watkins geben in [R 1] ein analoges Ergebnis an, wobei auch nichtorthogonale symplektische Transformationen zugelassen sind.

Um einen bestimmten isotropen H–invarianten Teilraum zu berechnen, benötigt man also einen darin enthaltenen Vektor x. Dieser Teilraum wird dann aufgespannt durch die ersten k Spalten von Q aus (4.4). Um solch einen Vektor für den Fall $k = n$ zu bestimmen, benutzen Ammar/Mehrmann [A 1] eine Verallgemeinerung der impliziten Doppelshift–Strategie für den QR–Algorithmus nach Francis, siehe z.B. Golub/Van Loan [G 1].
Dazu seien $\lambda_1, \ldots, \lambda_k \in \sigma(H)$ die zu einem reellen H–invarianten Teilraum $\mathcal{Q}$ gehörenden Eigenwerte. Da $\mathcal{Q}$ reell ist, muß zu jedem λ_i mit $Im(\lambda_i) \neq 0$ ein λ_j mit $\lambda_i = \bar{\lambda}_j$, $1 \leq i, j \leq k$, existieren. Damit ist

$$M_k := (H - \lambda_1 I) \cdots (H - \lambda_k I) \tag{4.7}$$

eine reelle $2n \times 2n$–Matrix. Unter der *Multishift*–Technik versteht man nun einen k-fachen QR–Schritt. Sei

$$\begin{array}{rcll} H - \lambda_1 I & = & U_1 R_1 & \text{(QR–Zerlegung)} \\ H_1 & = & R_1 U_1 + \lambda_1 I & \\ H_1 - \lambda_2 I & = & U_2 R_2 & \text{(QR–Zerlegung)} \\ H_2 & = & R_2 U_2 + \lambda_2 I & \\ & \vdots & & \\ H_{k-1} - \lambda_k I & = & U_k R_k & \text{(QR–Zerlegung)} \\ H_k & = & R_k U_k + \lambda_k I, & \end{array}$$

dann gilt mit $Q_k := U_1 \cdots U_k$

$$H_k = Q_k^T H Q_k \tag{4.8}$$

$$M_k = Q_k R_k \cdots R_1. \tag{4.9}$$

Dies kann durch vollständige Induktion gezeigt werden. Dabei ist die Induktionsverankerung ein QR–Schritt mit einfachem Shift, siehe [G 1], S.374/75. Es gelte die Behauptung also für alle j, $1 \leq j \leq k-1$. Dann ist

$$\begin{aligned} Q_k^T H Q_k &= U_k^T Q_{k-1}^T H Q_{k-1} U_k = U_k^T H_{k-1} U_k \\ &= U_k^T (U_k R_k + \lambda_k I) U_k = R_k U_k + \lambda_k I \\ &= H_k, \end{aligned}$$

also (4.8), und

$$
\begin{aligned}
Q_k R_k \cdots R_1 &= Q_{k-1}(H_{k-1} - \lambda_k I) R_{k-1} \cdots R_1 \\
&\overset{(4.8)}{=} H Q_{k-1} R_{k-1} \cdots R_1 - \lambda_k Q_{k-1} R_{k-1} \cdots R_1 \\
&= H M_{k-1} - \lambda_k M_{k-1} = M_k. \quad \square
\end{aligned}
$$

Weiter gilt mit obigen Bezeichnungen

$$
H_k = \begin{bmatrix} H_{11}^{(k)} & H_{12}^{(k)} \\ 0 & H_{22}^{(k)} \end{bmatrix} \begin{matrix} \} \, 2n-k \\ \} \, k \end{matrix} \tag{4.10}
$$

mit $\sigma(H_k) = \sigma(H_{11}^{(k)}) \cup \sigma(H_{22}^{(k)})$ und $\sigma(H_{11}^{(k)}) = \{\lambda_{k+1}, \ldots, \lambda_{2n}\}$, sowie $\sigma(H_{22}^{(k)}) = \{\lambda_1, \ldots, \lambda_k\}$.
Dies ergibt sich wieder durch eine vollständige Induktion. Für $k = 1$ gilt $\mathrm{rg}(H - \lambda_1 I) \leq 2n - 1$ und deshalb

$$
U_1 R_1 \triangleq U_1 \begin{bmatrix} * & \cdots & & * \\ 0 & \ddots & & \vdots \\ \vdots & & * & * \\ 0 & \cdots & 0 & 0 \end{bmatrix}.
$$

Daraus folgt (4.10) für $k = 1$ wegen

$$
H_1 = R_1 U_1 + \lambda_1 I = \begin{bmatrix} H_{11}^{(1)} & H_{12}^{(1)} \\ 0 & \lambda_1 \end{bmatrix}.
$$

Es gelte nun (4.10) für alle $1 \leq j \leq k-1$ und sei $\sigma(H_{22}^{(k-1)}) = \{\lambda_1, \ldots, \lambda_{k-1}\}$.
Dann ist

$$
H_{k-1} - \lambda_k I = \begin{bmatrix} H_{11}^{(k-1)} - \lambda_k I & H_{12}^{(k)} \\ 0 & H_{22}^{(k-1)} - \lambda_k I \end{bmatrix} \begin{matrix} \} \, 2n-k+1 \\ \} \, k-1 \end{matrix}
$$

und wegen $\lambda_k \in \sigma(H_{11}^{(k-1)})$ gilt $\mathrm{rg}(H_{11}^{(k-1)} - \lambda_k I) \leq 2n - k$.
Berechnet man die QR–Zerlegungen

$$
\begin{aligned}
H_{11}^{(k-1)} - \lambda_k I &= U_{11}^{(k)} R_{11}^{(k)}, \\
H_{22}^{(k-1)} - \lambda_k I &= U_{22}^{(k)} R_{22}^{(k)},
\end{aligned}
$$

so gilt

$$R_{11}^{(k)} = \begin{bmatrix} \tilde{R}_{11}^{(k)} \\ 0 \dots 0 \end{bmatrix}$$

und damit

$$\begin{aligned} H_k &= \begin{bmatrix} R_{11}^{(k)} & R_{12}^{(k)} \\ 0 & R_{22}^{(k)} \end{bmatrix} \begin{bmatrix} U_{11}^{(k)} & 0 \\ 0 & U_{22}^{(k)} \end{bmatrix} + \lambda_k I \\ &= \left[\begin{array}{c|c|c} & * & \\ H_{11}^{(k)} & \vdots & R_{12}^{(k)} U_{22}^{(k)} \\ & * & \\ \hline 0 \dots 0 & \lambda_k & * \dots * \\ \hline 0 & 0 & \tilde{H}_{22}^{(k)} \end{array}\right] \begin{array}{l} \left.\vphantom{\begin{array}{c} * \\ \vdots \\ * \end{array}}\right\} 2n-k \\ \\ \} \; k-1 \end{array} . \end{aligned}$$

Setzt man

$$H_{22}^{(k)} = \begin{bmatrix} \lambda_k & * \dots * \\ 0 & \tilde{H}_{22}^{(k)} \end{bmatrix},$$

so gilt wegen $\tilde{H}_{22}^{(k)} = U_{22}^{(k)^T} H_{22}^{(k-1)} U_{22}^{(k)}$

$$\sigma(H_{22}^{(k)}) = \sigma(H_{22}^{(k-1)}) \cup \{\lambda_k\} = \{\lambda_1, \dots, \lambda_k\},$$

und damit folgt (4.10). □

Für $k = n$ spannen wegen (4.10) die ersten n Spalten von Q_n aus (4.8), (4.9) einen n–dimensionalen H–invarianten Teilraum zu $\lambda_{n+1}, \dots, \lambda_{2n}$ auf. Sind dies die ersten n Spalten einer symplektischen Matrix, so handelt es sich dabei um einen Lagrange–Teilraum. Soll dieser Lagrange–Teilraum berechnet werden, so unterliegen die als Shifts gewählten Eigenwerte einigen Einschränkungen, die sich aus dem folgendem Lemma ergeben.

Lemma 4.3 *Es sei $H \in \mathcal{H}_{2n}$ und es werde ein n–dimensionaler H–invarianter Teilraum $\mathcal{Q}$ zu den Eigenwerten $\lambda_1, \dots, \lambda_n$ von den Spalten von $\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}$ aufgespannt. $\mathcal{Q}$ ist genau dann ein Lagrange–Teilraum, falls*

$$\{\lambda_1, \dots, \lambda_n\} \cap \{-\lambda_1, \dots, -\lambda_n\} = \emptyset. \tag{4.11}$$

Beweis: Nach Voraussetzung gilt

$$H \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} T$$

mit $\sigma(T) = \{\lambda_1, \ldots, \lambda_n\}$. Dies ist äquivalent zu

$$J \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} T = JH \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = H^T J^T \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}.$$

Transponiert man diese Gleichung, so folgt

$$-T^T [\, -Q_2^T \; Q_1^T \,] = [\, -Q_2^T \; Q_1^T \,] H$$

und damit

$$[\, -Q_2^T \; Q_1^T \,] \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} T - (-T^T) [\, -Q_2^T \; Q_1^T \,] \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = 0.$$

Mit dem Satz über die homogene Lyapunovsche Matrixgleichung (siehe z.B. [K 2]) gilt

$$[\, -Q_2^T \; Q_1^T \,] \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = 0$$

genau dann, wenn $\sigma(T)$ und $\sigma(-T^T)$ disjunkt sind. Dies ist aber gerade äquivalent zu (4.11). □

Die als Shifts gewählten Eigenwerte $\lambda_1, \ldots, \lambda_n$ müssen demnach folgende Eigenschaften erfüllen.

- Repräsentieren $\lambda_{i_1}, \ldots, \lambda_{i_j}$ einen Eigenwert von H der algebraischen Vielfachheit j, so müssen entweder alle oder keiner von ihnen als Shifts benutzt werden.

- Wird $\lambda \in \sigma(H)$ als Shift benutzt, so darf der nach Folgerung 2.12 ebenfalls zum Spektrum von H gehörende Eigenwert $-\lambda$ nicht auch als Shift benutzt werden.

Insbesondere sind die beiden obigen Eigenschaften erfüllt, falls H genau n stabile Eigenwerte besitzt und man als Shifts die instabilen Eigenwerte wählt.

Somit erhält man den zur Bestimmung der stabilisierenden Lösung der ARG benötigten Lagrange–Teilraum.
Allgemein kann mit dem oben beschriebenen n–fachen Shift ein beliebiger Lagrange–Teilraum berechnet werden. Dies erfordert jedoch i.a. einen Aufwand von $O(n^4)$ und ist damit nicht mehr effizient. Wie beim QR–Algorithmus mit implizitem Doppelshift berechnet man daher nur die erste Spalte der „Shift–Matrix“, und nutzt dann Lemma 4.1 aus.
Die Grundlage dieser Vorgehensweise für die Multishifttechnik formulierten Ammar/Mehrmann in folgendem Satz.

Satz 4.4 *Sei $H \in \mathcal{H}_{2n}$ mit Eigenwerten $\lambda_1, \ldots, \lambda_{2n}$. Falls ein n–dimensionaler H–invarianter Lagrange–Teilraum $\tilde{\mathcal{Q}}$ zu den Eigenwerten $\lambda_1, \ldots, \lambda_n$ existiert, dann ist der Vektor*

$$x = (H - \lambda_1 I) \cdots (H - \lambda_n I) e_1$$

im H–invarianten Lagrange–Teilraum $\mathcal{Q}$ zu den Eigenwerten $\lambda_{n+1}, \ldots, \lambda_{2n}$ enthalten.

Beweis: Da der Lagrange–Teilraum zu $\lambda_1, \ldots, \lambda_n$ reell vorausgesetzt wurde, ist mit derselben Argumentation wie bei (4.7)

$$M_n := (H - \lambda_1 I) \cdots (H - \lambda_n I)$$

reell. Bildet man die QR–Zerlegung $M_n = QR$ wie in (4.9), so ist $Qe_1 = q_1$ wegen (4.9), (4.10) im H–invarianten Lagrange–Teilraum zu den Eigenwerten $\lambda_{n+1}, \ldots, \lambda_{2n}$ enthalten. Wegen

$$M_n e_1 = QRe_1 = r_{11} q_1$$

ist dann auch $x \in \mathcal{Q} = J\tilde{\mathcal{Q}}$. $\square$

Mit Hilfe von Satz 4.4 und Lemma 4.1 kann ein Algorithmus zur Berechnung eines H–invarianten Lagrange–Teilraumes $\mathcal{Q}$ wie folgt formuliert werden.

Algorithmus 4.5 (LISH)
Eingabe: *$H \in \mathcal{H}_{2n}$, so daß ein H–invarianter Lagrange–Teilraum $\mathcal{Q}$ existiert und die zu $\tilde{\mathcal{Q}} = J\mathcal{Q}$ gehörenden Eigenwerte $\lambda_1, \ldots, \lambda_n$, sowie eine Toleranz ε.*

Ausgabe:

$$H \stackrel{\triangle}{=} \begin{bmatrix} \diagdown & \square \\ & \diagdown \end{bmatrix} \in \mathcal{H}_{2n}, \qquad Q = \begin{bmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{bmatrix} \in \mathcal{US}_{2n},$$

wobei die Spalten von $\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \in \mathbf{R}^{2n,n}$ *den Lagrange–Teilraum* $\mathcal{Q}$ *zu den Eigenwerten* $\lambda_{n+1}, \ldots, \lambda_{2n}$ *aufspannen.*

Setze $Q := I_{2n}$.

1.Schritt:

Berechne den **Multishiftvektor**

$$x = \alpha(H - \lambda_1 I) \cdots (H - \lambda_n I) e_1. \tag{4.12}$$

Bestimme mit Algorithmus 3.5 COLRED $Q_1 \in \mathcal{US}_{2n}$ *so, daß* $Q_1^T x = \pm\|x\|_2 e_1$. *Setze*

$$H := Q_1^T H Q_1, \qquad Q := Q Q_1. \tag{4.13}$$

2.Schritt:

Berechne mit Hilfe von Algorithmus 3.7 PVLRED $Q_2 \in \mathcal{US}_{2n}$ *mit*

$$Q_2^T H Q_2 \stackrel{\triangle}{=} \begin{bmatrix} \diagdown & \square \\ \diagdown & \diagdown \end{bmatrix}. \tag{4.14}$$

Setze

$$H := Q_2^T H Q_2, \qquad Q := Q Q_2. \tag{4.15}$$

3.Schritt: (**Deflation**)

```
p = 0
WHILE p < n
    FOR i = 1,...,n
        IF |h_{n+i,i}| < ε THEN h_{n+i,i} := 0.
    END FOR
    FOR i = 2,...,n
        IF |h_{i,i-1}| < ε THEN h_{i,i-1} := 0.
    END FOR
```

Bestimme

$$p := \max_{p+1\le k\le n}\{k \mid h_{k+1,k} = 0 \text{ und } h_{n+i,i} = 0 \text{ für } i = 1,\dots,k\}.$$

Partitioniere H entsprechend

$$H = \begin{bmatrix} A_{11} & A_{12} & G_{11} & G_{12} \\ 0 & A_{22} & G_{21} & G_{22} \\ 0 & 0 & -A_{11}^T & 0 \\ 0 & F_{22} & -A_{12}^T & -A_{22}^T \end{bmatrix} \begin{array}{l} \}\,p \\ \}\,n-p \\ \}\,p \\ \}\,n-p \end{array} \tag{4.16}$$

IF $p < n$

Setze

$$H_{22} := \begin{bmatrix} A_{22} & G_{22} \\ F_{22} & -A_{22}^T \end{bmatrix}, \tag{4.17}$$

$$x_2 := \alpha_2(H_{22} - \lambda_1 I)\cdots(H_{22} - \lambda_n I)e_1. \tag{4.18}$$

Bestimme mit Algorithmus 3.5 COLRED *eine Transformationsmatrix $P_1 \in \mathcal{US}_{2(n-p)}$ mit $P_1^T x_2 = \pm\|x_2\|_2 e_1$. Setze*

$$H_{22} := P_1^T H_{22} P_1$$

und bestimme wie im 2.Schritt mit Algorithmus 3.7 PVLRED *eine Transformationsmatrix $P_2 \in \mathcal{US}_{2(n-p)}$, die H_{22} auf PVL–Form reduziert. Setze*

$$P := \begin{bmatrix} U & -V \\ V & U \end{bmatrix} := P_1 P_2, \tag{4.19}$$

$$Q_p := \begin{bmatrix} I_p & 0 & 0 & 0 \\ 0 & U & 0 & -V \\ 0 & 0 & I_p & 0 \\ 0 & V & 0 & U \end{bmatrix}, \tag{4.20}$$

$$H := Q_p^T H Q_p, \qquad Q := QQ_p. \tag{4.21}$$

END IF

END WHILE

END.

Da im ganzen Algorithmus nur orthogonal symplektische Transformationen verwendet werden, kann die Hamiltonische Struktur von H erhalten werden. Wegen der weiter unten erläuterten numerischen Schwierigkeiten (siehe auch Kapitel 4.2) wurde LISH in obiger Form nicht implementiert. Mit Hilfe der FORTRAN Routine OSMSTP (Kapitel 5.1.6) können die Schritte (4.12) – (4.15) bzw. (4.18) – (4.21) durchgeführt werden, wobei durch die verwendete Speichertechnik H immer exakt Hamiltonisch bleibt.
Aufwandsschätzungen sind schwierig anzugeben, da der Aufwand von der Anzahl der Deflationen abhängt, siehe dazu Kapitel 5.1.6.
Die erzielte Genauigkeit hängt zum einen von der Genauigkeit der berechneten Eigenwerte ab, zum anderen vom im ersten Schritt berechneten Multishiftvektor. Durch diesen Vektor wird die erste Spalte der Transformationsmatrix festgelegt und damit im Wesentlichen die ganze zu berechnende Matrix $Q \in \mathcal{US}_{2n}$, durch deren ersten n Spalten wiederum der gesuchte Lagrange–Teilraum gegeben ist. Da die vorgegebenen Eigenwerte i.a. nicht exakt sind und (4.12) natürlich auch Rundefehlern unterliegt, führt dies dazu, daß der berechnete Vektor x nicht mehr genau im gesuchten Teilraum liegt. Daher hängt der Abstand des berechneten Lagrange–Teilraums vom gesuchten Teilraum wesentlich von der Genauigkeit des berechneten Vektors x ab. Dazu sollte in (4.12) der Skalierungsfaktor α so gewählt werden, daß die Rundefehler möglichst gering gehalten werden. Zur Berechnung des Multishiftvektors wurde die FORTRAN–Routine MSHIFT implementiert. MSHIFT und die Wahl des Skalierungsfaktors werden in Abschnitt 5.1.3 besprochen.

Für die durch x festgelegte erste Spalte von Q kann Lemma 4.1 angewendet werden. Man sieht, daß in exakter Arithmetik Deflation gerade für $p = k$ mit k wie in Lemma 4.1 eintritt, d.h. falls $k = n$, dann endet LISH wegen Folgerung 4.2 nach dem zweiten Schritt mit $p = n$.
Im Falle der Deflation wird also ein p–dimensionaler H–invarianter Teilraum $\mathcal{Q}_p$ abgespalten und im dritten Schritt der verbleibende $(n-p)$–dimensionale Teilraum $\mathcal{Q} \setminus \mathcal{Q}_p$ berechnet. Falls $\sigma(A_{22}) = \{-\lambda_{p+1}, \ldots, -\lambda_n\}$ gilt, so gehört dieser zu den Eigenwerten $\lambda_{p+1}, \ldots, \lambda_n$, so daß man (4.18) durch

$$x_2 = \alpha_2 (H_{22} - \lambda_{p+1} I) \cdots (H_{22} - \lambda_n I) e_1 \tag{4.22}$$

ersetzen könnte. Da man i.a. aber nicht weiß, welche Eigenwerte zu A_{11} gehören, müßten zunächst die Eigenwerte von H_{22} berechnet werden. Da sich durch Rundefehler diese Eigenwerte von den in LISH eingegebenen unterscheiden, könnte dies unter Umständen ratsam sein, um einen möglichst genauen invarianten Teilraum der berechneten Untermatrix H_{22} zu bestimmen. Der dadurch entstehende Mehraufwand wird dadurch vermindert, daß statt (4.18) nur noch (4.22) berechnet werden muß. Bei den durchgeführten numerischen Experimenten ergab sich jedoch keine Verbesserung bei Neuberechnung dieser Eigenwerte.

Bemerkung 1: Mit Hilfe ähnlicher Betrachtungen, wie die auf Algorithmus 4.5 führenden, geben Raines/Watkins [R 1] eine Klasse von Algorithmen an, die auf symplektischen, aber nicht notwendig orthogonalen Transformationen beruhen, und für die LISH einen Spezialfall darstellt.

Bemerkung 2: Die Benutzung von (nahezu) exakten Shifts im QR–Algorithmus kann zu einer vorzeitigen Abspaltung führen, die vom üblichen QR–Algorithmus nicht erkannt wird. Dieses Phänomen beschreibt Parlett in [P 2] und bezeichnet es dort als „Vorwärtsinstabilität des QR–Algorithmus“.
Da in LISH mit nahezu exakten Shifts gearbeitet wird, besteht die Möglichkeit, daß dieses Phänomen auch hier Probleme bereitet. Bei den Beispielrechnungen konnte dies zwar nicht beobachtet werden, es ist jedoch nicht auszuschließen, das ein ähnliches Problem auftreten kann.

4.2 Anwendung auf die algebraische Riccatigleichung

Nach Satz 2.19 kennt man die stabilisierende Lösung der ARG, sobald man den H–invarianten Teilraum zu den n stabilen Eigenwerten von H bestimmt hat. Da dies ein Lagrange–Teilraum ist, kann man ihn mit Algorithmus 4.5 LISH berechnen, indem man als Shifts gerade die n instabilen Eigenwerte benutzt.
Die Eigenwerte von H kann man nun mit Hilfe von Algorithmus 3.8 SQRED berechnen. Damit hat man ein orthogonal symplektisches Verfahren zur Berchnung der Eigenwerte zur Verfügung, das erheblich schneller als der gewöhnliche unsymmetrische QR–Algorithmus ist, jedoch den in Kapitel 3.3 erwähnten Genauigkeitsbeschränkungen unterliegt. Eine andere, schnelle Möglichkeit bildet der Hamiltonische SR–Algorithmus (siehe Kapitel 2.4).
Unabhängig von der verwendeten Methode erhält man aber nur Näherungen für die Eigenwerte. Zudem werden, wie im vorigen Abschnitt bereits erwähnt,

bei der Berechnung des Multishiftvektors (möglicherweise erhebliche) Rundefehler gemacht. Damit sind die in Lemma 4.1 vorhergesagten Eigenschaften der Subdiagonalelemente im (1,1)–Block und der Diagonalelemente im (2,1)–Block auch im Rahmen der bei Transformationen mit orthogonalen Matrizen zu erwartenden Rechengenauigkeit nicht immer erfüllt.
Sei nun $Q = \begin{bmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{bmatrix} \in \mathcal{US}_{2n}$ die von LISH berechnete Transformationsmatrix und

$$Q^T H Q = \hat{H} := \begin{bmatrix} \hat{A} & \hat{G} \\ \hat{F} & -\hat{A}^T \end{bmatrix}. \tag{4.23}$$

In exakter Arithmetik wäre nach Durchführung von LISH $\hat{F} = 0$, was wegen obiger Betrachtungen i.a. nicht der Fall ist. Die durch (4.2) gegebene Näherungslösung ist nach Satz 2.19

$$\hat{X} = -Q_2 Q_1^{-1}. \tag{4.24}$$

Berechnet man das Residuum, das beim Einsetzen von $\hat{X}$ in die ARG entsteht, so erhält man unter der Annahme, daß $\hat{X} = \hat{X}^T$ gilt,

$$\begin{aligned} \mathrm{Res}(\hat{X}) &= F + A^T\hat{X} + \hat{X}A - \hat{X}G\hat{X} \\ &= F - A^T Q_2 Q_1^{-1} - Q_1^{-T} Q_2^T A - Q_1^{-T} Q_2^T G Q_2 Q_1^{-1}. \end{aligned} \tag{4.25}$$

Multipliziert man (4.25) von links mit Q_1^T und von rechts mit Q_1, so folgt

$$Q_1^T \mathrm{Res}(\hat{X}) Q_1 = Q_1^T F Q_1 - Q_1^T A Q_2 - Q_2^T A Q_1 - Q_2^T G Q_2. \tag{4.26}$$

Multipliziert man die linke Seite von (4.23) aus, so ergibt sich für $\hat{F}$ genau der Ausdruck auf der rechten Seite von (4.26).
Falls $\hat{F}$ nicht Null im Rahmen der Rechengenauigkeit ist, ist auch die durch die ersten n Spalten von Q gegebene Näherungslösung $\hat{X}$ entsprechend ungenau. Daher benötigt man ein Verfahren, um diese zu verbessern. Dazu bietet sich das Defektkorrekturverfahren aus Kapitel 2.5 an. Wie oben gesehen, ist das dazu zu berechnende Residuum ähnlich zu $\hat{F}$. Zudem ist die Berechnung von $\hat{X}$ bei schlechter Kondition von Q_1 schlecht konditioniert und das damit berechnete Residuum evtl. kaum noch brauchbar, um damit Defektkorrektur durchzuführen. Man kann also davon ausgehen, daß $\hat{F}$, da ohne Matrixinvertierung berechnet, geeigneter ist, um aus der dadurch gelieferten Information

eine Verbesserung des invarianten Teilraums und damit der Lösung zu erreichen.
Statt das Defektkorrekturverfahren explizit anzuwenden, kann man LISH iterieren und die dabei erzeugten orthogonalen symplektischen Transformationen akkumulieren.
Dazu nehme zunächst an, daß $\sigma(\hat{H}) = \sigma(H)$, d.h. die Ähnlichkeitstransformationen seien exakt durchgeführt worden. Es sei nun der Lagrange–Teilraum zu den stabilen Eigenwerten von $\hat{H}$ aufgespannt durch die ersten n Spalten von $P \in \mathcal{US}_{2n}$. Dann gilt

$$\begin{bmatrix} \hat{A} & \hat{G} \\ \hat{F} & -\hat{A}^T \end{bmatrix} \begin{bmatrix} P_1 & -P_2 \\ P_2 & P_1 \end{bmatrix} = \begin{bmatrix} P_1 & -P_2 \\ P_2 & P_1 \end{bmatrix} \begin{bmatrix} T_1 & T_2 \\ 0 & -T_1^T \end{bmatrix} \tag{4.27}$$

mit $\sigma(T_1) = \{-\lambda_1, \ldots, -\lambda_n\}$, falls die instabilen, als Shifts benutzten Eigenwerte $\lambda_1, \ldots, \lambda_n$ waren. Damit folgt

$$\begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} T_1$$

mit

$$U := QP =: \begin{bmatrix} U_1 & -U_2 \\ U_2 & U_1 \end{bmatrix}.$$

Also spannen die ersten n Spalten von U den gesuchten H–invarianten Lagrange–Teilraum auf. Wird P wieder mit LISH berechnet, so ist der berechnete Teilraum natürlich wieder nicht der exakte, der (2,1)–Block der Hamiltonischen Matrix auf der rechten Seite von (4.2) wieder nicht exakt Null, d.h. man muß das Verfahren iterieren.
Man erhält somit ein Defektkorrekturverfahren für den gesuchten H–invarianten Lagrange–Teilraum statt für die Lösung der ARG. Die Näherungslösung für die ARG wird also verbessert, ohne sie explizit in jedem Schritt zu berechnen. Deshalb nenne diese Vorgehensweise *implizite Defektverbesserung.*
Damit kann man den *orthogonal symplektischen Multishift Algorithmus* zur Berechnung der Lösung der ARG, OSMARE (von engl. Orthogonal Symplectic Multishift algorithm for the solution of Algebraic Riccati Equations), wie folgt formulieren.

Algorithmus 4.6 (OSMARE)
Eingabe: $H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \in \mathcal{H}_{2n}$.

Ausgabe: *Eine Näherungslösung* X *der ARG*

$$0 = F + A^T X + XA - XGX.$$

1.Schritt:

Berechne die Eigenwerte von H *mit Algorithmus 3.8* SQRED.

2.Schritt:

Setze

$$H_0 := H =: \begin{bmatrix} A_0 & G_0 \\ F_0 & -A_0^T \end{bmatrix}, \quad Q := I_{2n}.$$

FOR $i = 1, 2, \ldots$ UNTIL $F_i \approx 0$

Berechne mit Algorithmus 4.5 LISH Q_i, H_i, *so daß*

$$H_i = Q_i^T H_{i-1} Q_i = \begin{bmatrix} A_i & G_i \\ F_i & -A_i^T \end{bmatrix},$$

und setze $Q := QQ_i$.

END FOR

3.Schritt:

Sei $Q = \begin{bmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{bmatrix}$. *Löse die lineare Matrixgleichung*

$$XQ_1 = -Q_2. \tag{4.28}$$

END.

Hat man in LISH einen p–dimensionalen Teilraum abgespalten, so bezieht sich die Iteration im zweiten Schritt nur noch auf die daraus resultierende Hamiltonische Untermatrix H_{22} aus (4.17).
Die lineare Matrixgleichung kann z.B. mit Hilfe der LR– oder QR–Zerlegung gelöst werden, siehe Abschnitt 5.1.7. Statt Q zu akkumulieren und X aus (4.28) zu berechnen, kann X mithilfe des Aufdatierungsverfahrens aus Kapitel 3.4 direkt aufdatiert werden. Nenne den daraus resultierenden Algorithmus dann OSMARU.
Der Aufwand des gesamten Verfahrens hängt von der Anzahl der Iterationsschritte im zweiten Schritt und der in LISH benötigten Deflationen ab und ist daher nicht genau zu bestimmen. Bei den berechneten Beispielen benötigt

der Algorithmus bei algebraischen Riccatigleichungen der Ordnung 15 oder kleiner i.d.R. nur 1–3 Iterationen. Mit wachsender Ordnung der ARG erhöht sich jedoch die Anzahl der Iterationen zusehends bzw. es kann sogar keine Konvergenz mehr beobachtet werden, siehe dazu auch Kapitel 5.1.8 und 5.3. Die gleichen Schwierigkeiten mit ihrem Multishift–Algorithmus beschreiben Raines/Watkins in [R 1].
Die Betrachtungen, die zur impliziten Defektverbesserung führten, zielten auf die Diagonalelemente von $\hat{F}$ ab, die in exakter Arithmetik nach einmaliger Durchführung von LISH Null wären. Nach Lemma 4.1 tritt Deflation dann ein, wenn ein Subdiagonalelement von $\hat{A}$ verschwindet. Wegen der oben erwähnten Rundefehler ergeben sich auch diese nicht mehr exakt. Damit wird es schwierig, die Deflation zu erkennen. Bei Beispiel 5.5 in Kapitel 5.3 tritt dieses Problem bei wachsender Ordnung der ARG auf. Die Schwierigkeit besteht nun darin, ein vernünftiges Abspaltungskriterium zu finden. Beim QR–Algorithmus benutzt man zu diesem Zwecke (siehe [G 1], [S 2]) meist

$$|\hat{h}_{k+1,k}| < c\mathbf{u}\left(|\hat{h}_{k,k}| + |\hat{h}_{k+1,k+1}|\right), \quad c = const.$$

Motiviert durch die Fehlerbetrachtung für die mit SQRED berechneten Eigenwerte wurde dieses Kriterium bei der Implementierung von OSMARE ersetzt durch

$$|\hat{h}_{k+1,k}| < c\sqrt{\mathbf{u}}\left(|\hat{h}_{k,k}| + |\hat{h}_{k+1,k+1}|\right). \tag{4.29}$$

Mit diesem Kriterium wurde die in Beispiel 5.5 zu erwartende Deflation auch noch bei höheren Dimensionen erkannt — für Näheres siehe Kapitel 5.3 — aber man handelt sich damit ein anderes Problem ein.
Spaltet man nach Kriterium (4.29) ab, wobei auch die zugehörigen Diagonalelemente des unteren linken Blockes, $\hat{h}_{n+1,1}, \ldots, \hat{h}_{2n,n}$, nur einem ähnlich gelockerten Kriterium genügen müssen, so verliert der daraus resultierende p–dimensionale Teilraum an Genauigkeit. Um das genauer zu beschreiben, kann man z.B. die aus der Partitionierung in (4.16) resultierende nichtsymmetrische Riccatigleichung

$$\begin{aligned} 0 \;=\; & \begin{bmatrix} A_{21} \\ F_{11} \\ F_{21} \end{bmatrix} + \begin{bmatrix} A_{22} & G_{21} & G_{22} \\ F_{12} & -A_{11}^T & -A_{21}^T \\ F_{22} & -A_{12}^T & -A_{22}^T \end{bmatrix} R - RA_{11} \\ & -R\,[\,A_{12} \;\; G_{11} \;\; G_{12}\,]\,R \end{aligned} \tag{4.30}$$

betrachten. Bestimmt man die Lösung von (4.30) mit Hilfe des abgespaltenen Teilraums und berechnet daraus die Frobeniusnorm des entstehenden Residuums, so beobachtet man in etwa die gleiche Größenordnung wie für $\hat{a}_{p+1,p}$, das Subdiagonalelement, das nach Kriterium (4.29) als Null betrachtet wurde. Damit hat man durch das gelockerte Abspaltungskriterium Informationen über den p–dimensionalen Teilraum verloren und es ist für die nach weiterer Durchführung von OSMARE berechnete Lösung keine höhere Genauigkeit zu erwarten.

Damit hat man das Problem, den abgespaltenen Teilraum zu verbessern. Eine Anwendung von LISH auf

$$\begin{bmatrix} \hat{A}_{11} & \hat{G}_{11} \\ \hat{F}_{11} & -\hat{A}_{11}^T \end{bmatrix}$$

führt zu einem Fill–in in den Blöcken $\hat{A}_{21}$ und den Nullblöcken in den Positionen (3,2) und (4,1). Damit muß LISH wieder auf die gesamte Matrix angewendet werden. Dadurch konnte in keinem berechneten Beispiel eine Verbesserung erzielt werden.

Eine andere Möglichkeit besteht darin, die Näherunglösung der nichtsymmetrischen Riccatigleichung (4.30) zu verbessern. Dazu kann z.B. eine der bei Demmel [D 1] beschriebenen Methoden benutzt werden. Damit verläßt man jedoch den vorgegebenen Rahmen, nur orthogonal symplektische Transformationen zuzulassen. Da dies nicht ohne weiteres in LISH bzw. OSMARE einbettbar ist, wurde dieser Ansatz zunächst nicht berücksichtigt.

Als am erfolgversprechendsten erwies es sich, eine mit einem gelockerten Abspaltekriterium wie z.B. (4.29) erkannte Deflation eines p–dimensionalen Teilraums nur dann als solche anzuerkennen, falls die ersten p Diagonalelemente des (2,1)–Blocks F Null im Rahmen der Rechengenauigkeit sind. Damit nutzt man allerdings nicht mehr alle theoretisch vorhandenen Eigenschaften des Multishift–Algorithmus aus, es werden zunächst einige implizite Defektkorrekturschritte nötig, bevor man abspalten kann. Dies führt wiederum zu den oben beschriebenen Problemen mit der Konvergenz dieses Prozesses, die allerdings nach einer Modifikation der Algorithmen COLRED und PVLRED, wie in Kapitel 5.1 beschrieben, erst bei großen Problemen ($n \approx 100$) auftreten.

4.3 Defektkorrektur mit dem Multishift–Algorithmus

In Kapitel 2.5 wurde ein allgemeines Defektkorrekturverfahren für die ARG vorgestellt. Zur Lösung der Defektgleichung in Algorithmus 2.35,

$$0 = \tilde{F} + \tilde{A}^T P + P\tilde{A} - PGP, \tag{4.31}$$

kann man auch OSMARE verwenden. Dieser ist hier besonders attraktiv, falls der zur Berechnung der Näherungslösung benutzte Algorithmus auch die (un-)stabilen Eigenwerte der zur ARG gehörenden Hamiltonischen Matrix liefert, wie z.B. die Schurvektormethode, der Hamiltonische SR–Algorithmus oder OSMARE selbst. Dabei hilft folgende Überlegung.
Es sei $\tilde{H} = \begin{bmatrix} \tilde{A} & G \\ \tilde{F} & -\tilde{A}^T \end{bmatrix}$ die zu ARG (4.31) für den Defekt gehörende Hamiltonische Matrix. Dann gilt

$$\tilde{H} = \begin{bmatrix} I & 0 \\ \tilde{X} & I \end{bmatrix} \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \begin{bmatrix} I & 0 \\ \tilde{X} & I \end{bmatrix}^{-1},$$

und daher $\sigma(H) = \sigma(\tilde{H})$. Damit kann der erste Schritt in OSMARE entfallen, was eine deutliche Kostenersparnis bedeutet, man erreicht jedoch i.A. trotzdem nicht die Konvergenzgeschwindigkeit des Newton–Verfahrens.
Das auf OSMARE beruhende explizite Defektkorrekturverfahren wurde als MATLAB–Funktion *dcorc* und als FORTRAN–Routine DCORC implementiert. Bei einigen Beispielen in Kapitel 5.3 wurde die explizite Defektkorrektur angewendet, um eine mit OSMARE gefundene Näherungslösung zu verbessern. Dabei kann man beobachten, daß nach ein bis drei Schritten meist keine Verbesserung mehr erzielt werden konnte. Dies beruht auf der bereits in Kapitel 2.5 festgestellten Tatsache, daß dann das neu zu berechnende Residuum durch Auslöschung an Genauigkeit verliert.

Kapitel 5

Implementierung und numerische Ergebnisse

5.1 Fortran–Subroutinen

Im Folgenden wird näher auf die Implementierung der zur Durchführung des Algorithmus 4.6 OSMARE und der Defektkorrektur nötigen Routinen eingegangen, wobei nur bisher nicht näher erläuterte Algorithmen ausführlich besprochen werden. Zudem werden Aufwandsschätzungen angegeben, und soweit nicht im Vorangegangenen bereits geschehen, einige Bemerkungen zu den numerischen Eigenschaften gemacht.

Die FORTRAN 77 Routinen sind gemäß den in [N 1] vorgegebenen Richtlinien ausführlich dokumentiert, so daß die Parameter hier nur kurz erläutert werden, ausführlichere Erklärungen findet man in den Kommentaren zu den Routinen. Im Folgenden bezeichnen in `typewriter` gesetzte Namen die in den Routinen benutzten Parameter.
Es werden die Bibliotheken **BLAS**, **LINPACK** [D 2] und **EISPACK** [S 2] benutzt. Außerdem werden einige Routinen von R. Byers [B 3] verwendet — insbesondere die Bestimmung von Householderreduktionen und deren Anwendung —, die sich auf der beiliegenden Diskette im Verzeichnis `FORTRAN\BYERS` befinden.

Da ausschließlich reelle Arithmetik verwendet wird, benötigt man zur Berechnung des Multishiftvektors in OSMARE bei Paaren von komplex konjugierten Shifts das Quadrat von $H \in \mathcal{H}_{2n}$. Aus diesem Grund wird $M = H^2$

explizit berechnet und abgespeichert. Sonst würde sich ein Aufwand von $O(n^4)$ ergeben, siehe Abschnitt 5.1.3. Da die quadrierte Hamiltonische Matrix explizit vorliegt, wurde zur Berechnung der Eigenwerte von H Algorithmus 3.8 in der in Kapitel 3.3 vorgestellten expliziten Version implementiert. Diese Routine (Abschnitt 5.1.2) wird zur Unterscheidung von der impliziten Implementierung [B 3] mit SQREDE bezeichnet. Der Algorithmus 4.5 LISH wurde nicht explizit implementiert. Der Aufruf von LISH in Algorithmus 4.6 wird hier durch die Routine OSMSTP bzw. deren Varianten OSMST2, OSMSTU realisiert. In diesen Routinen wird ein Iterationsschritt mit den Formeln (4.12) – (4.15) bzw. (4.18) – (4.21) aus LISH durchgeführt.

Die Bezeichnungen der an die Routinen übergegebenen Parameter genügen den folgenden Konventionen. Die zur ARG gehörende Hamiltonische Matrix

$$H = \begin{bmatrix} A & G \\ F & -A^T \end{bmatrix} \tag{5.1}$$

und ihr Quadrat

$$M = H^2 = \begin{bmatrix} AS & GS \\ FS & AS^T \end{bmatrix} \tag{5.2}$$

werden in den Feldern `A,G,F` bzw. `AS,GS,FS` abgespeichert. Dabei sind `A,AS` Felder der Dimension `(LDA,N)` mit `N` $= n$ für $A \in \mathbf{R}^{n,n}$ und `LDA` bezeichnet die führende Dimension von `A,AS` wie im aufrufenden Programm festgelegt. `G,GS,F,FS` liegen als Vektoren der Länge `N(N+1)/2` in symmetrischer Speicherweise vor, d.h. es wird in der Reihenfolge

`G(1,1)` $\to \cdots$ `G(1,N)` $\to$ `G(2,2)` $\to \cdots$ `G(2,N)` $\to \cdots$ `G(N,N)`

abgespeichert. Die als Shifts benutzten n instabilen Eigenwerte von H sind in zwei Vektoren der Länge `N` enthalten, wobei `WR` für die Realteile, `WI` für die Imaginärteile steht. Die ersten n Spalten der orthogonal symplektischen Transformationsmatrix

$$Q = \begin{bmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{bmatrix} \in \mathcal{US}_{2n}, \tag{5.3}$$

also $\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}$, werden im `(2*LDA,N)`–Feld `Q` abgespeichert.

Wird das Aufdatierungsverfahren aus Kapitel 3.4 benutzt, so werden die dazu benötigten Matrizen X, Y, Z entsprechend als `(LDA,N)`–Feld `Z` und als `N(N+1)/2` Vektoren `X,Y` in symmetrischer Speicherweise übergeben. Die darauf beruhenden Routinen werden mit PVLUPD.F, OSMSTU.F und OSMARU.F bezeichnet.
Die Dimension p eines abgespaltenen Teilraums wird in einem INTEGER `P` übergeben.

Benötigter Arbeitsspeicher wird mit `IWORK` für INTEGER–Werte, `RWORK` für DOUBLE PRECISION–Werte bezeichnet, Warnungen mit `IWARN` und Fehlermeldungen mit `IERR`.
Die Bedeutung aller anderer Parameter wird in den folgenden Abschnitten kurz erklärt.

Bei den Beispielrechnungen, die auf der Implementierung gemäß den in den Kapiteln 3 und 4 besprochenen Algorithmen beruhen, traten mit zunehmender Ordnung der ARG Probleme mit der Konvergenz der impliziten Defektkorrektur auf. Betrachtet man die Komponenten des Multishiftvektors und der Hamiltonischen Matrix in einem Iterationsschritt, so sieht man, daß diese sehr klein werden und z.T. stark unterschiedliche Größenordnungen besitzen. Daher kann man für die mit Householder–Reduktionen zu eliminierenden Elemente Probleme erwarten, da in Algorithmus 3.2 HHGEN diese unterschiedlichen Größenordnungen nicht berücksichtigt werden und damit die — auch in betragsmäßig sehr kleinen Elementen — vorhandene Information verlorengeht.
Nun kann man die Aufrufe von HHGEN in COLRED und PVLRED durch Givens–Rotationen ersetzen. Dazu sei x der jeweils zu reduzierende Vektor, also der Multishiftvektor oder die j–te Spalte der Hamiltonischen Matrix. Dann können die mit den Algorithmen 3.5 (COLRED) und 3.6 bestimmten Reduktionen auch wie folgt berechnet werden.

Eliminierung der Elemente in der unteren Hälfte:

$$
\begin{array}{l}
\text{FOR } i = n, \ldots, 1\ (j+1) \\
\quad \begin{array}{lcl}
[c, s] & = & \text{GIVGEN}(i, n+i, x) \\
x(i) & = & cx(i) + sx(n+i) \\
x(n+i) & = & 0
\end{array} \\
\text{END.}
\end{array}
\tag{5.4}
$$

Eliminierung der Elemente in der oberen Hälfte:

$$
\begin{array}{ll}
 & \text{FOR } i = n, \ldots, 2\,(j+2) \\
 & \quad [c, s] \quad = \ \text{GIVGEN}(i-1, i, x) \\
(5.5) & \quad x(i-1) \ = \ cx(i-1) + sx(i) \\
 & \quad x(i) \quad\ \, = \ 0 \\
 & \text{END.}
\end{array}
$$

Dabei bezieht sich obiger Algorithmus auf den Aufruf in COLRED, die Angaben in Klammern auf den Aufruf für die j–te Spalte in PVLRED.
Bei den Givens–Rotationen aus (5.4) handelt es sich um gewöhnliche symplektische Givens–Matrizen gemäß (3.7). Die Rotationen aus (5.5) definieren Givens–Matrizen $G(i-1, i, c, s) \in \mathbf{R}^{n,n}$. Setzt man

$$
U = \begin{bmatrix} G(i-1, i, c, s) & 0 \\ 0 & G(i-1, i, c, s) \end{bmatrix},
$$

so überprüft man leicht, daß $U \in \mathcal{US}_{2n}$, so daß alle aus obigem Algorithmus resultierenden Transformationen orthogonal und symplektisch sind. Ein Aufruf von GIVGEN erfolgt gemäß

```
CALL GIVGEN(X, Y, CO, SI)
```

Hierbei definieren `CO,SI` eine 2×2–Givens–Rotation, so daß

$$
\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \pm\sqrt{x^2 + y^2} \\ 0 \end{bmatrix}.
$$

Die Ähnlichkeitstransformationen der Hamiltonischen Matrix mit obigen Givens–Rotationen können mit den in Kapitel 3.1.2 angegebenen Algorithmen erfolgen.
Die aus (5.4), (5.5) resultierenden Algorithmen bezeichne mit COLRED2 und PVLRED2, die darauf basierenden FORTRAN–Routinen heißen COLRD2, PVLRD2, OSMST2 (siehe Abschnitte 5.1.4, 5.1.5 und 5.1.6). Die Implementierung von Algorithmus 4.6, die diese Routinen benutzt, wurde mit OSMAR2 bezeichnet. Bei dieser Vorgehensweise sind also alle benutzten Reduktionen Givens–Rotationen.
Im Falle des symmetrischen Aufdatierungsverfahrens gelten für (5.4) die Formeln (3.18) – (3.20). Da für das Aufdatieren jeder symplektischen Givens–Matrix mit diesem Formelsatz ein Aufwand von $O(n^2)$ nötig ist und bei der

Verwendung in PVLRED durch (5.4) insgesamt $n(n-1)/2$ dieser Transformationen erzeugt werden, ergäbe ein Algorithmus, der (5.4), (5.5) und das symmetrische Aufdatierungsverfahren aus Kapitel 3.4 verwendet, einen Aufwand von $O(n^4)$ und wäre somit nicht mehr effizient. Daher wurde auf die Implementierung dieser Kombination verzichtet.

5.1.1 HAMSQR.F

Diese Routine dient zum Quadrieren einer Hamiltonischen Matrix H.
Die Berechnung der Eigenwerte einer Hamiltonischen Matrix mit der expliziten Form des Algorithmus 3.8 benutzt H^2, beim Algorithmus MSHIFT wird H^2 benötigt, um bei einem Shift mit einem komplexen Eigenwert komplexe Arithmetik zu vermeiden (vgl. Abschnitt 5.1.3).
Das Quadrieren einer Hamiltonischen Matrix $H \in \mathcal{H}_{2n}$ kann natürlich durch den Aufruf einer Routine, die zwei allgemeine Matrizen multipliziert, geschehen. Dafür werden $16n^3$ Flops und zur Speicherung der quadrierten Matrix ein $(2n)^2$–dimensionales Feld benötigt. (Wie bereits erwähnt, werden in OSMARE sowohl H^2 als auch H benötigt, so daß der Speicherplatz von H nicht überschrieben werden kann.)
Berücksichtigt man jedoch die spezielle Struktur Hamiltonischer Matrizen, so kann man einige Verbesserungen erzielen. Es gilt

$$(5.6) \qquad H^2 = S = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} = \begin{bmatrix} A^2 + GF & AG - GA^T \\ FA - A^T F & (A^2 + GF)^T \end{bmatrix}.$$

Damit folgt sofort $S_{22} = S_{11}^T$, $S_{12} = -S_{12}^T$ und $S_{21} = -S_{21}^T$, d.h. S_{12} und S_{21} sind schiefsymmetrisch und daher

$$\mathrm{diag}(S_{12}) = \mathrm{diag}(S_{21}) = (0, \ldots, 0).$$

Berechnet man $AS := S_{11}$, $GS := S_{12}$ und $FS := S_{21}$ gemäß (5.6), so erhält man die Struktur einer quadrierten Hamiltonischen Matrix S, die ansonsten aufgrund von Rundefehlern verlorengeht.
Speichert man GS und FS in symmetrischer Speicherform, so reicht dafür je ein Vektor der Länge $n(n-1)/2$. Insgesamt benötigt man zur Speicherung von S also ein n^2-dimensionales Feld für AS und zwei Vektoren der Länge $n(n-1)/2$, d.h. weniger als die Hälfte des Speicherbedarfs für das normale Matrixprodukt.

Bei der Implementierung wurde dies weitestgehend ausgenutzt. Da die Felder `A,G,F` nicht überschrieben werden können,muß zusätzlicher Speicherplatz zur Verfügung gestellt werden. Dabei werden die schiefsymmetrischen Blöcke `GS,FS` in symmetrischer Form abgespeichert, also in Vektoren der Länge $n(n+1)/2$, da diese Felder somit entsprechend den symmetrischen Matrizen angesprochen und an anderer Stelle zur Speicherung von symmetrischen Matrizen benutzt werden können.
Die Berechnung von $AS = A^2 + GF$ erfordert $4n^3$ Flops, während für die $n(n-1)/2$ abzuspeichernden Komponenten von GS bzw. FS je $4n$ Flops nötig sind, so daß sich der Aufwand für HAMSQR insgesamt zu $8n^3 - 4n^2$ ergibt, was wiederum weniger als der Hälfte des Aufwandes für eine gewöhnliche Matrixmultiplikation entspricht. Die erzielte Geschwindigkeitssteigerung gegenüber der gewöhnlichen Matrixmultiplikation entspricht nicht ganz der theoretisch vorhergesagten, da die Feldzugriffe und Indizierung hier aufwendiger sind.
Damit kann man eine Fehlerschätzung für das numerisch berechnete Quadrat einer Hamiltonischen Matrix auf der Basis von (5.6) angeben. Mit Standardtechniken (vgl. Kapitel 3.3 bzw. [G 1]) ergibt sich

$$\begin{aligned} \|fl(H^2) - H^2\|_F &= 2n\mathbf{u}\|H\|_F^2 + O(\mathbf{u}^{3/2}) \\ &= 2n\mathbf{u}\left(2\|A\|_F^2 + \|G\|_F^2 + \|F\|_F^2\right) + O(\mathbf{u}^{3/2}). \end{aligned}$$

Benutzt man (5.6) und die üblichen Abschätzungen für numerisch berechnete Produkte und Quadrate von Matrizen [G 1], übertragen auf die Frobeniusnorm,

$$\|fl(A+B) - (A+B)\|_F \leq \mathbf{u}(\|A\|_F + \|B\|_F) \tag{5.7}$$

$$\|fl(AB) - (AB)\|_F \leq n\mathbf{u}\|A\|_F\|B\|_F + O(\mathbf{u}^{3/2}), \tag{5.8}$$

mit $A \in \mathbf{R}^{l,n}$ und $B \in \mathbf{R}^{l,n}$ für (5.7) bzw. $B \in \mathbf{R}^{n,k}$ für (5.8), so ergibt sich mit der bereits festgestellten Eigenschaft, daß auch $fl(H^2)$ eine quadrierte Hamiltonische Matrix darstellt, folgende Fehlerabschätzung

$$\begin{aligned} \|fl(H^2) - H^2\|_F &= \Big\{2\|fl(S_{11}) - S_{11}\|_F^2 + \|fl(S_{12}) - S_{12}\|_F^2 \\ &\qquad + \|fl(S_{21}) - S_{21}\|_F^2\Big\}^{1/2} \\ &\leq (n+1)\mathbf{u}\Big\{2(\|A\|_F^2 + \|G\|_F\|F\|_F)^2 + (2\|A\|_F\|G\|_F)^2 \end{aligned}$$

$$
\begin{aligned}
&\quad + (2\|A\|_F\|F\|_F)^2\Big\}^{1/2} + O(\mathbf{u}^{5/4}) \\
= &\quad \sqrt{2}(n+1)\mathbf{u}\Big\{\|A\|_F^4 + \|G\|_F^2\|F\|_F^2 \\
&\quad + 2\|A\|_F^2(\|G\|_F^2 + \|G\|_F\|F\|_F + \|F\|_F^2)\Big\}^{1/2} + O(\mathbf{u}^{5/4})
\end{aligned}
$$

Man sieht, daß das Quadrieren einer Hamiltonischen Matrix mit großer Norm große Fehler erzeugen kann, da der Fehler mit dem Quadrat der Norm von A und dem Produkt der Normen von A, G und F wächst.

Der Aufruf von HAMSQR erfolgt mit

```
CALL HAMSQR(N, P, A, LDA, G, F, AS, GS, FS, IERR)
```

Für `1` $\leq$ `P` $\leq$ `N-1` wird das Quadrat einer aus den unteren rechten (`N-P`$\times$`N-P`)–Blöcken von `A,G,F` gebildeten Hamiltonischen Matrix berechnet. Als Fehlermeldung kann hier auftreten

`IERR = 1` : `N < 1` oder `LDA < N` oder `P < 0` oder `P` $\geq$ `N`.

5.1.2 SQREDE.F, SKWREF.F

Die Routine SQREDE.F dient zur Berechnung der Eigenwerte einer Hamiltonischen Matrix, SKWREF.F zur Ähnlichkeitstransformation symmetrischer Matrizen mit einer Householder–Transformation.

Zum Transformieren einer quadrierten Hamiltonischen Matrix mit Householder–Matrizen benötigt man eine Routine, die die schiefsymmetrischen Matrizen GS, FS effizient transformiert. Dies kann ähnlich wie bei symmetrischen Matrizen, siehe [W 1], geschehen. Dazu sei A schiefsymmetrisch und $P = I - \beta vv^T$ eine Householder–Matrix. Eine Ähnlichkeitstransformation von A mit P kann mit Hilfe des Formelsatzes

$$w = \beta A v \tag{5.9}$$

$$A = A + vw^T - wv^T \tag{5.10}$$

durchgeführt werden. Dies wurde so in der Routine SKWREF.F implementiert, die mit

```
CALL SKWREF(K, N, A, V, S, P, RWORK, IERR)
```

aufgerufen wird. Dabei entspricht β aus (5.9) `S`. Durch `K,N,V,S` wird eine Householdermatrix $H(k, n, v)$ gemäß Kapitel 3.1.1 definiert, `P` gibt die erste Zeile der Hauptuntermatrix von `A`, die transformiert werden soll, an.
Als Fehlermeldung kann auftreten
`IERR = 1` : `N < 1` oder `K < 1` oder `K` $\geq$ `N` oder `P < 1` oder `P > K`.

Die numerischen Eigenschaften von SQREDE wurden bereits in Kapitel 3.3 erläutert, dort wurde auch der Aufwand mit $32n^3$ angegeben. Die Implementierung hält sich exakt an Algorithmus 3.8, wobei die Paige/Van Loan–Reduktion der quadrierten Hamiltonischen Matrix in der Routine selbst erfolgt, d.h. PVLRED wird nicht explizit aufgerufen. Es werden n Eigenwerte mit nicht–negativem Realteil zurückgegeben, wobei, soweit möglich, komplex konjugierte Paare aufeinanderfolgen. Falls rein–imaginäre Eigenwerte mit ungerader Vielfachheit existieren, erfolgt eine Warnung mittels `IWARN`.
Der Aufruf von SQREDE erfolgt mit

```
CALL SQREDE(N, AS, LDA, GS, FS, WR, WI, RWORK, IWARN, IERR)
```

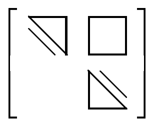
Es muß also die bereits quadrierte Hamiltonische Matrix in `AS,GS,FS` übergeben werden, `WR,WI` enthalten dann bei der Ausgabe die Real- bzw. Imaginärteile von `N` nicht–negativen Eigenwerten der Hamiltonischen Matrix.
Es können folgende Fehler und Warnungen auftreten:

`IWARN = 1` : Es ist mindestens ein Eigenwert ungerader Vielfachheit vorhanden.
`IERR  = 1` : `N < 1` oder `LDA < N` oder SKWREF gibt `IERR = 1` zurück.
`IERR  =-J` : Falls die Berechnung des `J`–ten Eigenwertes nicht innerhalb der maximalen Anzahl an Iterationsschritten möglich ist, siehe EISPACK–Routinen RG, HQR.

5.1.3 MSHIFT.F

Diese Routine dient zur Berechnung der Multishiftvektoren (4.12), (4.18).
Die grundlegende Idee von OSMARE besteht darin, die n instabilen Eigenwerte der zur ARG gehörenden Hamiltonischen Matrix H zu shiften, so daß mit

$$Q^T H Q =$$



für die in Algorithmus 4.5 berechnete orthogonal symplektische Matrix Q die stabilen Eigenwerte von H gerade die Eigenwerte des linken oberen Hessenberg–Blocks sind.
Dieser Multishift wird im ersten Schritt von Algorithmus 4.5 für H und bei jeder Abspaltung eines p–dimensionalen Teilraums sowie der impliziten Defektverbesserung für die entsprechenden transformierten Hamiltonischen Matrizen durchgeführt (Formeln (4.12), (4.18) in LISH).
Bei diesem n–fachen Shiften handelt es sich um die Berechnung eines Vektors z gemäß

$$z = (H - \lambda_1 I_{2n})(H - \lambda_2 I_{2n}) \cdots (H - \lambda_n I_{2n})e_1. \tag{5.11}$$

Dabei sei $\sigma(H) = \{\lambda_1, -\lambda_1, \ldots, \lambda_n, -\lambda_n\}$. Wegen Folgerung 2.12 kann man $\sigma(H)$ so anordnen, daß $Re(\lambda_i) \geq 0$ für $i = 1, \ldots, n$ gilt und $\{\lambda_1, \ldots, \lambda_n\}$ abgeschlossen unter komplexer Konjugation ist. Also kann man Œ annehmen, daß die λ_i so angeordnet sind, daß für komplexe Werte $\lambda_{i+1} = \bar{\lambda}_i$ gilt. Dann kann (5.11) in *reeller* Arithmetik wie folgt durchgeführt werden:

Man setzt zunächst $z = e_1$, falls nun λ_i reell ist, so berechnet man

$$z = (H - \lambda_i I_{2n})z, \tag{5.12}$$

sonst wegen $\lambda_{i+1} = \bar{\lambda}_i$

$$z = (H - \lambda_i I_{2n})(H - \bar{\lambda}_i I_{2n})z. \tag{5.13}$$

Offensichtlich läßt sich der *Doppelshift* (5.13) reell berechnen unter Ausnutzung folgender Beziehung für $x \in \mathbf{R}$, $\lambda \in \mathbf{C}$,

$$(x - \lambda)(x - \bar{\lambda}) = x^2 - 2Re(\lambda)x + |\lambda|^2.$$

Bei der Implementierung von MSHIFT wird dies benutzt. Da H^2 bereits für SQREDE benötigt wird, kann (5.13) mit $16n^2$ Flops berechnet werden, bzw., falls ein p–dimensionaler Teilraum abgespalten wurde, mit $16(n-p)^2$ Flops. Für (5.12) benötigt man $8n^2$ Flops, bzw. im Falle der Abspaltung $8(n-p)^2$ Flops. Sei nun n_{re} die Anzahl der reellen Eigenwerte und n_{im} die Anzahl der Paare von komplex konjugierten Eigenwerten, also $n = n_{re} + 2n_{im}$, dann ergibt sich der Aufwand für MSHIFT zu $8n^2 n_{re} + 16n^2 n_{im} = 8n^3$ Flops bzw. im Falle der Abspaltung zu $8n(n-p)^2$ Flops.
Um den Rundefehlereinfluß gering zu halten, wird nach (5.12) und (5.13) jeweils z normiert, also $z := z/\|z\|_2$ gesetzt, so daß nach jedem Einfach–

oder Doppelshift $\|z\|_2 = 1$ gilt. Bezeichnet man die aus (5.12), (5.13) resultierenden Vektoren mit $z_1, \ldots, z_{n_{re}+n_{im}}$, so ergibt sich α aus Algorithmus 4.5 als

$$\alpha = \frac{1}{\|z_1\|_2 \cdots \|z_{n_{re}+n_{im}}\|_2} .$$

Der Aufruf von MSHIFT erfolgt mit

```
  CALL MSHIFT(N, L, P, A, AS, LDA, G, GS, F, FS, WR, WI, Z,
 *            RWORK, TOL, IERR)
```

Dabei bezeichnet `L` die Anzahl der übergebenen Shifts, in `Z` wird der Multishiftvektor zurückgegeben. Hier ist zu beachten, daß gemäß (4.12) und (4.18) $\mathtt{N} = 2n$ oder $\mathtt{N} = 2(n-p)$ gesetzt werden muß.
`TOL` bezeichnet eine Toleranz, die dazu benutzt wird, zu enscheiden, ob ein Eigenwert reell ist, bzw. ob es sich bei zwei aufeinanderfolgenden Eigenwerten um ein komplex konjugiertes Paar handelt. Mit $\lambda_k = \mathtt{WR(K)} + \imath\,\mathtt{WI(K)}$ gilt:

$$\begin{aligned} |\mathtt{WI(K)}| < \mathtt{TOL}|\lambda_k| \quad &\longrightarrow \quad \lambda_k \quad \text{reell}, \\ |\mathtt{WI(K)} + \mathtt{WI(K+1)}| < \mathtt{TOL}|\lambda_k| \quad &\longrightarrow \quad \lambda_k = \bar{\lambda}_{k+1}. \end{aligned}$$

Es können folgende Fehlermeldungen auftreten:
`IERR = 1` : `N < 1` oder `L < 1` oder `LDA < N` oder `P < 0` oder `P` $\geq$ `N`.
`IERR = 2` : $\mathtt{WR(1)} + \imath\,\mathtt{WI(1)}, \ldots, \mathtt{WR(L)} + \imath\,\mathtt{WI(L)}$ ist nicht abgeschlossen unter komplexer Konjugation bzgl. der relativen Toleranz `TOL`.

5.1.4 COLRED.F, COLRD2.F

Mit Hilfe dieser Routinen kann ein Vektor $z \in \mathbf{R}^{2n}$ durch orthogonal symplektische Matrizen auf ein Vielfaches des ersten Einheitsvektors reduziert werden.
Dabei wurde COLRED.F anhand von Algorithmus 3.5, COLRD2.F mit Hilfe von (5.4), (5.5) implementiert. Da jeweils nur orthogonale Matrizen benutzt werden, sind beide Algorithmen, wie bereits in Kapitel 3.1.3 dargestellt, numerisch rückwärts stabil. Der Aufwand für COLRED beträgt $10n$ Flops, der für COLRED2 $18n$ Flops. Aufgerufen werden diese Routinen mit

```
  CALL COLRED(N, Z, S1, S2, RWORK, IERR)
```

```
  CALL COLRD2(N, Z, IERR)
```

Dabei ist Z der zu reduzierende Vektor, N die Länge des Vektors, d.h. $\mathtt{N} = 2n$. In beiden Fällen enthält Z(1) bei der Ausgabe $\pm\|z\|_2$, während die restlichen Komponenten von Z zur Abspeicherung von Informationen über die orthogonal symplektischen Transformationen genutzt werden. Dabei werden Givens–Rotationen zur Eliminierung von Z(I) in Z(I) codiert (zur Speicherung einer Givens–Rotation in einem einzigen Skalar siehe [G 1], S.204/205). Dazu werden pro Rotation zusätzlich zwei Flops und zwei IF–Abfragen (eine aus der Codierung, eine um das richtige Vorzeichen zu erhalten) benötigt.
Mit den Bezeichnungen aus Algorithmus 3.5 gilt für COLRED: Die wesentlichen Komponenten von v_1 werden in Z(1+N/2), ..., Z(N), die von v_2 in Z(2), ..., Z(N/2) abgespeichert, β_1, β_2 entsprechen S1,S2 und c, s werden in Z(1+N/2) codiert.
Bei COLRD2.F enthalten Z(2), ..., Z(N) die codierte Information über die verwendeten Givens–Rotationen. Wegen der für die Codierung nötigen Flops ist der Gesamtaufwand für COLRD2.F mit $20n = \mathtt{10N}$ doppelt so hoch wie für COLRED.F, die zusätzlichen IF–Abfragen verlangsamen den Algorithmus noch weiter. Außerdem erfordert das Decodieren im aufrufenden Programm 4 Flops pro Rotation. Dieser Mehraufwand fällt bei der Benutzung in OSMARE aber nicht ins Gewicht, da es sich dabei um einen $O(n^3)$–Algorithmus handelt und zudem die numerischen Eigenschaften bei der Defektkorrekturiteration wie eingangs erwähnt denen der Householderreduktionen überlegen sind.
Bei beiden Routinen kann folgender Fehlercode auftreten:
IERR = 1 : N < 1 oder N $\neq$ 2M für eine natürliche Zahl M.

5.1.5 PVLRED.F, PVLRD2.F, PVLUPD.F

Diese Routinen dienen zur orthogonal symplektischen Transformation einer Hamiltonischen Matrix auf PVL–Form.
Algorithmus 3.7 wurde in drei verschiedenen Routinen implementiert. Dabei werden in PVLRED.F und PVLRD2.F die orthogonal symplektischen Transformationen akkumuliert, während PVLUPD.F das symmetrische Aufdatierungsverfahren aus Kapitel 3.4 benutzt. In PVLRED.F und PVLUPD.F wurde Algorithmus 3.7 direkt benutzt, in PVLRD2.F werden die Reduktionen mit (5.4), (5.5) berechnet, d.h. es werden ausschließlich Givens–Rotationen verwendet. Da alle Routinen nur Ähnlichkeitstransformationen aus $\mathcal{US}_{2n}$ benutzen und die Hamiltonische Struktur erhalten bleibt, sind sie aufgrund

der Betrachtungen aus Kapitel 3.1.3 numerisch rückwärts stark stabil. Die Routinen werden wie folgt aufgerufen:

```
CALL PVLRED(N, P, A, LDA, G, F, Q, RWORK, WITHQ, IERR)

CALL PVLRD2(N, P, A, LDA, G, F, Q, WITHQ, IERR)
```

bzw.

```
CALL PVLUPD(N, P, A, LDA, G, F, X, Y, Z, RWORK, IWARN, IERR)
```

Dabei legt `P` die Dimension der zu reduzierenden Hamiltonischen Untermatrix fest, d.h. es wird die durch die rechten unteren `(N-P)`$\times$`(N-P)`–Blöcke von `A,G,F` gegebene Hamiltonische Matrix auf PVL–Form transformiert, die Transformationen aber auf die gesamte Hamiltonische Matrix angewendet. Bei PVLRED.F, PVLRD2.F gibt `WITHQ` an, ob die Transformationen in `Q` akkumuliert werden sollen oder nicht.

Als Fehler und Warnungen können auftreten:

`IWARN = 1` : Beim Aufdatieren nach (3.18) – (3.20) mußte durch $\alpha < 10^{-8}$ dividiert werden (nur bei PVLUPD).

`IERR  = 1` : `N < 1` oder `N > LDA` oder `P < 0` oder `P` $\geq$ `N`.

Den Aufwand der einzelnen Algorithmen kann man folgender Tabelle entnehmen:

	PVLRED.F	PVLRD2.F	PVLUPD2
Erz.[a]	$5(n-p)^2$	$6(n-p)^2$	$5(n-p)^2$
Ähnl.[b]	$(14n-6p)(n-p)^2$	$(15n-6p)(n-p)^2$	$(14n-6p)(n-p)^2$
Aufd.[c]	$8n(n-p)^2$	$9n(n-p)^2$	$n(n-p)(20n-8p)$
Gesamt	$(22n-6p)(n-p)^2$	$(24n-6p)(n-p)^2$	$34n(n-p)^2+6p(n^2-p^2)$

[a]Erzeugung der Transformationen
[b]Ähnlichkeitstransformationen
[c]Aufdatieren der Transformationen

5.1.6 OSMSTP.F, OSMST2.F, OSMSTU.F

Diese Routinen dienen zur Durchführung eines Deflations- oder impliziten Defektkorrekturschrittes für Algorithmus 4.6.

Wie bereits erwähnt, besteht ein Schritt im Multishiftalgorithmus 4.6 prinzipiell aus

1. Berechnung eines „Multishiftvektors“,
2. Bestimmung einer orthogonal symplektischen Matrix zur Reduktion dieses Vektors auf ein Vielfaches des ersten Einheitsvektors,
3. Reduktion der mit dieser Matrix transformierten Hamiltonischen Matrix auf PVL–Form,

wobei in 2. und 3. die orthogonal symplektischen Transformationen entweder in $Q \in \mathcal{US}_{2n}$ akkumuliert oder die Lösung direkt aufdatiert wird. Es werden also (4.12) – (4.15) oder (4.18) – (4.21) aus Algorithmus 4.5 durchgeführt. OSMSTP liegt analog zu PVLRED in den drei Routinen

- OSMSTP.F (ruft COLRED und PVLRED auf),
- OSMST2.F (ruft COLRD2 und PVLRD2 auf),
- OSMSTU.F (ruft COLRED und PVLUPD auf)

vor, die wie folgt aufgerufen werden:

```
  CALL OSMSTP(N, P, A, LDA, G, F, AS, GS, FS, Q, WR, WI,
 *            RWORK, TOL, WITHSQ, IERR)

  CALL OSMST2(N, P, A, LDA, G, F, AS, GS, FS, Q, WR, WI,
 *            RWORK, TOL, WITHSQ, IERR)

  CALL OSMSTU(N, P, A, LDA, G, F, AS, GS, FS, X, Y, Z, WR, WI,
 *            RWORK, TOL, WITHSQ, IWARN, IERR)
```

Falls eine der Routinen mit `P > 0` aufgerufen wird, so werden die Schritte (4.18) – (4.21) aus Algorithmus 4.5, sonst (4.12) – (4.15) ausgeführt.
`WITHSQ` entscheidet, ob das Quadrat der Hamiltonischen Matrix in `AS,GS,FS` vorliegt oder noch berechnet werden muß, `TOL` wird als Toleranz für den Aufruf von MSHIFT verwendet und hat die dort (Abschnitt 5.1.3) dargelegte Bedeutung.
Als Fehler und Warnungen können auftreten:
`IWARN = 2` : Beim Aufdatieren nach (3.18) – (3.20) mußte durch $\alpha < 10^{-8}$ dividiert werden (nur bei OSMSTU).

`IERR = 1` : `N < 1` oder `LDA < N` oder `P < 0` oder `P` $\geq$ `N` oder HAMSQR, COLRED (COLRD2), MSHIFT, PVLRED (PVLRD2, PVLUPD) liefern `IERR = 1` zurück.

`IERR = 2` : MSHIFT liefert `IERR = 2` zurück, d.h. die Shifts sind nicht abgeschlossen unter komplexer Konjugation bzgl. der relativen Toleranz `TOL`.

Der Aufwand beträgt, falls `WITHSQ .EQ. .FALSE.`, für

- OSMSTP $(30n - 6p)(n - p)^2$,
- OSMST2 $(32n - 6p)(n - p)^2$,
- OSMSTU $42n(n - p)^2 + 6p(n^2 - p^2)$

Flops. Dazu kommen jeweils $8(n - p)^3$ Flops, falls die Hamiltonische Matrix noch quadriert werden muß, d.h. falls `WITHSQ .EQ. .TRUE.`.

5.1.7 XSOLVE.F

Diese Subroutine dient zum Lösen einer linearen Matrixgleichung

$$XA = B. \tag{5.14}$$

Dieses Problem tritt bei der Lösung der ARG mit OSMARE auf. Falls die orthogonal symplektischen Transformationsmatrizen in

$$Q = \begin{bmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{bmatrix}$$

akkumuliert werden, ergibt sich die Lösung der ARG mit Satz 2.19 gemäß

$$X = -Q_2 Q_1^{-1},$$

d.h. X ist Lösung von (5.14) mit $A = Q_1,\ B = -Q_2$.
Zur Lösung von (5.14) wird zunächst die LR–Zerlegung von A mit Spaltenpivotisierung (partielle Pivotisierung, vgl. [G 1]) berechnet, d.h. es werden eine Permutationsmatrix Π, eine untere Dreiecksmatrix L und eine obere Dreiecksmatrix R bestimmt, so daß

$$\Pi A = LR.$$

Mit $XA = B$ ist $A^T X^T = B^T$, so daß sich X durch Vorwärts- und Rückwärtseinsetzen wie folgt ergibt:

$$R^T Y = B^T, \quad L^T \tilde{X} = Y, \quad X = \tilde{X}^T \Pi.$$

Zur Berechnung der LR–Zerlegung von A wird die LINPACK–Subroutine DGEFA oder DGECO benutzt (DGECO liefert zusätzlich eine Schätzung für $1/\kappa_\infty(A)$, die reziproke Kondition von A in der Zeilenmaximumnorm), das Vorwärts- und Rückwärtseinsetzen geschieht durch n–maligen Aufruf der LINPACK–Subroutine DGESL, so daß im i–ten Aufruf die i–te Spalte von X berechnet wird. Da DGESL den Vektor der rechten Seite mit dem Lösungsvektor überschreibt, geschieht das Transponieren von X implizit.

Der Aufwand für die LR–Zerlegung beträgt im Wesentlichen $2n^3/3$ Flops, für das Vorwärtseinsetzen n^2 Flops pro Spalte von X, ebenso für das Rückwärtseinsetzen, so daß der Gesamtaufwand für XSOLVE $8n^3/3$ beträgt.

Für DGEFA und DGESL findet man in [D 2] eine Fehlerabschätzung, wobei der relative Fehler für einen mit diesen Routinen berechneten Lösungsvektor eines linearen Gleichungssystems (hier also für jede Spalte von X) linear von der Kondition von A abhängt.
In der Anwendung auf die Lösung der ARG kann man also Schwierigkeiten erwarten, falls Q_1 schlecht konditioniert ist. Dies tritt bei Beispiel 5.6 in Kapitel 5.3 auf. Daher erklärt sich auch das bessere Ergebnis in diesem Beispiel bei Verwendung des symmetrischen Aufdatierungsverfahrens.

Der Aufruf von XSOLVE geschieht mit

```
  CALL XSOLVE(N, M, A, LDA, B, LDB, RCOND, IWORK, RWORK,
 *             WITHRC, IERR)
```

Dabei ist `A` eine `N`×`N`, `B` eine `M`×`N`–Matrix, `LDA,LDB` bezeichnen die führenden Dimensionen von `A, B` wie im aufrufenden Programm festgelegt. Bei der Ausgabe enthält `A` die LR–Zerlegung, `B` die Lösung X.
Mit `WITHRC` wird festgelegt, ob DGEFA oder DGECO für die LR–Zerlegung benutzt werden. Falls `WITHRC .EQ. .TRUE.` wird DGECO aufgerufen und in `RCOND` eine Schätzung für $1/\kappa_\infty(A)$ zurückgegeben.
Es können folgende Fehler auftreten:
`IERR = 1` : `N < 1` oder `LDA < N` oder `M < 1` oder `LDB < M` oder `M > N`.
`IERR = 2` : `A` ist singulär bzgl. der Rechengenauigkeit.

Letztere Fehlermeldung wird nur erzeugt, falls DGEFA direkt aufgerufen wird und eine entsprechende Meldung zurückgibt, in OSMARE wird dieser Fehler in `IERR = 4` umgewandelt.

5.1.8 OSMARE.F, OSMAR2.F, OSMARU.F

Hierbei handelt es sich um vollständige Implementierungen von Algorithmus 4.6.
Analog zu OSMSTP liegt auch OSMARE zusätzlich in den Versionen OSMAR2 (alle Transformationen sind Givens–Rotationen, $\rightarrow$ OSMST2) und OSMARU (die Lösung der ARG wird direkt aufdatiert, $\rightarrow$ OSMSTU) vor. Falls die Eigenwerte der zugrundeliegenden Hamiltonischen Matrix H bekannt sind, können diese an OSMARE übergeben werden, ansonsten werden sie gemäß Schritt 1 aus Algorithmus 4.6 mit SQREDE berechnet. Ein Iterationsschritt entspricht dem Aufruf von OSMSTP (OSMST2, OSMSTU), wobei Deflation anhand des zu (4.29) aus Kapitel 4.2 analogen Kriteriums,

$$|\mathtt{A(K+1,K)}| < \frac{\mathtt{TOL}}{\sqrt{\mathbf{u}}}\left(|\mathtt{A(K,K)}| + |\mathtt{A(K+1,K+1)}|\right)$$

überprüft wird. Ist dieses Kriterium für ein `P`, $1 \leq$ `P` $<$ `N`, erfüllt und sind die Diagonalelemente von `F`, $|\mathtt{F(1,1)}|, \ldots, |\mathtt{F(P,P)}| < \mathtt{TOL}$, so wird OSMSTP (OSMST2, OSMARU) mit `P` aufgerufen. Dies führt z.B. bei Beispiel 5.5 zu den in Kapitel 4.2 angeführten Schwierigkeiten mit dem abgespaltenen Teilraum, wobei die Probleme für OSMAR2 erst bei sehr großen `N` eintreten.
Der Aufruf der Routinen geschieht mit

```
  CALL OSMARE(N, A, LDA, G, F, Q, WR, WI, MAXSTP, RCOND, IWORK,
 *            AS, GS, FS, RWORK1, RWORK2, TOL, WITHEV, WITHRC,
 *            IWARN, IERR)

  CALL OSMAR2(N, A, LDA, G, F, Q, WR, WI, MAXSTP, RCOND, IWORK,
 *            AS, GS, FS, RWORK1, RWORK2, TOL, WITHEV, WITHRC,
 *            IWARN, IERR)

  CALL OSMARU(N, A, LDA, G, F, X, Y, Z, WR, WI, MAXSTP, IWORK,
 *            AS, GS, FS, RWORK1, RWORK2, TOL, WITHEV, IWARN,
 *            IERR)
```

Dabei muß bei OSMARE, OSMAR2 `Q` bei der Eingabe eine orthogonal symplektische Matrix repräsentieren, entweder die Einheitsmatrix oder frühere Transformationen, entsprechend enthalten bei OSMARU `X,Y,Z` entweder die Aufdatierungen aus früheren Transformationen oder die Initialisierungen $X = Y = 0$, $Z = I_n$.
Falls `WITHEV .EQ. .TRUE.`, dann werden die Eigenwerte mit SQREDE berechnet, sonst müssen `WR,WI` bei der Eingabe die `N` instabilen Eigenwerte von H enthalten. `AS,GS,FS` werden dazu benutzt, in jedem Iterationsschritt die quadrierte Hamiltonische Matrix abzuspeichern, so daß bei einer Deflation in den unteren `(N-P)` $\times$ `(N-P)`–Blöcken von `AS,GS,FS` Elemente einer anderen quadrierten Matrix stehen als im Rest. Daher werden diese Felder nur als Arbeitsspeicher benutzt und haben bei der Ausgabe keine Bedeutung.
In `MAXSTP` wird die maximale Anzahl auszuführender Iterationsschritte übergeben, bei der Ausgabe enthält `MAXSTP` die tatsächliche Anzahl der Iterationen. Dabei ist zu beachten, daß Deflationen nicht als Iterationsschritte gezählt werden, d.h. der Schrittzähler wird nur erhöht, falls OSMSTP, usw., mit dem gleichen `P` wie im letzten Schritt aufgerufen werden.
Falls bei OSMARE, OSMAR2 `WITHRC .EQ. .TRUE.`, dann wird in `RCOND` eine Schätzung für die Kondition von Q_1 aus (5.3) zurückgegeben (siehe XSOLVE).
Es können folgende Fehler und Warnungen auftreten:

`IWARN = 1` : Die geforderte Genauigkeit ($\max_{1\le i,j\le n} |F_{ij}| <$ `TOL`) wird nicht innerhalb von `MAXSTP` Schritten erreicht.
`IWARN = 2` : Beim Aufdatieren nach (3.18) – (3.20) mußte durch $\alpha < 10^{-8}$ dividiert werden (nur bei OSMARU).
`IWARN = 3` : `IWARN = 1` und `IWARN = 2` (nur bei OSMARU).
`IERR = 1` : `N < 1` oder `LDA < N`.
`IERR = 2` : Die als Shifts benutzten Eigenwerte sind nicht abgeschlossen unter komplexer Konjugation.
`IERR = 3` : Mindestens einer der als Shifts benutzten Eigenwerte hat einen negativen Realteil.
`IERR = 4` : Q_1 ist singulär bzgl. der Rechengenauigkeit. (nur bei OSMARE, OSMAR2 mit `WITHRC .EQ. .TRUE.`).
`IERR =-J` : Fehlermeldung von SQREDE bzw. EISPACK Routine HQR.

Da nur Transformationen aus $\mathcal{US}_{2n}$ benutzt werden, kann man die mit OSMARE berechnete Hamiltonische Matrix als Ergebnis eines numerisch rückwärts stark stabilen Algorithmus ansehen. Jedoch treten mitunter durch die

Berechnung der Eigenwerte (auch, wenn statt SQREDE der QR–Algorithmus benutzt wird) und des Multishiftvektors mit Hilfe der quadrierten Hamiltonischen Iterierten so große Rundefehler auf, daß die implizite Defektkorrektur nur langsam bzw. bei zunehmender Ordnung der ARG nicht mehr innerhalb von `N/2` (Standardvorgabe) Schritten konvergiert. Aus denselben Gründen werden z.T. theoretisch vorhandene Deflationen nicht oder erst nach mehreren Iterationsschritten erkannt (siehe Diskussion in Kapitel 4.2 und Beispiel 5.5). Die ausschließliche Verwendung von Givens–Rotationen in OSMAR2 bringt hierbei eine deutliche Verbesserung gegenüber den hauptsächlich auf Householderreduktionen beruhenden Implementierungen OSMARE und OSMARU.

An Speicher benötigt man für alle drei Routinen $\approx 8n^2$–mal DOUBLE PRECISION Speicherplatz, was dem Speicherbedarf für Laub's Schurvektormethode entspricht. Dabei wird $2n^2$ Speicher benötigt, um die von SQREDE überschriebene quadrierte Hamiltonische Matrix abzuspeichern. Alternativ könnte man H^2 auch erneut berechnen, was diesen Speicher einspart. Damit würden $8n^3$ Flops zusätzlich nötig.

Eine Aufwandsschätzung ist schwierig anzugeben, da der Aufwand von der Anzahl der Deflationen und Iterationsschritte abhängt. Berechnet man die Eigenwerte mit SQREDE und ist man nach einem Schritt fertig, so benötigt

- OSMARE $\approx 65n^3$,
- OSMAR2 $\approx 67n^3$,
- OSMARU $\approx 74n^3$

Flops. Pro Iterationsschritt ohne Deflation kommen für

- OSMARE $\approx 38n^3$,
- OSMAR2 $\approx 40n^3$,
- OSMARU $\approx 50n^3$

Flops dazu, d.h. ab dem vierten Iterationsschritt sind diese Routinen i.a. nicht mehr günstiger als die Schurvektormethode von Laub mit Clements/ Glover–Deflation. Tritt eine Deflation ein, so wird die Situation komplizierter, d.h. pro Iterationsschritt kommt der Aufwand aus Abschnitt 5.1.6 hinzu (entspricht obigen Werten für $p = 0$), womit der Vergleich etwas günstiger ausfällt.

5.1.9 DCORC.F

In DCORC wurde der Defektkorrekturalgorithmus 2.35 implementiert.
Zur Lösung der Defektgleichung (2.34) können OSMARE, OSMAR2 oder OSMARU benutzt werden. Der Aufruf geschieht mit

```
  CALL DCORC(N, A, LDA, G, F, X, WR, WI, MAXSTP, ERREST, RCOND,
 *          DX, IWORK, AS, GS, FS, RWORK1, RWORK2, TOL,
 *          METHOD, WITHEV, IWARN, IERR)
```

Mit `METHOD` wird entschieden, welche Methode zur Lösung der Defektgleichung benutzt wird. Dabei gilt

`METHOD < 0` $\longrightarrow$ OSMARE,
`METHOD = 0` $\longrightarrow$ OSMAR2,
`METHOD > 0` $\longrightarrow$ OSMARU.

Die Parameter `RCOND, WITHEV, WITHRC` haben dieselbe Bedeutung wie bei OSMARE, falls `METHOD > 0`, so werden `RCOND, WITHRC` nicht benutzt.

`MAXSTP` gibt die maximale Anzahl auszuführender Defektkorrekturschritte an, bei der Ausgabe enthält dieser Parameter die tatsächliche Anzahl von Iterationen (für `MAXSTP` in OSMARE, OSMAR2, OSMARU wird `N/2` benutzt).

`DX` enthält die Lösung der letzten gelösten Defektgleichung. Dabei ist zu beachten, daß `DX` in DCORC bei Lösung der Defektgleichung als Eingabe für `Q` in OSMARE oder OSMAR2 benutzt wird. Die Lösung der Defektgleichung steht also im Bereich `(N+1,1) -- (2*N,N)` dieses Feldes. In `ERREST` wird die Frobeniusnorm dieser Lösung zurückgegeben, d.h. `DX, ERREST` sind Schätzungen für den absoluten Fehler von `X`, elementweise bzw. in der Frobeniusnorm.

`TOL` definiert die geforderte obere Schranke für `ERREST`, vgl. Algorithmus 2.5.

Folgende Fehler und Warnungen können auftreten:

`IWARN` $\in \{$ `1,2,3` $\}$: Werden von OSMARE, OSMAR2, OSMARU zurückgeliefert, siehe dort.

`IWARN = 4` : `ERREST .LT. TOL` wird nicht innerhalb von `MAXSTP` Schritten erreicht.

`IERR  = 1` : `N < 1` oder `LDA < N` oder die zur Lösung der Defektgleichung benutzte Routine liefert `IERR = 1` zurück.

`IERR` $\in \{$ `2,3,4,-J` $\}$: Werden von OSMARE, OSMAR2, OSMARU zurückgeliefert, siehe dort.

Der gesamte Algorithmus ist numerisch rückwärts stabil, siehe Kapitel 2.5. Der Aufwand pro Schritt beträgt $4n^3$ für die Berechnung der $\tilde{A}, \tilde{F}$ aus Algorithmus 2.35 entsprechenden neuen Matrizen `A,F`. Für die Lösung der Defektgleichung entspricht der Aufwand je nach benutzter Methode den Angaben im letzten Abschnitt, wobei die $32n^3$ Flops für SQREDE entfallen, falls die Eigenwerte schon bekannt sind, also insbesondere, falls OSMARE, OSMAR2 oder OSMARU zur Bestimmung der stabilisierenden Approximation X_0 benutzt wurden.

5.2 Matlab–Routinen

Die MATLAB Funktionen wurden hauptsächlich zur Entwicklung der Algorithmen benutzt und nutzen daher nicht alle Eigenschaften von MATLAB aus, insbesondere wird auch hier bei der Bildung des Multishiftvektors auf komplexe Arithmetik verzichtet. Andererseits sind die Funktionen auch hinsichtlich Effizienz nicht optimiert. Householder–Reduktionen werden z.B. durch Aufruf der eingebauten MATLAB Funktion *qr* erzeugt, die direkt $P = I - \beta vv^T$ zurückgibt. Die Aufdatierungen und Ähnlichkeitstransformationen werden daher statt mit den Formeln aus Kapitel 3.1.1 durch Block–Matrixmultiplikationen ausgeführt, d.h. soll ein Vektor der Länge $n-k$ reduziert werden, so werden diese Operationen analog zu

$$\begin{aligned}
\tilde{Q} &= QH(k,n,v) = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} I_{k-1} & 0 \\ 0 & P \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12}P \\ Q_{21} & Q_{22}P \end{bmatrix}, \\
\tilde{A} &= H(k,n,v)AH(k,n,v) = \begin{bmatrix} I_{k-1} & 0 \\ 0 & P \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} I_{k-1} & 0 \\ 0 & P \end{bmatrix} \\
&= \begin{bmatrix} A_{11} & A_{12}P \\ PA_{21} & PA_{22}P \end{bmatrix}
\end{aligned}$$

durchgeführt.
Die Namensgebung entspricht weitestgehend der der FORTRAN Routinen. Als Implementierungen von Algorithmus 4.6 existieren die Funktionen *osmare, osmare2* und *osmarupd*. Die in MSHIFT und COLRED (COLRD2) durchgeführten Operationen sind hier in den Funktionen *fcolred, fcolred2,* bzw. *fcolrupd* zusammengefaßt, in denen auch die Aufdatierung und Transformation mit der durch die Reduktion des Multishiftvektors bestimmten orthogonal symplektischen Matrix erfolgt. Im Unterschied zu den FORTRAN

Implementierungen wird ein Iterationsschritt mit der ganzen Hamiltonischen Matrix direkt in *osmare*, usw., durchgeführt, während nach einer Deflation alle weiteren Iterationsschritte mit Hilfe der Funktionen *deflat, deflat2* bzw. *deflupd* erfolgen.
Neben den Funktionen, die weitestgehend zu den FORTRAN Routinen analog sind, existieren noch einige Weitere, so z.B. Implementierungen des Newton–Verfahrens (*newtonar*) und der Signumfunktion–Methode (*signf*). Es steht auch eine Funktion *osmarnwt* zur Verfügung, bei der eine mit *osmare, osmare2* oder *osmarupd* ohne implizite Defektkorrektur berechnete Approximation der Lösung mit Hilfe des Newton–Verfahrens verbessert wird.
Eine komplette Liste aller vorhandenen MATLAB Funktionen und deren Zweck wird in den ASCII Dateien *liesmich* bzw. *readme* angegeben. Der Zweck der Funktionen und die Bedeutung der Ein- und Ausgabeparameter können in MATLAB mit Hilfe von

```
help <funktionsname>
```

angezeigt werden.
Alle verfügbaren Löser für die ARG, also *osmare, osmare2, osmarupd, signf, newtonar* und, falls die CONTROL Toolbox [T 2] zur Verfügung steht, *are* (Implementierung von Laub's Schurvektormethode), können mit der MATLAB Skript–Datei *aredemo.m* anhand verschiedener Beispiele getestet und die berechnete Lösung mit expliziter Defektkorrektur (Lösung der Defektgleichung mit *osmare, osmare2* oder *osmarupd*) oder dem Newton–Verfahren verbessert werden.

5.3 Beispiele

In diesem Abschnitt werden die Ergebnisse für die Anwendung des orthogonal symplektischen Multishift Algorithmus auf verschiedene Beispiele vorgestellt. Anhand der ersten sechs Beispiele wurden die drei FORTRAN Routinen OSMARE, OSMAR2 und OSMARU getestet, in Beispiel 5.7 werden die verfügbaren ARG-Löser in MATLAB anhand verschiedener zufällig erzeugter Hamiltonischer Matrizen verglichen. Alle Rechnungen wurden auf Silicon Graphics IRIS Indigo R3000 ($\mathbf{u} \approx 2.22 \cdot 10^{-16}$) am Institut für Geometrie und praktische Mathematik der RWTH Aachen durchgeführt.

Die Beispiele 5.1 – 5.6 können mit Hilfe der FORTRAN Routine MAKEEX erzeugt werden, die mit

```
CALL MAKEEX(NO, SIZE, EPS, A, LDA, G, F, IERR)
```

aufgerufen wird. Dabei gibt `NO` an, welches Beispiel konstruiert werden soll und `SIZE` die Ordnung der ARG (es existieren für alle Beispiele Voreinstellungen, mit denen `SIZE` überschrieben wird, falls unsinnige Werte übergegeben werden). `EPS` gibt bei den ersten drei Beispielen (Beispiele 5.1 – 5.3) den Parameter ε an.
Dabei können folgende Fehler auftreten:
`IERR = 1` : `LDA < 1` oder `SIZE > LDA`.
`IERR = 2` : `NO < 1` oder `NO > 6`.
`IERR = 3` : Bei Beispiel 5.4 (`NO = 4`) ist `SIZE` gerade.

Beispiel 5.1: Dieses Beispiel wird durch MAKEEX mit `N = 1` erzeugt. Man erhält damit folgende Systemmatrizen:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & -2 \end{bmatrix}, \quad G = \begin{bmatrix} \varepsilon^2 & 0 \\ 0 & 0 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

Die exakte Lösung der ARG lautet

$$X = \begin{bmatrix} \frac{1+\sqrt{1+\varepsilon^2}}{\varepsilon^2} & \frac{1}{2+\sqrt{1+\varepsilon^2}} \\ \frac{1}{2+\sqrt{1+\varepsilon^2}} & \frac{1}{4}\left(1 - \frac{\varepsilon^2}{\left(2+\sqrt{1+\varepsilon^2}\right)^2}\right) \end{bmatrix}.$$

Das Problem bei diesem Beispiel liegt darin, daß für $\varepsilon \to 0$ (A, G) nahezu unstabilisierbar wird. Für die Lösung X gilt

$$\lim_{\varepsilon \to 0} x_{11} = \infty.$$

Die relativen Fehler von x_{11}, x_{12}, x_{22} für die mit OSMARE, OSMAR2 und OSMARU berechneten Lösungen können folgender Tabelle entnommen werden. Dabei bezeichnen die Werte in den Klammern wie auch bei den folgenden Ergebnistabellen die Anzahl von impliziten Defektkorrekturschritten.

ε		OSMARE		OSMAR2		OSMARU	
1	x_{11}	0.0		0.0		$5.5 \cdot 10^{-16}$	
	x_{12}	$3.8 \cdot 10^{-16}$	(0)	$3.8 \cdot 10^{-16}$	(0)	0.0	(0)
	x_{22}	$2.4 \cdot 10^{-16}$		$1.2 \cdot 10^{-16}$		$1.3 \cdot 10^{-14}$	
10^{-2}	x_{11}	$6.6 \cdot 10^{-13}$		$6.6 \cdot 10^{-13}$		$6.6 \cdot 10^{-13}$	
	x_{12}	$1.6 \cdot 10^{-12}$	(1)	$1.6 \cdot 10^{-12}$	(1)	$1.6 \cdot 10^{-12}$	(1)
	x_{22}	0.0		$4.4 \cdot 10^{-16}$		$7.3 \cdot 10^{-11}$	
10^{-4}	x_{11}	$1.2 \cdot 10^{-9}$		$1.2 \cdot 10^{-9}$		$1.2 \cdot 10^{-9}$	
	x_{12}	$7.7 \cdot 10^{-9}$	(1)	$7.7 \cdot 10^{-9}$	(1)	$7.7 \cdot 10^{-9}$	(1)
	x_{22}	$1.1 \cdot 10^{-16}$		$2.2 \cdot 10^{-16}$		$1.2 \cdot 10^{-7}$	
10^{-6}	x_{11}	$3.0 \cdot 10^{-5}$		$3.0 \cdot 10^{-5}$		$3.0 \cdot 10^{-5}$	
	x_{12}	$1.9 \cdot 10^{-4}$	(1)	$1.9 \cdot 10^{-4}$	(1)	$1.9 \cdot 10^{-4}$	
	x_{22}	0.0		0.0		$3.0 \cdot 10^{-2}$	

Für die Elemente x_{11}, x_{12} ist das Verhalten aller drei Routinen gleich, d.h. man verliert jeweils $O(\varepsilon^2)$ Stellen an Genauigkeit. Dies entspricht auch der Konditionsschätzung für Q_1, die $O(1/\varepsilon^2)$ ergibt.
Während OSMARE und OSMAR2 x_{22} sehr genau berechnen, ist dies für OSMARU nicht der Fall. Betrachtet man α aus Kapitel 3.4, also den Wert, durch den beim Aufdatieren einer symplektischen Givens–Matrix dividiert wird, so erkennt man, daß $\alpha = O(\varepsilon^2/10)$, was dann auch in x_{22} den erwarteten Genauigkeitsverlust nach sich zieht (vgl. Ende Kapitel 3.4).
Alle berechneten Lösungen konnten mit einem expliziten Defektkorrekturschritt verbessert werden, so daß die relativen Fehler in allen Elementen $O(10^{-16})$ betrugen. □

Beispiel 5.2: Dieses Beispiel wird durch MAKEEX mit `N = 2` erzeugt. Die Systemmatrizen haben folgende Gestalt:

$$A = \begin{bmatrix} -\varepsilon & 1 & 0 & 0 \\ -1 & -\varepsilon & 0 & 0 \\ 0 & 0 & \varepsilon & 1 \\ 0 & 0 & -1 & \varepsilon \end{bmatrix}, \quad G = F = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}.$$

Für $\varepsilon \to 0$ nähert sich ein Paar konjugiert komplexer Eigenwerte der Hamiltonischen Matrix der imaginären Achse und das System wird nahezu unstabilisierbar (bei SQREDE sind die Realteile dieser Eigenwerte ab $\varepsilon = 10^{-7}$ Null bzgl. der Rechengenauigkeit).

Da keine analytische Lösung des Problems bekannt ist, wird als Maß für die Qualität der berechneten Lösungen die Frobeniusnorm des Residuums herangezogen.

ε	OSMARE		OSMAR2		OSMARU	
1	$1.3 \cdot 10^{-14}$	(1)	$2.8 \cdot 10^{-14}$	(1)	$4.4 \cdot 10^{-14}$	(1)
10^{-1}	$2.6 \cdot 10^{-15}$	(1)	$2.0 \cdot 10^{-15}$	(1)	$3.0 \cdot 10^{-15}$	(1)
10^{-2}	$9.4 \cdot 10^{-16}$	(1)	$1.4 \cdot 10^{-15}$	(1)	$1.5 \cdot 10^{-15}$	(1)
10^{-3}	$1.5 \cdot 10^{-15}$	(1)	$1.2 \cdot 10^{-15}$	(1)	$4.4 \cdot 10^{-15}$	(1)
10^{-4}	$3.6 \cdot 10^{-15}$	(2)	$5.7 \cdot 10^{-14}$	(0)	$2.9 \cdot 10^{-15}$	(2)
10^{-5}	$7.9 \cdot 10^{-15}$	(2)	$4.1 \cdot 10^{-14}$	(1)	$7.3 \cdot 10^{-15}$	(2)
10^{-6}	$3.2 \cdot 10^{-14}$	(4)	$6.5 \cdot 10^{-14}$	(3)	$3.1 \cdot 10^{-14}$	(4)
10^{-7}	$1.2 \cdot 10^{-14}$	(19)	$3.5 \cdot 10^{-15}$	(8)	$1.4 \cdot 10^{-14}$	(19)

Man sieht, daß die Anzahl der Iterationsschritte stark ansteigt, wenn sich die Eigenwerte der imaginären Achse nähern. Da die Matrix–Signumfunktion nicht definiert ist, falls Eigenwerte auf der imaginären Achse liegen (siehe Kapitel 2.4), liefert die Signumfunktion–Methode hier wesentlich schlechtere Resultate für $\varepsilon \to 0$. Die Ergebnisse der Schurvektormethode unterscheiden sich von denen in der Tabelle nur geringfügig. □

Beispiel 5.3: Dieses Beispiel wird durch MAKEEX mit `N = 3` erzeugt. Die Systemmatrizen haben folgende Gestalt:

$$
\begin{aligned}
A &= \begin{bmatrix} -0.1 & 0 \\ 0 & -0.02 \end{bmatrix}, \quad F = \begin{bmatrix} 100 & 1000 \\ 1000 & 10000 \end{bmatrix}, \\
G &= \begin{bmatrix} 0.1 & 0 \\ 0.001 & 0.01 \end{bmatrix} \begin{bmatrix} 1+\varepsilon & 1 \\ 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0.1 & 0 \\ 0.001 & 0.01 \end{bmatrix}^{T}.
\end{aligned}
$$

Hier werden für $\varepsilon \to 0$ die Elemente und die Kondition von G groß, die Elemente der Lösung X klein.

Auch hier ist keine analytische Lösung des Problems bekannt. Die Tabelle gibt wieder die Frobeniusnorm der Residuen an.

ε	OSMARE		OSMAR2		OSMARU	
1	$4.9 \cdot 10^{-12}$	(0)	$1.8 \cdot 10^{-12}$	(0)	$9.1 \cdot 10^{-12}$	(0)
10^{-1}	$6.2 \cdot 10^{-10}$	(0)	$1.1 \cdot 10^{-11}$	(0)	$6.3 \cdot 10^{-10}$	(0)
10^{-2}	$2.1 \cdot 10^{-8}$	(0)	$1.7 \cdot 10^{-10}$	(0)	$2.1 \cdot 10^{-8}$	(0)
10^{-3}	$5.3 \cdot 10^{-8}$	(1)	$7.6 \cdot 10^{-8}$	(0)	$5.6 \cdot 10^{-8}$	(1)
10^{-4}	$1.8 \cdot 10^{-6}$	(0)	$8.3 \cdot 10^{-7}$	(0)	$1.8 \cdot 10^{-6}$	(0)
10^{-5}	$1.3 \cdot 10^{-5}$	(1)	$4.2 \cdot 10^{-7}$	(1)	$1.3 \cdot 10^{-5}$	(1)
10^{-6}	$1.4 \cdot 10^{-4}$	(1)	$5.3 \cdot 10^{-7}$	(1)	$1.4 \cdot 10^{-4}$	(1)
10^{-7}	$1.6 \cdot 10^{-3}$	(0)	$1.1 \cdot 10^{-3}$	(0)	$1.5 \cdot 10^{-3}$	(0)

Hier zeigt sich ein leichter Vorteil von OSMAR2 gegenüber den anderen beiden Routinen. Allerdings liefert Laub's Schurvektormethode Residuen, die in der Größenordnung eine (bei $\varepsilon = 10^{-7}$ drei) Zehnerpotenz besser sind als die von OSMAR2.

Die Ergebnisse konnten durch ein bis drei explizite Defektkorrekturschritte verbessert werden, wobei mehr Schritte wegen den in Kapitel 2.5 erwähnten Problemen mit der Auslöschung bei Neuberechnung des Residuums meist keine Verbesserung mehr brachten. Damit konnten mit allen drei Routinen Ergebnisse erzielt werden, die in der Größenordnung denen der Schurvektormethode entsprechen. Dabei wurden die Anfangsnäherung und die Lösung der Defektgleichung jeweils mit derselben Methode bestimmt, die Zahl in Klammern gibt hier die Anzahl der expliziten Defektkorrekturschritte an.

ε	OSMARE		OSMAR2		OSMARU	
1	$3.6 \cdot 10^{-12}$	(1)	$*$		$1.8 \cdot 10^{-12}$	(2)
10^{-1}	$1.8 \cdot 10^{-12}$	(1)	$8.1 \cdot 10^{-13}$	(1)	$1.8 \cdot 10^{-11}$	(1)
10^{-2}	$3.7 \cdot 10^{-12}$	(2)	$4.7 \cdot 10^{-11}$	(1)	$1.1 \cdot 10^{-11}$	(2)
10^{-3}	$2.4 \cdot 10^{-10}$	(1)	$6.4 \cdot 10^{-10}$	(1)	$1.3 \cdot 10^{-9}$	(1)
10^{-4}	$4.7 \cdot 10^{-9}$	(1)	$8.6 \cdot 10^{-9}$	(2)	$4.8 \cdot 10^{-9}$	(2)
10^{-5}	$1.3 \cdot 10^{-8}$	(2)	$6.1 \cdot 10^{-8}$	(1)	$4.0 \cdot 10^{-8}$	(3)
10^{-6}	$1.9 \cdot 10^{-7}$	(1)	$2.2 \cdot 10^{-7}$	(2)	$9.7 \cdot 10^{-7}$	(3)
10^{-7}	$1.4 \cdot 10^{-5}$	(1)	$7.7 \cdot 10^{-7}$	(2)	$4.3 \cdot 10^{-6}$	(2)

$*$ Keine Verbesserung möglich.

□

Bei den bisher betrachteten Beispielen war das Verhalten für OSMARE und OSMAR2 meistens gleich. Dies kann allgemein bei kleinen Ordnungen der zu lösenden ARG beobachtet werden. In den folgenden Beispielen sieht man, wie bei zunehmender Ordnung das Verhalten der Routinen voneinander abweicht.

Beispiel 5.4: (Laub [L 1], Example 4)
Dieses Beispiel wird durch MAKEEX mit `N = 4` erzeugt. Die Systemmatrizen haben folgende Gestalt:

$$
\begin{aligned}
A &= \left[\begin{array}{cccccc}
A_{11} & A_{12} & 0 & \dots & & 0 \\
0 & A_{22} & A_{23} & 0 & \dots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
 & & 0 & A_{N-2,N-2} & A_{N-2,N-1} & 0 \\
 & & & 0 & A_{N-1,N-1} & \begin{array}{c} 0 \\ -1 \end{array} \\
0 & \dots & & & 0 \quad 0 & -1
\end{array}\right] \in \mathbf{R}^{2N-1,2N-1} \\
G &= \operatorname{diag}(1,0,1,0,\dots,1,0,1) \\
F &= \operatorname{diag}(0,10,0,10,\dots,0,10,0).
\end{aligned}
$$

Dabei ist

$$
\begin{aligned}
A_{k,k} &= \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, \quad 1 \le k < N, \\
A_{k+1,k} &= \begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix}, \quad 1 \le k < N-1.
\end{aligned}
$$

Dieses Problem stammt aus einem Steuerungsproblem für Position und Geschwindigkeit einer Kolonne von N Hochgeschwindigkeitsfahrzeugen. Die Ordnung der ARG ist $n = 2N-1$, daher muß beim Aufruf von MAKEEX `SIZE` ungerade sein.
Die Frobeniusnormen der mit den berechneten Lösungen gebildeten Residuen für verschiedene n können der folgenden Tabelle entnommen werden.

n	OSMARE	OSMAR2	OSMARU
5	$3.4 \cdot 10^{-14}$ (1)	$7.3 \cdot 10^{-14}$ (1)	$1.6 \cdot 10^{-14}$ (1)
9	$1.2 \cdot 10^{-13}$ (1)	$7.5 \cdot 10^{-14}$ (1)	$6.9 \cdot 10^{-14}$ (1)
19	$2.5 \cdot 10^{-13}$ (2)	$2.6 \cdot 10^{-13}$ (1)	$3.1 \cdot 10^{-13}$ (2)
25	$*$	$4.8 \cdot 10^{-13}$ (2)	$*$
39	$*$	$2.0 \cdot 10^{-12}$ (7)	$*$

$*$ Keine Konvergenz der impliziten Defektkorrektur.

Hier sieht man sehr deutlich die Probleme der impliziten Defektkorrektur bei Verwendung von Householder–Transformationen. Während bis $n = 19$ das Verhalten aller Routinen vergleichbar dem der Signumfunktion- und Schurvektormethode ist, konnte schon bei $n = 25$ keine Konvergenz für OSMARE, OSMARUPD erzielt werden. Bei OSMAR2 steigt die Anzahl der Iterationen bei großen n stark an, wobei hier alle Iterationsschritte auf der ganzen Matrix durchzuführen sind, so daß die Signumfunktion- und Schurvektormethode bei ähnlichen Ergebnissen wesentlich schneller sind. □

Beispiel 5.5: (Laub [L 1], Example 5)
Dieses Beispiel wird durch MAKEEX mit `N = 5` erzeugt. Dabei wird nur der Fall $n = 64$ in [L 1] behandelt. Das Problem besteht aus der durch die folgenden Matrizen gegebenen ARG:

$$A = \begin{bmatrix} -2 & 1 & 0 & & \dots & 0 & 1 \\ 1 & -2 & 1 & 0 & \dots & & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & & & & & & 1 \\ 1 & 0 & \dots & & 0 & 1 & -2 \end{bmatrix}, \quad G = F = I_n.$$

Die Eigenwerte der daraus resultierenden Hamiltonischen Matrix sind alle reell und von der Größenordnung $O(1)$, wobei fast alle die Vielfachheit 2 besitzen. Daher ist bei Algorithmus 4.5 LISH Deflation zu erwarten. Diese tritt für kleine n auch ein, man kann beobachten, daß ein H–invarianter Teilraum zu allen verschiedenen stabilen Eigenwerten abgespalten wird.
Mit zunehmender Ordnung der ARG wird das über die Abspaltung entscheidende Element $h_{p+1,p}$ aber immer größer, so daß Abspaltung nur noch durch gelockerte Kriterien erreicht werden kann, was aber zu dem in Kapitel 4.2 erläuterten Problem führt, daß der abgespaltene Teilraum entsprechend ungenau ist. Bei diesem Beispiel konnte durch die alleinige Verwendung von Givens–Rotationen in OSMAR2 eine deutliche Verbesserung gegenüber OSMARE erzielt werden.
Die numerischen Ergebnisse können folgender Tabelle entnommen werden, wobei hier wieder die Frobeniusnormen der Residuen für verschiedene n aufgeführt werden.

n	OSMARE		OSMAR2		OSMARU	
5	$3.3 \cdot 10^{-12}$	(2)	$2.1 \cdot 10^{-15}$	(1)	$3.3 \cdot 10^{-12}$	(1)
10	$5.5 \cdot 10^{-8}$	(4)	$3.4 \cdot 10^{-15}$	(1)	$5.5 \cdot 10^{-8}$	(4)
20	$1.8 \cdot 10^{-10}$	(13)	$9.6 \cdot 10^{-15}$	(4)	$1.8 \cdot 10^{-10}$	(13)
30	$2.9 \cdot 10^{-8}$	(17)	$1.8 \cdot 10^{-14}$	(6)	$2.9 \cdot 10^{-8}$	(17)
64	*		$8.7 \cdot 10^{-14}$	(15)	*	

* Keine Konvergenz der impliziten Defektkorrektur.

Man sieht, daß OSMARE und OSMARUPD nur noch mit einer hohen Anzahl an Iterationen konvergieren und die dabei erzielten Ergebnisse trotzdem unbefriedigend sind. Das mit OSMAR2 erzielte Ergebnis für $n = 64$ ist analog zu dem in [L 1] Angegebenen. Da hier schon nach wenigen Iterationen abgespalten wird, ist auch der Geschwindigkeitsverlust gegenüber der Schurvektormethode gering. Bei noch höheren n treten jedoch die oben erwähnten Probleme ein. Während z.B. für $n = 100$ die Signumfunktion- und Schurvektormethode noch immer Residuen der Größenordnung $O(10^{-13})$ liefern, kann mit OSMAR2 nur noch ein Residuum von $O(10^{-10})$ erreicht werden. □

Beispiel 5.6: (Laub [L 1], Example 6)
Dieses Beispiel wird durch MAKEEX mit `N = 6` erzeugt. Die Systemmatrizen haben folgende Gestalt:

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & & 0 & 1 \\ 0 & \dots & & 0 & 0 \end{bmatrix}, \\ G &= \operatorname{diag}(0,\dots,0,1), \quad F = \operatorname{diag}(1,0,\dots,0). \end{aligned}$$

Die Eigenwerte der Hamiltonischen Matrix sind die Wurzeln von

$$\lambda^{2n} = (-1)^{n+1}.$$

Aufgrund theoretischer Überlegungen weiß man, daß $x_{1n} = 1$ sein muß. Bei diesem Beispiel tritt das Problem ein, daß die Lösung von

$$XQ_1 = -Q_2 \tag{5.15}$$

sehr schlecht konditioniert ist. Damit unterliegt die Berechnung des Residuums mit X so großen Rundefehlern, daß man daraus nur wenig an Information über die Güte der Lösung bekommt. Ein besseres und leicht zu überprüfendes Kriterium ist daher die Anzahl korrekter Stellen für das berechnete Element x_{1n}.
Die von DGECO gelieferte Konditionsschätzung für Q_1 ergab bei OSMARE und OSMAR2 $O(10^9)$, was auch der Größnordnung des größten in OSMARU vorkommenden α (siehe Kapitel 3.4) entspricht. Mit

$$\delta_{rel} = |x_{1n} - 1|$$

ergaben die Berechnungen für $n = 21$:

	ohne impl. Def.korr.		mit impl. Def.korr.	
	$\|\text{Res.}\|_F$	δ_{rel}	$\|\text{Res.}\|_F$	δ_{rel}
OSMARE	$2.7 \cdot 10^{-2}$	$6.8 \cdot 10^{-13}$	381 (1)	$2.2 \cdot 10^{-7}$
OSMAR2	$3.0 \cdot 10^{-4}$	$2.4 \cdot 10^{-15}$	244 (1)	$1.4 \cdot 10^{-7}$
OSMARU	0.15	$3.7 \cdot 10^{-13}$	0.17 (1)	$4.8 \cdot 10^{-11}$

Der erwartete Vorteil des Aufdatierungsverfahrens bei diesem Problem zeigt sich nur, falls implizite Defektkorrektur durchgeführt wird. Der damit erzielte Wert für x_{1n} ist um vier Stellen genauer als bei den beiden anderen Routinen sowie der Schurvektor- und Signumfunktion–Methode.
Die Verschlechterung der Ergebnisse bei implizter Defektkorrektur beruht darauf, daß das zur Verbesserung herangezogene Residuum wegen (4.26) und der großen Kondition von Q_1 sehr schlecht ist. Wird der Multishiftalgorithmus nach dem ersten Schritt abgebrochen, so erzielen alle drei Routinen Ergebnisse, die deutlich besser als die von der Schurvektor- oder Signumfunktion–Methode gelieferten sind. Man kann also mit der Kenntnis des Problems, das hier die Berechnung des Residuems schlecht konditioniert ist, sehr gute Ergebnisse erzielen. □

Beispiel 5.7: Hier wurden Hamiltonische Matrizen H verschiedener Ordnung zufällig erzeugt und die Lösung der zugehörigen ARG mit verschiedenen MATLAB Funktionen bestimmt. Die erzeugten Beispiele hatten folgende Eigenschaften, wobei mit Re_{min}, Re_{max} der Betrag des kleinsten und größten Realteils der Eigenwerte von H bezeichnet wird:

	Beispiel Nr.					
	1	2	3	4	5	6
n	10	10	20	20	30	40
$\|H\|_F$	39.9	325.6	147.3	96.3	330.3	577.0
Re_{max}	26.44	216.14	102.3	9.309	229.51	403.21
Re_{min}	0.59	0.803	0.367	0.051	0.835	1.098

Die Beispiele befinden sich im `MATLAB` Verzeichnis auf der beigelegten Diskette als Dateien *example1.mat,...,example6.mat*, wobei die die Hamiltonische Matrix definierenden Blöcke `A,G,F` abgespeichert wurden. Damit ergaben sich für die Frobeniusnormen der Residuen folgende Werte:

	Beispiel Nr.					
	1	2	3	4	5	6
osmare	$7 \cdot 10^{-14}$	$5 \cdot 10^{-10}$	$4 \cdot 10^{-5}$	$2 \cdot 10^{-10}$	$3 \cdot 10^{-6}$	$*$
osmare2	$7 \cdot 10^{-14}$	$3 \cdot 10^{-13}$	$2 \cdot 10^{-13}$	$3 \cdot 10^{-12}$	$7 \cdot 10^{-13}$	$2 \cdot 10^{-12}$
osmarupd	$7 \cdot 10^{-14}$	$5 \cdot 10^{-10}$	$4 \cdot 10^{-5}$	$2 \cdot 10^{-10}$	$3 \cdot 10^{-6}$	$*$
signf	$1 \cdot 10^{-13}$	$5 \cdot 10^{-13}$	$2 \cdot 10^{-13}$	$3 \cdot 10^{-13}$	$9 \cdot 10^{-13}$	$2 \cdot 10^{-12}$
are	$1 \cdot 10^{-13}$	$5 \cdot 10^{-13}$	$2 \cdot 10^{-13}$	$4 \cdot 10^{-12}$	$8 \cdot 10^{-13}$	$2 \cdot 10^{-12}$

$*$ Keine Konvergenz der impliziten Defektkorrektur.

Alle sechs Beispiele haben keine außergewöhnlichen Eigenschaften. Man sieht, daß schon bei kleinen Ordnungen der ARG für *osmare, osmarupd* aufgrund des schlechteren Verhaltens bei der impliziten Defektkorrektur das Residuum groß wird. Beim 2. – 4. Beispiel konnten diese schlechten Residuen mit Hilfe eines expliziten Defektkorrekturschrittes (mit der jeweiligen Funktion auch als Löser für die Defektgleichung) um zwei bis vier Größenordnungen verbessert werden. Für das 4. Beispiel mit OSMARE2 ergab ein expliziter Defektkorrekturschritt ein Residuum von $3 \cdot 10^{-13}$.

Das Verhalten von *osmare2* weist bei diesen Beispielen keine großen Unterschiede zur Signumfunktion– und Schurvektormethode auf. Es konnte beobachtet werden, daß bei allen Beispielen nicht mehr als vier implizite Defektkorrekturschritte und einige Deflationsschritte benötigt wurden, so daß der Aufwand dem für die Schurvektormethode ähnelt.

Die beobachteten Ergebnisse decken sich auch mit anderen gerechneten Zufallsbeispielen. Dabei kann man beobachten, daß für Beispiele bis zur Ordnung 15 das Verhalten von *osmare, osmarupd* noch gut, d.h. gleich dem von *are, signf* und *osmare2* ist. $\square$

Kapitel 6

Schlußfolgerungen und Ausblick

In dieser Arbeit wurde ein orthogonal symplektischer Algorithmus dargestellt, der die stabilisierende Lösung der algebraischen Riccatigleichung bestimmt. Dieser basiert auf dem Algorithmus von Ammar/Mehrmann, mit dem Lagrange–Teilräume Hamiltonischer Matrizen berechnet werden können. Da ausschließlich orthogonal symplektische Transformationen verwendet werden, wird die Hamiltonische Strukur während des gesamten Algorithmus beibehalten. Dabei kann die Lösung der ARG bestimmt werden, indem während der Berechnung des Lagrange–Teilraums alle Transformationen in einer orthogonal symplektischen Matrix akkumuliert werden und anschließend eine lineare Matrixgleichung gelöst wird oder man die Lösung der ARG mit der Methode von Byers/Mehrmann direkt aufdatiert. Bei den meisten berechneten Beispielen sind die aus beiden Vorgehensweisen resultierenden Ergebnisse nahezu identisch, ist die beim Lösen der linearen Matrixgleichung zu invertierende Matrix jedoch schlecht konditioniert, so zeigt sich das direkte Aufdatierungsverfahren überlegen.
Die numerischen Ergebnisse legen es nahe, bei der Berechnung des Lagrange–Teilraums ausschließlich Givens–Rotationen zu verwenden, da sich Householder–Transformationen bei dem zur impliziten Defektkorrektur nötigen Iterationsprozeß als zu rundefehleranfällig erwiesen. Dieser Effekt tritt allerdings erst bei steigender Dimension des zugrundeliegenden Problems zu Tage. Da das direkte Aufdatierungsverfahren für symplektische Givens–Matrizen sehr aufwendig — bei der Reduktion einer Spalte der Hamiltonischen Matrix

$O(n^3)$ gegenüber $O(n^2)$ für Householder–Transformationen — ist, eignet es sich bei ausschließlicher Verwendung von Givens–Rotationen nicht mehr zur Lösung der ARG.

Grundsätzliche Schwierigkeiten bereitet bei diesem Algorithmus das korrekte Erkennen einer Deflation. Da man die Eigenwerte der Hamiltonischen Matrix meist nicht exakt kennt und zudem die Berechnung des Multishiftvektors rundefehleranfällig ist, ist die in jedem Iterationsschritt berechnete erste Spalte der Transformationsmatrix entsprechend ungenau. Daher sind die in Kapitel 4.1 gemachten Annahmen nicht exakt erfüllt und eine theoretisch eintretende Deflation kann in der Praxis schwierig zu erkennen sein, bzw. es sind mehrere Iterationsschritte nötig, bis Deflation eintritt. Verwendet man ein gelockertes Kriterium, so ist der abgespaltene Teilraum und damit auch die berechnete Lösung der ARG entsprechend ungenau.
Für Beispiele, bei denen dieses Problem nicht eintritt, liefert der Multishiftalgorithmus ähnliche Ergebnisse wie die Schurvektormethode nach Laub. Bei „kleinen“ Matrizen ist dieser Algorithmus durch die Ausnutzung der Struktur deutlich schneller als Laub's Verfahren, mit zunehmender Dimension des Problems werden jedoch immer mehr Iterationsschritte für die implizite Defektkorrektur nötig, so daß dieser Vorteil verlorengeht.

Mit Hilfe expliziter Defektkorrektur, wobei zur Lösung der Defektgleichung wieder der Multishiftalgorithmus benutzt wird, kann die berechnete Lösung meist noch verbessert werden. Da man dabei die bereits berechneten Eigenwerte benutzen kann, ist dieses Vorgehen vergleichsweise günstig. Eine Kopplung des Multishiftalgorithmus und iterativer Verfeinerung mit dem Newton–Verfahren liefert i.a. ebenfalls gute Ergebnisse.

Eine Verbesserung des Algorithmus könnte durch eine genauere Berechnung der ersten Spalte der Transformationsmatrix erzielt werden, wobei zu untersuchen wäre, ob der Rundefehlereinfluß bei der Berechnung des Multishiftvektors weiter verringert werden kann. Eine Möglichkeit hier ist das Fallenlassen der Forderung nach reeller Arithmetik, so daß auch bei komplexen Shifts nach jeder Matrix–Vektor–Multiplikation skaliert werden kann. Damit kann man auch das Quadrieren der Hamiltonischen Matrix vermeiden, womit man einen Teil des Speicherbedarfs einspart.
Bei den hier betrachteten Beispielen traten die bei der Verwendung von SQRED zur Berechnung der Eigenwerte nach Kapitel 3.3 zu erwartenden

Probleme nicht in einem solchen Maße ein, daß die Verwendung des langsameren QR–Algorithmus sichtbare Vorteile mit sich bringen würde.

Um diesen Algorithmus zur Lösung allgemeiner algebraischer Riccatigleichungen einzusetzen, wäre es nötig, die sich aus der (numerischen) Deflation ergebenden Probleme zu lösen. Dazu benötigt man zunächst ein besseres Kriterium, um zu entscheiden, wann abgespalten werden soll. Ein gelockertes Kriterium müßte mit einer effizienten Methode gekoppelt werden, die den abzuspaltenden Teilraum verbessert.
Die Ergebnisse bei ausschließlicher Verwendung von Givens–Rotationen einerseits und für das direkte Aufdatierungsverfahren bei schlecht konditionierten Problemen andererseits lassen es erstrebenswert erscheinen, eine effiziente Kopplung beider Vorgehensweisen zu finden.

Abschließend kann man sagen, daß der Multishift Algorithmus in der Version OSMAR2, die ausschließlich Givens–Rotationen verwendet, eine Alternative zu den vorhandenen Algorithmen für die Lösung der algebraischen Riccatigleichung bietet. Können die oben angesprochenen Probleme gelöst und damit auch die bei großen Ordnungen hohe Anzahl an Iterationsschritten verringert werden, so könnte sich dieser Algorithmus den anderen Verfahren sogar als überlegen erweisen.

Literaturverzeichnis

[A 1] G.S. Ammar, V. Mehrmann: *On Hamiltonian and symplectic Hessenberg forms*, Lin. Alg. Appl. 149 (1991), S.55–72.

[B 1] A. Bunse–Gerstner, R. Byers, V. Mehrmann: *Numerical methods for algebraic Riccati equations*, Lecture Notes of the Workshop on "The Riccati Equation in Control, Systems and Signal", S.Bittanti (Hrsg.), Pitagora Editrice Bologna (1989), S.107–115.

[B 2] A. Bunse–Gerstner, R. Byers, V. Mehrmann: *A Chart of Numerical Methods for Structured Eigenvalue Problems*, SIAM J. Matrix Anal. Appl. 13, No.2 (1992), S.419–453.

[B 3] R. Byers: *Hamiltonian and symplectic algorithms for the algebraic Riccati equation*, Ph.D. Thesis Cornell University (Januar 1983).

[B 4] R. Byers: *A Hamiltonian QR Algorithm*, SIAM J. Sci. Stat. Comp. 7 (1986), S. 212–229.

[B 5] R. Byers: *Solving the algebraic Riccati equation with the Matrix Sign Function*, Lin. Alg. Appl. 85 (1987), S.267–279.

[B 6] R. Byers, V. Mehrmann: *Symmetric updating of the solution of the algebraic Riccati equation*, Proc. of X. Symp. on Oper. Res. München, Methods of Operations Research 54 (1986), S.117–125.

[C 1] J.L. Casti: *Linear Dynamical Systems*, Mathematics in Science and Engineering, Academic Press (1987).

[C 2] D. Clements, K. Glover: *Spectral transformations via Hermitian pencils*, Lin. Alg. Appl. 123 (1989), S.797–846.

[D 1] J.W. DEMMEL: *Three methods for refining estimates of invariant subspaces*, Computing 38 (1987), S.42–57.

[D 2] J. DONGARRA, J.R. BUNCH, C. MOLER, G.W. STEWART: *Linpack User's Guide*, SIAM Philadelphia (1979).

[G 1] G.H. GOLUB, C.F. VAN LOAN: *Matrix Computations*, 2nd Ed., The Johns Hopkins University Press, Baltimore, Md (1989).

[K 1] R. KLEINMANN: *On an iterative technique for Riccati equation computations*, IEEE Trans. Autom. Control AC–13 (1968), S.114–115.

[K 2] H.W. KNOBLOCH, H. KWAKERNAAK: *Lineare Kontrolltheorie*, Springer (1985).

[K 3] V. KUĆERA: *A contribution to matrix quadratic equations*, IEEE Trans. Automatic Control (1972), S.344–347.

[L 1] A.J. LAUB: A Schur method for solving algebraic Riccati equations, IEEE Trans. Automatic Control AC–24 (1979), S.912–921.

[L 2] W.W. LIN: *On Schur type decompositions for Hamiltonian and symplectic pencils*, Preprint Institute of Applied Mathematics, National Tsing Hua University, Taiwan (1990).

[N 1] NAG: *Implementation and Documentation Standards for the Subroutine Library in Control and Systems Theory SLICOT*, Numerical Algorithm Group Publication NP2032, Eindhoven/Oxford (1990).

[M 1] V. MEHRMANN: *The Autonomous Linear Quadratic Control Problem: Theory and Numerical Solution*, Lecture Notes in Control and Information Sciences 163, M. Thoma, A. Wyner (Hrsg.), Springer, Berlin (1991).

[M 2] V. MEHRMANN, E. TAN: *Defect correction methods for the solution of algebraic Riccati equations*, IEEE Trans. Automatic Control AC–33 (1988), S.695–698.

[P 1] C. PAIGE, C.F. VAN LOAN: *A Schur decomposition for Hamiltonian matrices*, Lin. Alg. Appl. 41 (1981), S.11–32.

[P 2] B.N. Parlett: *Forward instability of the QR algorithm for symmetric tridiagonal matrices*, Vortrag 4.9.92 an der Fernuniversität Hagen.

[P 3] J.E. Potter: *Matrix quadratic solutions*, SIAM Journal Applied Mathematics 14 (1966), S. 496–500.

[R 1] A.C. Raines, D.S. Watkins: *A Class of Hamiltonian–Symplectic Methods for Solving the Algebraic Riccati Equation*, Washington State University; Pullman, Washington 99164–3113 (1993).

[S 1] H.R. Schwarz: *Numerische Mathematik*, B.G. Teubner, Stuttgart (1986).

[S 2] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, C.B. Moler: *Matrix Eigensystem Routines–EISPACK Guide*, Lecture Notes in Computer Science 6, Springer, Berlin (1976).

[T 1] The Mathworks, Inc.: *386–Matlab User's Guide*, (1990).

[T 2] The Mathworks, Inc.: *Control System Toolbox, User's Guide*, (1990).

[V 1] C.F. Van Loan: *A symplectic method for approximating all the eigenvalues of a Hamiltonian matrix*, Lin. Alg. Appl. 61 (1984), S.233–251.

[W 1] J.H. Wilkinson: *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford (1965).

Erklärung

Ich versichere, daß ich die Arbeit selbstständig verfaßt und keine anderen Hilfsmittel als die angegebenen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quelle gekennzeichnet.

Aachen, den 27. März 1993.