

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT DER TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kardinaal Mercierlaan 94, 3001 Leuven (Heverlee)

FAST AND RELIABLE ALGORITHMS FOR STRUCTURED TOTAL LEAST SQUARES AND RELATED MATRIX PROBLEMS

Jury:
Prof. dr. ir. J. Berlamont, voorzitter
Prof. dr. ir. S. Van Huffel, promotor
Prof. dr. ir. P. Van Dooren, promotor
Prof. dr. D. A. Bini, (Univ. Pisa, Italy)
Prof. dr. ir. A. Bultheel,
Prof. dr. ir. B. De Moor,
Prof. dr. ir. M. Van Barel.

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Nicola MASTRONARDI

U.D.C. 519.654

May 2001

©Katholieke Universiteit Leuven – Faculteit Toegepaste Wetenschappen
Arenbergkasteel, B-3001 Heverlee (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2001/7515/20

ISBN 90-5682-309-4

CONTENTS

SYMBOLS AND ACRONYMS ix

PREFACE xi

1 INTRODUCTION 1
- 1.1 Situation 2
- 1.2 Structured Total Least Squares problems 2
- 1.3 STLS in applications 5
  - 1.3.1 Nuclear Magnetic Resonance spectroscopy 5
  - 1.3.2 Speech compression scheme 6
  - 1.3.3 Deconvolution example in renography 9
- 1.4 Subspace–based system identification problems 11
- 1.5 Estimation of the power spectral density of a widesense stationary random process 13
- 1.6 Brownian motion in Signal Processing, Brownian and diagonal plus semi–separable matrices 14
- 1.7 Chapter by chapter overview 16

2 DISPLACEMENT STRUCTURE AND GENERALIZED SCHUR ALGORITHM 19
- 2.1 Introduction 20
- 2.2 Displacement Theory for symmetric matrices 20
- 2.3 Equivalence between Schur reduction and Gaussian elimination 22
  - 2.3.1 Schur Reduction (Deflation) 22
  - 2.3.2 Close Relation to Gaussian Elimination 23
- 2.4 Generalized Schur Algorithm and ∇_Φ–generators of the Schur complement 24

2.5 Displacement of symmetric Toeplitz and Toeplitz–block matrices 27
2.5.1 Displacement of symmetric Toeplitz matrices 27
2.5.2 Displacement of symmetric Toeplitz–block matrices 28
2.5.3 Displacement of symmetric block–Toeplitz matrices 30
2.6 QR Factorization of Structured Matrices 31
2.7 Conclusion 32

3 GENERALIZED SCHUR ALGORITHM: IMPLEMENTATIONS AND STABILITY 33
3.1 Introduction 34
3.2 Reduction of a generator matrix in proper form 34
3.2.1 Reduction in proper form by means of elementary transformations 35
3.3 Stability analysis 38
3.3.1 Definitions and notations 38
3.3.2 Stability Analysis of GSA 40
3.4 Conclusions 44

4 FAST ALGORITHM FOR SOLVING THE HANKEL/TOEPLITZ STRUCTURED TOTAL LEAST SQUARES PROBLEM 45
4.1 Introduction 46
4.2 Basic algorithms 47
4.3 Fast implementation of STLS2 51
4.3.1 The generalized Schur algorithm applied to M 52
4.3.2 Stability of LDL^T factorization 58
4.3.3 Solution of the linear systems 58
4.4 Application 59
4.4.1 Description 59
4.4.2 Numerical tests 60
4.5 Conclusion 61

5 FAST ALGORITHM FOR THE BASIC DECONVOLUTION PROBLEM 65
5.1 Introduction 66
5.2 The basic deconvolution problem 67
5.3 Park–Rosen–Glick (RPG) algorithm 69
5.4 Fast generalized Schur algorithm 71
5.4.1 Description of the fast generalized Schur algorithm 73
5.4.2 Modified problem 77

5.4.3 Stability of the computation of the $\boldsymbol{R}$ factor 79
5.5 Numerical experiments 79
5.5.1 Examples 79
5.5.2 Efficiency 81
5.5.3 Accuracy 83
5.6 Conclusions 83

6 A FAST ALGORITHM FOR SUBSPACE STATE-SPACE SYSTEM IDENTIFICATION 85
6.1 Introduction 86
6.2 Fast computation of the R factor of the QR factorization of H 87
6.3 The generalized Schur algorithm for rank–deficient matrices 89
6.4 Numerical results 91
6.5 Conclusions 93

7 COMPUTING THE SMALLEST EIGENVALUE OF A SYMMETRIC POSITIVE DEFINITE TOEPLITZ MATRIX 95
7.1 Introduction 96
7.2 Characteristic Polynomial Associated to a Symmetric Positive Definite Toeplitz Matrix 96
7.3 Construction of the Algorithm 97
7.3.1 Newton's Method 98
7.3.2 The Algorithm 99
7.4 Starting Point for the CVN Method 100
7.5 Computing the Eigenvector associated with $\lambda_n(T_n)$ 102
7.6 Numerical Results 102
7.7 Conclusions 103

8 FAST AND STABLE ALGORITHMS FOR DIAGONAL PLUS SEMI–SEPARABLE MATRICES 105
8.1 Introduction 106
8.2 Ch–Gu algorithm 107
8.3 Two–way algorithm 109
8.4 One-way chasing algorithm 111
8.5 Two-way chasing algorithm 113
8.6 Bidiagonalizing diagonal plus unsymmetric semi–separable matrices 117
8.6.1 Transformation of a diagonal plus an unsymmetric semi–separable matrix into an unsymmetric tridiagonal one 118

8.6.2 Transformation of the unsymmetric tridagonal matrix into an upper tridiagonal one 119
8.6.3 Transformation of the upper tridiagonal matrix into an upper bidiagonal one 120
8.7 Conclusions 121

9 CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS 123
9.1 Conclusions 124
9.2 Further research and open problems 124

A ELEMENTARY TRANSFORMATIONS 127
A.1 Elementary Householder Transformations 128
A.2 Elementary Circular or Givens Rotations 128
A.3 Hyperbolic Transformations 129
A.3.1 The H Procedure 131

B MATLAB–LIKE FUNCTIONS OF STLS PROBLEMS 133
B.1 Matlab-like program for the Toeplitz/Hankel STLS problem 134
B.2 Matlab-like program for the basic deconvolution problem 135
B.3 Matlab-like program for the modified basic deconvolution problem 137

C MATLAB–LIKE FUNCTIONS FOR DIAGONAL PLUS SEMI–SEPARABLE MATRICES 141
C.1 Matlab–like function `semisep` 142
C.2 Matlab–like function `semisep2` 142
C.3 Matlab–like function `trid` 144
C.4 Matlab–like function `trid2` 145
C.5 Matlab–like function `tridns` 146
C.6 Matlab–like function `triup` 146
C.7 Matlab–like function `tridbid` 147

BIBLIOGRAPHY 149

SYMBOLS AND ACRONYMS

The current section lists the symbols and acronyms that occur frequently in this thesis. The last column refers to the page on which the acronym is first used and/or defined.

List of symbols

$\mathbb{N}$	The set of natural numbers.
$\mathbb{Z}$	The set of integer numbers.
$\mathbb{R}$	The set of real numbers.
$\mathbb{C}$	The set of complex numbers.
$.^T$	Matrix transposition.
$a \triangleq b$	The quantity a is defined as b.
$\text{col}\{a, b\}$	A column vector with entries a and b.
$\text{diag}\{a, b\}$	A diagonal matrix with diagonal entries a and b.
$a \oplus b$	The same as $\text{diag}\{a, b\}$.
$\mathbf{0}$	A zero scalar, vector, or matrix.
I_n	The identify matrix of size $n \times n$.
$\diamondsuit$	Denotes the end of a proof, an example, or a remark.
$\|\cdot\|_2$	The Euclidean norm of a vector or the maximum singular value of a matrix.
$\|\cdot\|_1$	The sum of the absolute values of the entries of a vector or the maximum absolute column sum of a matrix.
$\|\cdot\|_\infty$	The largest absolute entry of a vector or the maximum absolute row sum of a matrix.
$\|\cdot\|_\mathrm{F}$	The Frobenius norm of a matrix.
$E(x)$	Expected value of x.
i.i.d.	independently identically distributed.
QR	The QR factorization of a matrix.
LDU	The lower-diagonal-upper triangular factorization of a matrix.

$\lvert A \rvert$	A matrix with elements $\lvert a_{ij} \rvert$.
$\lambda_i(A)$	$i-$th eigenvalue of A.
$\sigma_i(A)$	$i-$th singular value of A.
$\kappa(A)$	Condition number of a matrix A, given by $\|A\|_2 \|A^{-1}\|_2$.
$\text{cond}_k(A)$	Equal to $\|A\|_k \|A^{-1}\|_k$.
$\det(A)$	Determinant of A.
ε	Machine precision.
$O(n)$	A constant multiple of n, or of the order of n.

List of acronyms

AR	Auto Regressive	page 6
CB	Code Book	page 6
CTLS	Constrained Total Least Squares	page 4
DSS	Diagonal plus semiseparable	page16
EIV	Errors-In-Variables	page 3
GSA	generalized Schur algorithm	page 2
HF	High-Frequency	page 6
LF	Low-Frequency	page 6
LS	Least Squares	page 3
ML	Maximum Likelihood	page 9
NMR	Nuclear Magnetic Resonance	page 5
RiSVD	Riemannian Singular Value Decomposition	page 4
SPDT	symmetric positive definite Toeplitz	page 96
SVD	Singular Value Decomposition	page 3
STLN	Structured Total Least Norm	page 4
TLS	Total Least Squares	page 3
STLS	Structured Total Least Squares	page 2
VQ	Vector Quantization	page 6

PREFACE

I am deeply grateful to Professor Sabine Van Huffel for providing me with the opportunity to pursue my PhD degree under her supervision, and to be a member of her highly qualified and productive research group. Her generous support, encouragement, patience, and her useful suggestions greatly influenced the contents of this dissertation.

I am very grateful to Professor Paul Van Dooren for serving as my associate supervisor and for spending many hours in useful discussions on relevant topics and for his careful comments and suggestions.

I am also very grateful to Professor Marc Van Barel for many hours of interesting discussions and for carefully reading the manuscript of the thesis.

I would like to thank Professor Bart De Moor for serving on my reading committee despite his very busy schedule.

I would also like to thank Professor Adhemar Bultheel and Professor Dario Bini for being members of the jury. In particular I would like to thank Professor Dario Bini for inviting me to attend the nice workshop on *"Toeplitz matrices"*, organized in Cortona, Professor Vadim Olshevsky for inviting me to attend the nice workshop on *"Structured Matrices in Operator Theory, Numerical Analysis, Control, Signal and Image Processing"*, held in Boulder, and Professor Albrecht Boettcher for inviting me to attend the workshop *"Toeplitz matrices"* in honour of Professor Silbermann, held in Chemnitz. Moreover, I would like to thank my friend Professor Stefano Serra for his encouragement and for inviting me to attend the conference *"Numerical Analysis and Its applications"*, held in Rousse. I would like to thank Professor Daniel Boley and Professor Shivkumar Chandrasekaran for their kind hospitality and for giving me the opportunity to collaborate with them. I have had the honour and the pleasure to have, as colleagues, Dr. Philippe Lemmerling and Dr. Leentje Vanhamme. I greatly appreciated their availability, their encouragement and the helpful discussions on Structured Total Least Squares problems. It's a pity that we gave up playing tennis together. It was a pleasure sharing the office and working together with Professor Vasile Sima and Professor Giansalvo Cirrincione. I also would like to thank Daniel Kressner for collaborating with me.

I would like to express my sincere thanks to the National Research Council

of Italy for supporting me during my first visit in Leuven under the Short Term Mobility Program, to European Community in the last to years on a *"Training and Mobility of Researchers"* project entitled *"Advanced Signal Processing of Medical Magnetic Resonance Imaging and Spectroscopy"* (coordinated by Dr. ir. Dirk van Ormond), ando to the Belgian Programme on Interuniversity Poles of Attraction (project IUAP 4/2 entitled *"Modeling, identification, simulation and control of complex systems"*) for supporting me in the last months.

I would like to thank Leo Kottke, my favourite guitar player, for helping me, by means of his music, to be awake during the nights spent working in ESAT.

Last but not least, I would like to thank my wife Teresa for her endless patience and encouragement. This work would not have been possible without her support.

Chapter 1

INTRODUCTION

1.1 Situation

As the title suggests, this thesis deals with algorithms for different related matrix problems, in particular, algorithms for Structured Total Least Squares (STLS) problems, and for subspace–based system identification problems.

Despite the diversity of these problems, the red wire throughout them is the *structure* of the involved matrices. Indeed, the matrices involved have a Toeplitz/Hankel structure. However, in many applications, the size of the problems can be very large and, moreover, these problems need to be solved over and over again so that they can hardly be handled by standard numerical techniques. Fast and reliable algorithms for solving these problems can be based on the same *kernel* algorithm: the "classical" (generalized) Schur algorithm. Then a new problem arises. How can we implement the generalized Schur algorithm (GSA) in an efficient and reliable way for the above-mentioned problems? The thesis gives a positive answer for most considered problems.

The last two chapters of the thesis are devoted to two other *structured* problems: the computation of the smallest eigenpair of a symmetric positive definite Toeplitz matrix and some basic fast and stable linear system and eigenvalue solvers for diagonal plus semi–separable matrices.

The aim of this introductory chapter is to describe the features and applications of the main topics considered in the thesis. In particular, § 1.2 introduces the Structured Total Least Squares problem, followed by a description of three STLS applications in § 1.3. A short description of a computational kernel problem arising in subspace system identification can be found in § 1.4. In § 1.5 and § 1.6 two signal processing applications are considered where the computation of the smallest eigenvalue of symmetric positive definite Toeplitz matrices and the solution of linear systems involving diagonal plus semi–separable matrices is required.

Finally we present a detailed chapter–by–chapter overview where the main contributions of this thesis are printed out, together with some references to the literature where these results have appeared.

1.2 Structured Total Least Squares problems

Linear models play an important role in a broad class of disciplines and related applications such as signal processing, control theory, system theory, statistics, economics, biomedical applications, etc. These models can be described in the following way

$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n = \beta \tag{1.2.1}$$

with $x \in \mathbb{R}^{n\times 1}$ the parameter vector, $\alpha \in \mathbb{R}^{n\times 1}$ and $\beta \in \mathbb{R}$ the observed variables. In an experiment, typically many measurements are performed, say m, all of which are supposed to satisfy relation (1.2.1). This leads to an overdetermined set of

equations

$$\sum_{j=1}^{n} a_{ij}x_j \approx b_i,\ i = 1, \ldots, m$$

with $A \triangleq (a_{ij}) \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^{m \times 1}$ and $m > n$ or in matrix form:

$$Ax \approx b. \tag{1.2.2}$$

The first methods for solving this type of problems date back to the beginning of the 19th century, when Gauss [49] described the well-known Least Squares (LS) method. In the LS approach a solution for the overdetermined system of equations (1.2.2) is found by solving the following unconstrained optimization problem:

$$\min_x \|Ax - b\|_2.$$

In many applications not only the observed variables $b_i, i = 1, \ldots, m$, but also the observed variables $a_{ij}, i = 1, \ldots, m,\ j = 1, \ldots n$ are affected by errors. The latter observation led to an approach known as Total Least Squares (TLS) in the field of numerical analysis, or also as Errors-In-Variables (EIV) regression or orthogonal regression in statistics. The TLS approach allows errors in the entries of b and in the entries of A as well. It can be formulated as the following constrained optimization problem:

$$\begin{aligned} &\min_{\Delta A, \Delta b, x} \|[\Delta A\ \Delta b]\|_F \\ &\text{such that } (A + \Delta A)x = b + \Delta b. \end{aligned} \tag{1.2.3}$$

The discovery of the link between TLS and the Singular Value Decomposition (SVD) [52], [129] increased the insight in TLS problems and also led to robust SVD-based algorithms, thereby increasing the popularity of TLS in real-life applications. The algebraic, algorithmic and geometrical properties of the TLS problem as well as extensions towards so-called non-generic TLS problems were extensively studied in [129].

One of those properties is the fact that the TLS approach yields a Maximum Likelihood (ML) estimate for the parameter vector under certain assumptions. In order to specify these assumptions, a statistical framework is now presented. Let A and b contain measurements of the true unobservable values in $A_0 \in \mathbb{R}^{m \times n}$ and $b_0 \in \mathbb{R}^{m \times 1}$. Furthermore let

$$\begin{aligned} A_0 x_0 &= b_0 \\ A &= A_0 + \Delta A_0 \\ b &= b_0 + \Delta b_0, \end{aligned}$$

in which $\Delta A_0 \in \mathbb{R}^{m \times n}$ and $\Delta b_0 \in \mathbb{R}^{m \times 1}$ contain the "measurement" errors on the elements of A_0 and b_0. In [129] it is then shown that the solution x of (1.2.3) is

a ML estimator for x_0 when the rows of $[\Delta A_0 \ \Delta b_0]$ are independently identically distributed (i.i.d.) with common zero mean vector and common covariance matrix that is (an unknown) multiple of the identity matrix $I_{(n+1)\times(n+1)}$. In reality, these noise conditions are often violated when there exist linear relations between the true unobserved variables represented by the entries $[A_0 \ b_0]$ as well as between the observed variables represented by the entries of $[A \ b]$. Mostly the linear relations between the entries of $[A_0 \ b_0]$ are the same as those between the entries of $[A \ b]$. Thus, in case the *different* entries of $[\Delta A_0 \ \Delta b_0]$ are i.i.d. zero mean noise of equal (unknown) variance, a ML estimate of x can only be obtained when the entries of $[\Delta A \ \Delta b]$ in (1.2.3) obey the same linear relations as the entries of $[A \ b]$ and the entries of $[A + \Delta A \ b + \Delta b]$ (for a proof see [3]).

An example of such linear relations occurs when elements of a signal vector s are stored in a Toeplitz matrix $[A \ b]$ (such that the signal can be read from the first row and the first column),

$$[A \ b] = \begin{bmatrix} s_{n+1} & s_n & \cdots & s_2 & s_1 \\ s_{n+2} & s_{n+1} & \ddots & \ddots & s_2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ s_{m+n} & s_{m+n-1} & \cdots & s_{m+1} & s_m \end{bmatrix}.$$

The linear relations are a mathematical way of expressing that all the elements on a diagonal are equal, since they represent one and the same element of the signal vector s. In matrix terms these linear relations can be represented by a matrix structure constraint. Adding the latter matrix structure constraint yields the Structured Total Least Squares (STLS) problem:

$$\begin{aligned} &\min_{\Delta A, \Delta b, x} \|[\Delta A \ \Delta b]\|_X^2 \\ &\text{such that } (A + \Delta A)x = b + \Delta b, \\ &\text{and } [\Delta A \ \Delta b] \text{ has the same linear structure as } [A \ b], \end{aligned}$$

where $\|.\|_X$ is an appropriate 2-norm defined on the entries of $[\Delta A \ \Delta b]$. An overview of the most important existing methods for solving STLS problems, the Constrained Total Least Squares (CTLS) approach [1], [3], the Riemannian SVD (RiSVD) approach [36], [37] and the Structured Total Least Norm (STLN) approach [106], [107], [127], can be found in [79]. The algebraic and geometric properties of STLS problems are investigated in [79] as well.

STLS problems arise in many applications. In Ch. 4, a recently introduced speech compression scheme in which the considered STLS problem constitutes one of the kernel problems is handled. A medical application in renography is considered as a STLS problem and is dealt with in Ch. 5. A large list of applications can be found in [79].

In the next section three applications are described.

1.3 STLS in applications

To give an idea of the large amount of fields and related applications in which the STLS problem constitutes the kernel problem, three examples are described in some detail in the next three subsections.

1.3.1 Nuclear Magnetic Resonance spectroscopy

The first example concerns a medical application in Nuclear Magnetic Resonance (NMR) spectroscopy. NMR spectroscopy is a frequently used non-invasive medical diagnostic tool. Somewhat simplified (for a comprehensive treatment of NMR spectroscopy see [34], [47]), the basic principle of NMR spectroscopy can be explained as follows: atomic nuclei in a static magnetic field can be in 2 different spin or energy states. Application of an oscillating magnetic field causes the transition of nuclei to the high-energy state. After removing the oscillating magnetic field, the population of nuclei returns to its equilibrium state, thereby emitting the previously absorbed energy as weak electromagnetic radiation. The latter signal can be modeled (in the time-domain) as a sum of complex damped exponentials perturbed by i.i.d. complex Gaussian noise:

$$s_l = \sum_{k=1}^{K} a_k e^{jp_k} e^{(j2\pi f_k + d_k)l\Delta t} + v_l, \; l = 1, \ldots, N, \tag{1.3.4}$$

with a_k the amplitude, f_k the frequency, p_k the phase and d_k the damping of the kth component, Δt the sampling interval and $v \in \mathbb{C}^{N\times 1}$ a vector of zero-mean i.i.d. Gaussian noise samples. Every spectral component or term in (1.3.4) corresponds to a different chemical substance or metabolite present in the volume element from which the signal was obtained. The amplitude a_k is the most important parameter since it is directly proportional to the concentration of the corresponding metabolite (i.e. the metabolite associated to the resonance frequency f_k) in the volume element under investigation. Therefore, if signals are obtained from several adjacent volume elements in a cross section of e.g. a patient's head, it is possible to produce an image that visualizes the concentration of a specific metabolite over the cross section. As shown in [79], a signal consisting of a sum of K complex damped exponentials, satisfies a linear prediction equation of order K. Writing out the equations for the different time instants we obtain the following system of equations:

$$Ax \approx b \tag{1.3.5}$$

$$A = \begin{bmatrix} s_1 & s_2 & \cdots & s_K \\ s_2 & s_3 & \cdots & s_{K+1} \\ \vdots & \vdots & & \vdots \\ s_{N-K} & s_{N-K+1} & \cdots & s_{N-1} \end{bmatrix}, b = \begin{bmatrix} s_{K+1} \\ s_{K+2} \\ \vdots \\ s_N \end{bmatrix},$$

which is not compatible because the signal s is a sum of complex damped exponentials perturbed by i.i.d. additive complex Gaussian noise. In order to remove

the noise and make (1.3.5) compatible, it is intuitively obvious (for a more detailed explanation see [34]) that the correction $[\Delta A \ \Delta b]$ applied to $[A \ b]$ should have the same Hankel structure. If not, it could mean e.g. that the entries on the fourth anti-diagonal of $[\Delta A \ \Delta b]$ are different although they all represent the "perturbation" to one and the same element s_4. Due to the i.i.d. complex Gaussian noise, a ML estimate of x (and therefore also ML estimates of the parameters $a_k, p_k, f_k, d_k, k = 1, \ldots, K$, since they can be determined from x and s in a straightforward way, see e.g. [128]) is obtained by solving the following problem:

$$\begin{aligned} &\min_{\Delta s} \Delta s^H \Delta s \\ &\text{such that } (A + \Delta A)x = b + \Delta b \\ &\text{and } [A + \Delta A \ b + \Delta b] \text{ has the same Hankel structure as } [A \ b]. \end{aligned}$$

where $\Delta s_l, \ l = 1, \ldots N$ represents the correction to $s_l, l = 1, \ldots, N$. Clearly, the latter kernel problem in NMR spectroscopy is an STLS problem and can be solved using the STLS algorithm described in Ch. 4, thereby improving the speed up of STLS based NMR data quantification algorithms [25] with a factor 300 (N=256) to 1000 (N=1024).

1.3.2 Speech compression scheme

We describe the new subband based coding scheme proposed in [76], [77], [79]. In a first step a frame of the incoming signal is fed to a lowpass filter. This yields the low-frequency (LF) part. By subtracting the latter from the incoming signal, we obtain the high-frequency (HF), non-smoothed part. At the sender side, the high-frequency part of the original signal is also processed on a frame–by–frame basis (N_{HF} samples per frame). First of all the HF part of the incoming frame is preemphasized in order to put more stress on the highest frequencies in this band. Each frame is encoded using a straightforward waveform vector quantization (VQ) algorithm, based on a Code Book (CB), generated by the well known LBG algorithm [50]. The codevectors in the CB are w bits long, leading to 2^w different codevectors. At the receiver side decoding is performed by selection of the appropriate waveform from the CB in order to reduce the quantization noise of the resulting signal q (see [79] for the details).

The speech signal is split into two bands because the LF part of a speech signal can efficiently be modeled by means of a linear model (Auto Regressive (AR) model). Let s denote the downsampled output of the lowpass filter. The classical AR approach applied to frame i can then be written as follows:

$$\min_{\substack{e_j, \ j = 1 + (i-1)N_{LFD}, \ldots, iN_{LFD}, \\ x_l, \ l = 1, \ldots, n}} \sum_{j=1+(i-1)N_{LFD}}^{iN_{LFD}} (e_j)^2 \qquad (1.3.6)$$

$$\text{such that } s_l = \sum_{k=1}^{n} x_k s_{l-k} + e_l,$$
$$l = n+1+(i-1)N_{LFD}, \ldots, iN_{LFD},$$

with N_{LFD} the number of samples per downsampled frame , i the frame number, $x \in \mathbb{R}^{n\times 1}$ the vector of model parameters of frame i and n the model order.
If we use the convention that the vector $s(1+(i-1)N_{LFD} : iN_{LFD})$ can be read from the first column and the last row of the Hankel matrix $S = [A\ b] \in \mathbb{R}^{(N_{LFD}-n)\times(n+1)}$, by starting in the upper left corner and ending in the lower right corner, the constraints of (1.3.6) can be written as

$$Ax = b + e. \tag{1.3.7}$$

Together with the objective function in (1.3.6), we see that (1.3.7) defines a Least Squares (LS) problem for solving the incompatible set of equations

$$Ax \approx b.$$

Using this LS approach it is not sufficient to transmit the vector x containing the model parameters together with the first n samples of $s(1+(i-1)N_{LFD} : iN_{LFD})$, in order to reconstruct the signal. At the receiver side, the knowledge of the residual vector e of the corresponding frame is needed. This of course limits the attainable compression ratio, since also the residual vector e needs to be quantized and transmitted.
The approach of the LF part proposed in [76], [77], [79] is different. The problem formulation to solve for the ith frame, is the following:

$$\min_{\substack{\Delta s_j,\ j=1+(i-1)N_{LFD},\ldots,iN_{LFD},\\ x_k,\ k=1,\ldots,n}} \sum_{j=1+(i-1)N_{LFD}}^{iN_{LFD}} (\Delta s_j)^2 \tag{1.3.8}$$
$$\text{such that } s_l + \Delta s_l = \sum_{k=1}^{n} x_k (s_{l-k} + \Delta s_{l-k}),$$
$$l = n+1+(i-1)N_{LFD}, \ldots, iN_{LFD}.$$

If we use the convention that the vector $\Delta s(1+(i-1)N_{LFD} : iN_{LFD})$ can be read from the first column and the last row of the Hankel matrix $\Delta S = [\Delta A\ \Delta b] \in \mathbb{R}^{(N_{LFD}-n)\times(n+1)}$, by starting in the upper left corner and ending in the lower right corner, (1.3.8) can be reformulated as follows:

$$\min_{\substack{\Delta s_j,\ j=1+(i-1)N_{LFD},\ldots,iN_{LFD},\\ x_k, k=1,\ldots,n}} \sum_{j=1+(i-1)N_{LFD}}^{iN_{LFD}} (\Delta s_j)^2 \tag{1.3.9}$$

such that $(A + \Delta A)x = b + \Delta b$

and $\Delta S \triangleq [\Delta A\ \Delta b]$ has same structure as $S \triangleq [A\ b]$.

The latter clearly defines an STLS problem. Moreover, since it is an STLS problem in which the Hankel structure of the extended data matrix needs to be preserved, that problem (1.3.9) can be reparametrized as follows [79]:

$$\min_{\substack{\Delta s_j,\ j = 1 + (i-1)N_{LFD}, \ldots, iN_{LFD}, \\ x_k,\ k = 1, \ldots, n}} \sum_{j=1+(i-1)N_{LFD}}^{iN_{LFD}} (\Delta s_j)^2 \qquad (1.3.10)$$

$$\text{such that } s_{l+(i-1)N_{LFD}} + \Delta s_{l+(i-1)N_{LFD}} = \sum_{k=1}^{n} c_k z_k^l,$$

$$l = 1, \ldots, N_{LFD}$$

with $c_k = a_k e^{jp_k}$ the complex amplitude of the kth component and $z_k = e^{j2\pi f_k + d_k}$ the complex pole of the kth component. Furthermore we have that a_k is the amplitude, p_k the phase, d_k the damping and f_k the frequency of the kth component. Note that by using the above signal model, we exclude the non-generic case in which 2 or more signal poles would coincide (this would lead to additional polynomial terms in t, see [104]). Since only real data are considered, the signal poles

$$e^{d_k + j2\pi f_k}, k = 1, \ldots, n$$

appear in complex conjugate pairs and the same holds for the complex amplitudes

$$a_k e^{jp_k}, k = 1, \ldots, n$$

The latter means that when n is odd, there will at least be 1 component k such that $f_k = 0$ and $p_k = 0$ (we say "at least" since there can be an even number of real exponentials when n is even and an odd number of real exponentials when n is odd). As a result, the sum of complex damped exponentials in (1.3.10) becomes a sum of exponentially damped sinusoids.
The original N_{LFD} speech samples are now compressed to $2n$ samples $s_l + \Delta s_l$, $l = 1, \ldots, n$ and x_k, $k = 1, \ldots, n$. Sending these samples allows to reconstruct at receiver's side the entire LFD speech frame by making use of the constraint in (1.3.8). However, to avoid any risk for error propagation especially for large frames, it is numerically better to use the parametrized form and represent the N_{LFD} samples by the $2n$ parameters a, p_k, d_k, f_k, $k = 1, \ldots, n/2$ (the remaining parameters are known either by complex conjugation or by equality, as mentioned above). In the latter case, the LFD speech frame is reconstructed at receiver's side by making use of the constraint in (1.3.10).

Summarizing we can say that the particular STLS problem that needs to be solved for each frame of the LF signal corresponds to finding the sum of n complex damped exponentials that fits as close (in L_2 norm) as possible the LF frame. In this thesis (see chapter 4) fast algorithms are presented for solving the STLS problems (1.3.9) and (1.3.10) which speed up the computational efficiency of the classical STLS algorithms, as outlined in [79], with a factor proportional to the number of rows of the involved Hankel matrix, thereby enabling an online implementation of the proposed speech compression scheme.

In this sense the LF part of this scheme is related to other sinusoidal coders, the main difference being that in this scheme damped sinusoids are employed instead of undamped sinusoids. The fact that classical sinusoidal coders only use undamped sinusoids obviously implies that the number of sinusoids needed to fit 1 frame in a satisfactory way is quite high compared to fitting the same frame with damped sinusoids. For this reason classical sinusoidal coders use heuristic methods to reduce the number of parameters, such as e.g. the fitting of an envelope that passes through the obtained amplitudes (see e.g. spectral envelope estimation vocoder (SEEVOC) [100]). Other implementations try to exploit the properties of the speech production mechanism even more (see e.g. the sinusoidal transform coder (STC) [69]).

1.3.3 Deconvolution example in renography

Deconvolution problems occur in many areas such as reflection seismology, telecommunications, medical applications [19], [101], [7], [57],... to name just a few. The basic deconvolution problem is depicted in Fig. 1.1, where u_k represents the input and y_k the output at time k; Δu_k and Δy_k are i.i.d. white Gaussian measurement noise added respectively to the input and to the output. The system, represented by its transfer function $X(z)$, is a linear time-invariant system with impulse response $x \in \mathbb{R}^{n\times 1}$. The basic deconvolution problem can now be formulated as follows:

> Given the measurements $u_k + \Delta u_k, k = 1, \ldots, m$ and $y_k + \Delta y_k$, $k = 1, \ldots, m$, of a system, find a Maximum Likelihood (ML) estimate for the system impulse response $x_i, i = 1, \ldots, n$.

Starting from this problem formulation of the basic deconvolution problem, it is straightforward to show that a ML estimate can be found as the solution of the following problem (for a proof, see [3]):

$$\begin{aligned} &\min_{E,x} \alpha^T\alpha + \beta^T\beta \\ &\text{s.t. } (A+E)x = y + \beta, \end{aligned} \tag{1.3.11}$$

with

$$A = \begin{bmatrix} u_n & u_{n-1} & \cdots & u_1 \\ u_{n+1} & u_n & \cdots & u_2 \\ \vdots & \ddots & & \vdots \\ u_{m+n-1} & u_{m+n-2} & \cdots & u_m \end{bmatrix} \in \mathbb{R}^{m\times n},$$

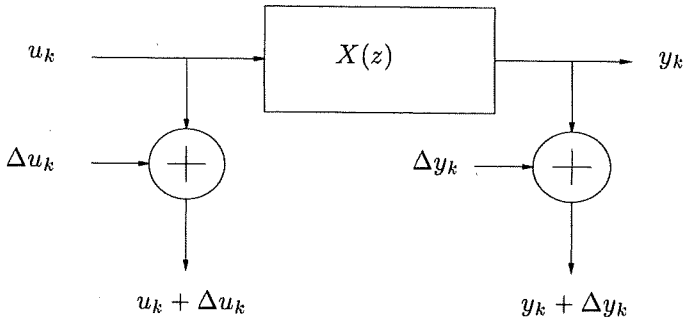


Figure 1.1. This figure shows a schematic outline of the basic deconvolution problem. The goal is to estimate the impulse response x starting from the noisy input $(u + \Delta u)$ and the noisy output $(y + \Delta y)$.

$$E = \begin{bmatrix} \alpha_n & \alpha_{n-1} & \dots & \alpha_1 \\ \alpha_{n+1} & \alpha_n & \dots & \alpha_2 \\ \vdots & \ddots & & \vdots \\ \alpha_{m+n-1} & \alpha_{m+n-2} & \dots & \alpha_m \end{bmatrix} \in \mathbb{R}^{m \times n},$$

$$\beta = (A + E)x - y \in \mathbb{R}^{m \times 1},$$

with E the correction applied to A, β the correction applied to y, $y \in \mathbb{R}^{m \times 1}$ the output, and $x \in \mathbb{R}^{n \times 1}$ the impulse response. Problem (1.3.11) is an STLS problem, since corrections can be applied to the left-hand side matrix A of the constraints in (1.3.11) (implying that it is a *total* LS type problem) and in addition the corresponding correction matrix E is structured (implying that we have to deal with a *structured* TLS problem).

As an example we consider a deconvolution problem that occurs in renography [44]. The goal is to determine via deconvolution the so-called renal retention function of the kidney, which in system theoretic terms corresponds to the impulse response x of the system in Fig. 1.1. This retention function visualizes the mean whole kidney transit time of one unit of a tracer injected into the patient, and it enables a physician to evaluate the renal function and renal dysfunction severity after transplantation. In order to obtain this impulse response, the following experiment is conducted. A radioactive tracer is injected in an artery of the patient (see Fig.1.2).

The measured input of the system ($u + \Delta u$ in Fig. 1.1) is the by noise perturbed arterial concentration of the radioactive tracer as a function of time. This concentration is measured by means of a gamma camera, and thus in discretized time $u_k + \Delta u_k$ represents the number of counts registered in the vascular region at the entrance of the kidney under study during the kth sampling interval. The measured

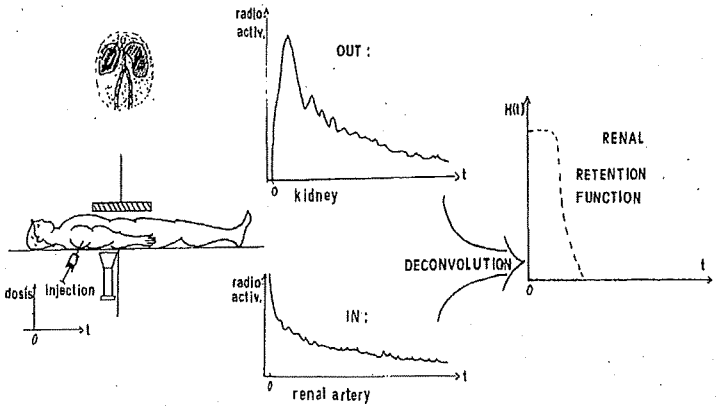


Figure 1.2. After the injection of a radioactive tracer into an antecubital vein of the patient, the radioactive counts in each kidney, as well as in the renal artery, are enregistered during a certain time interval (20–30 minutes) and visualized in an activity–time curve. Using these samples, the desired renal retention function is computed by deconvolution.

output $y_k + \Delta y_k$ (the so-called renogram) represents the (by noise perturbed) number of counts registered in the whole kidney region by the gamma camera in the kth sampling interval. Deconvolution analysis of the renogram is based on modeling the kidney as a linear time-invariant causal system with zero-initial state.

Fast algorithms for solving the basic deconvolution problem (1.3.11) are presented in Ch. 5 and are shown to speed up the computational efficiency of the classical STLS algorithms, as outlined in [106], with a factor of 10 for small sized data sets as occuring in renography and, in general, with a factor proportional to the size m of the problem. Moreover, the statistical accuracy of the STLS estimator also clearly increases compared to the biased TLS estimator (the errors are typically 10% lower in the renal deconvolution problem).

1.4 Subspace–based system identification problems

Subspace based system identification has become very popular in the last decade [39]. The success of this state–space identification approach is mainly due to the fact that it relies on a simple matrix decomposition for which reliable numerical algorithms are available. Its major drawback, on the other hand, is that large "data" matrices are involved and that this may lead to high computing and storage

costs. We now briefly recall the basic formulation of the problem. An extensive treatment of the subject can be found in [132].

A discrete time, linear, time–invariant state space model can be described as

$$x_{k+1} = Ax_k + Bu_k + w_k, \tag{1.4.12}$$

$$y_k = Cx_k + Du_k + v_k, \tag{1.4.13}$$

with

$$E\left(\begin{bmatrix} w_p \\ v_p \end{bmatrix} \begin{bmatrix} w_p^T & v_p^T \end{bmatrix}\right) = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \delta_{pq} \geq 0. \tag{1.4.14}$$

The vectors $u_k \in \mathbb{R}^{m\times 1}$ and $y_k \in \mathbb{R}^{l\times 1}$ are the measurements at time instant k of respectively the m inputs and l outputs of the process. The vector x_k is the state vector of the process at discrete time instant k, $v_k \in \mathbb{R}^{l\times 1}$ and $w_k \in \mathbb{R}^{n\times 1}$ are unobserved vector signals. v_k is called the measurement noise and w_k is called the process noise. It is assumed that they are zero mean, stationary white noise vectors sequences. The matrices $Q \in \mathbb{R}^{n\times n}$, $S \in \mathbb{R}^{n\times l}$ and $R \in \mathbb{R}^{l\times l}$ are the covariance matrices of the noise sequences w_k and v_k. $E(\cdot)$ is the expected value. In addition, the following orthogonality conditions are required to be satisfied,

$$E\left(\begin{bmatrix} x_k \\ u_k \end{bmatrix} \begin{bmatrix} w_k^T & v_k^T \end{bmatrix}\right) = 0. \tag{1.4.15}$$

We are now ready to state the main time domain subspace identification problem:

Given a large number N of measurements of the input $u_k \in \mathbb{R}^m$ and the output $y_k \in \mathbb{R}^l$ generated by the unknown systems (1.4.12–1.4.13–1.4.14). Determine, using only the inputs u_k and outputs y_k the order n of the unknown system, the system matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{l\times n}$, $D \in \mathbb{R}^{l\times m}$, up to a similarity transformation and the matrices $Q \in \mathbb{R}^{n\times n}$, $S \in \mathbb{R}^{n\times l}$, $R \in \mathbb{R}^{l\times l}$.

One of the important conceptual ideas behind time domain subspace algorithms is to re-introduce the concept of the state of a dynamical system within the system identification context. In contrast to "classical" identification algorithms, subspace algorithms first estimate/calculate the state (sequence) (implicitly or explicitly), while next the (state space) model is determined.

By choosing $s \ll \frac{N}{2(m+l)}$ the $N \times 2(m+l)s$ matrix $H = [U_{2s,N}^T Y_{2s,N}^T]$, where $U_{2s,N}$ and $Y_{2s,N}$ are block–Hankel matrices defined in terms of the input and output data, respectively, is constructed :

$$U_{2s,N} = \begin{bmatrix} u_1 & u_2 & u_3 & \dots & u_N \\ u_2 & u_3 & u_4 & \dots & u_{N+1} \\ u_3 & u_4 & u_5 & \dots & u_{N+2} \\ \vdots & \vdots & \vdots & & \vdots \\ u_{2s} & u_{2s+1} & u_{2s+2} & \dots & u_{N+2s-1} \end{bmatrix},$$

$$Y_{2s,N} = \begin{bmatrix} y_1 & y_2 & y_3 & \cdots & y_N \\ y_2 & y_3 & y_4 & \cdots & y_{N+1} \\ y_3 & y_4 & y_5 & \cdots & y_{N+2} \\ \vdots & \vdots & \vdots & & \vdots \\ y_{2s} & y_{2s+1} & y_{2s+2} & \cdots & y_{N+2s-1} \end{bmatrix},$$

It can be seen [132] that the knowledge of the $\hat{R}$ factor of the $\hat{Q}\hat{R}$ factorization of H directly leads to estimates of the state–space matrices A, B, C, D, Q, R and S. Hence the first and most time consuming step of subspace identification algorithms always consists of computing the $\hat{R}$ factor of the $\hat{Q}\hat{R}$ factorization of the block–Hankel-block matrix H. In Ch. 6, a fast and reliable algorithm is derived for computing the $\hat{R}$ factor, which reduces the computation time with a factor varying from 10 up to 100 in a variety of benchmark problems of size varying from 1000 up to 8000.

1.5 Estimation of the power spectral density of a widesense stationary random process

The estimation of the power spectral density of a widesense stationary random process is an important problem with applications in different fields [68], [55]. There are many different techniques available for power spectrum estimation and these techniques vary in the type of *a priori* information available about the random process and in the model assumed for the generation of the process. A parametric technique developed by V.F. Pisarenko assumes that the random process may be modeled as a sum of sinusoids in white noise [108], [109]

$$x_n = \sum_{l=1}^{L} A_l \cos(n\omega_l + \theta_l) + w_n \tag{1.5.16}$$

where θ_l are random variables uniformly distributed over the interval $(0, 2\pi)$ and w_n is a zero mean white noise process with variance σ^2. The problem addressed by Pisarenko [109] concernes the estimation of the amplitudes and frequencies of the sinusoids, A_l and ω_l, for $l = 1, \ldots, L$, and the white noise power σ^2. The autocorrelation sequence of x_n is

$$r_n = \sigma^2 \delta_n + \frac{1}{2} \sum_{l=1}^{L} A_l^2 \cos(n\omega_l). \tag{1.5.17}$$

What is worth noting is the harmonic form of r_n. In particular, note that although r_n contains no information about the phase of the sinusoids, the autocorrelation sequence does contain information concerning the power and the frequency of each sinusoidal component. In order to specify the solution of this problem, let us define $A(z)$ to be the polynomial with $2L$ roots on the unit circle at the frequencies of the

sinusoids in (1.5.16), i.e.,

$$\begin{aligned} A(z) &= 1 + \sum_{k=1}^{2L} a_k z^{-k} \\ &= \prod_{k=1}^{L} (1 - e^{j\omega_l} z^{-1})(1 - e^{-j\omega_l} z^{-1}). \end{aligned} \tag{1.5.18}$$

If $\mathbf{a}$ is the vector which contains the coefficients of $A(z)$,

$$\mathbf{a} = \begin{bmatrix} 1 & a_1 & a_2 & \cdots & a_{2L} \end{bmatrix}^T,$$

then $\mathbf{a}$ satisfies the eigenequation [68]

$$R_x \mathbf{a} = \sigma^2 \mathbf{a}$$

where,

$$R_x = \begin{bmatrix} r_0 & r_1 & \ddots & r_{2L} \\ r_1 & r_0 & \ddots & \ddots \\ \ddots & \ddots & \ddots & r_1 \\ r_{2L} & \ddots & r_1 & r_0 \end{bmatrix},$$

the autocorrelation matrix of x_n and σ^2 is the variance of the white noise w_n, is a symmetric positive definite Toeplitz matrix. Thus, σ^2 is an eigenvalue of the autocorrelation matrix R_x and $\mathbf{a}$ is the corresponding eigenvector. The polynomial $A(z)$ associated with the eigenvector $\mathbf{a}$ is referred to the *eigenfilter* of x_n [87]. Although R_x has $2L + 1$ eigenvalues, it is easy to show [109], [55] that σ^2 must be the *minimum* eigenvalue of R_x. An interesting property is that the eigenfilter (1.5.18) associated with the minimum eigenvalue of a symmetric positive definite Toeplitz matrix has all its zeros on the unit circle, provided the minimum eigenvalue has multiplicity equal to one.

A fast algorithm for solving this structured matrix problem is outlined in Ch. 7. Exploiting a recurrence relation for the principal minors of a symmetric positive definite Toeplitz matrix of order n, an $O(n^2)$ algorithm has been derived. Although its conceptual simplicity, the performances of the considered algorithm turn out to be comparable with those ones of the other proposed algorithms for solving such a problem.

1.6 Brownian motion in Signal Processing, Brownian and diagonal plus semi–separable matrices

Brownian motion is an important tool for modeling continuous–time random signals. It is the simplest and most rigorous way to introduce the classical white noise, which

is very often used in signal theory [9]. But as we are most interested in discrete–time Brownian motion, we shall use discrete–time Brownian motion, sometimes called the random walk stochastic process. This signal can be written as

$$b_n = \sum_{k=0}^{n} u_k,$$

where the random variables u_k, increments of b_n, are zero mean, independent, and with variance c_k. The elements of the covariance matrix $R \triangleq (r_{mn})$ of b_n become

$$r_{mn} \triangleq E(b_m b_n) = \sum_{k=0}^{\min\{m,n\}} c_k \triangleq \rho_{\min\{m,n\}}.$$

Moreover, the elements of the covariance matrix $\hat{R} \triangleq (\hat{r}_{mn})$ of the signal x_n, which is discrete–time Brownian motion with added independent white noise w_n, are given by

$$\hat{r}_{mn} = r_{mn} + \sigma_n^2 \delta_{mn}, \tag{1.6.19}$$

where σ_n^2 is the variance of w_n and

$$\delta_{mn} = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{if } m \neq n \end{cases}$$

is the Kronecker symbol. The extraction of a signal s_n from Brownian noise modeled by x_n requires signal processing techniques such as detection and estimation. For these techniques, it is generally convenient to represent the signals in vectorial form [103], and then to solve the linear systems involving the covariance matrices associated with (1.6.19).

Define $v_i \triangleq \rho_i + \sigma_i^2$ and consider the random vector $x = [x_0, x_1, \ldots, x_{N-1}]^T$. Its covariance matrix $\tilde{R}$ is an $N \times N$ matrix, called *Brownian matrix,*

$$\tilde{R} = \begin{bmatrix} v_0 & r_0 & r_0 & \cdots & r_0 \\ r_0 & v_1 & r_1 & \cdots & r_1 \\ r_0 & r_1 & v_2 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \ddots & r_{N-2} \\ r_0 & r_1 & \vdots & r_{N-2} & v_{N-1} \end{bmatrix}.$$

Brownian matrices belong to the broader class of matrices that are the sum of a diagonal matrix and a semi–separable one, where a matrix $S \triangleq (s_{jm})$ of order N is called *semi–separable* if the offdiagonal entries can be written as

$$s_{ij} = \begin{cases} u_i w_j & \text{for } i < j, \\ \\ p_i q_j & \text{for } i > j. \end{cases}$$

Diagonal plus semiseparable (DSS) matrices also appear when one discretizes the Green's function for the two point boundary value problem appropriately, and also from the discretization of eigenvalue problems for integral operators whose kernels can be written as the sum of a semiseparable part and a degenerate one [54], [63], [117]. The inverse of a tridiagonal matrix has this form too.

In Ch. 8, new fast and stable algorithms for solving basic linear algebra problems involving diagonal plus semi–separable matrices. In particular, fast and stable algorithms for solving linear systems involving DSS matrices, for reducing symmetric DSS matrices into symmetric tridiagonal ones and for reducing DSS matrices into bidiagonal ones by means of orthogonal transformations are considered. The proposed algorithms are at least two times faster than stable algorithms known in the literature for solving such problems.

1.7 Chapter by chapter overview

This thesis presents new fast and reliable algorithms for solving a variety of structured matrix problems as introduced in the previous sections. This is achieved by exploiting the structure of the involved matrices through the use of displacement theory and/or underlying rank properties and points out the main contribution of this thesis.

Chapter 2 deals with fast direct methods for the triangular factorization of structured matrices, based on the displacement theory. The emphasis is mostly on the generalized Schur algorithm.

In Ch. 3, a particular implementation of the generalized Schur algorithm, which speeds up the computations with a factor proportional to the size of the handled problem of the kernel Toeplitz/Hankel–structured STLS algorithms considered in Ch. 4 and Ch. 5 is described. The analysis of the numerical stability of the algorithm is given as well. The results, obtained in collaboration with Paul Van Dooren and Sabine Van Huffel, have been published in [97].

The kernel of the STLS algorithm proposed in Ch. 4 is the solution of a linear system involving a particular Toeplitz–block matrix. The main result of this chapter is the exploitation of the structure of the generators involved in the generalized Schur algorithm thereby drastically reducing the computational complexity of the algorithm. The performance of this algorithm is compared with that of two other implementations of algorithms for solving the Toeplitz (Hankel) STLS problem. The comparison is carried out on a recently proposed speech compression scheme. In Ch. 5 a fast algorithm, based on the generalized Schur algorithm, for the basic deconvolution problem is developed in order to show how an efficient STLS algorithm can be obtained once a particular problem or application has been specified. Also in this case, exploiting the structure of the involved generators, we are able to construct an algorithm with a speed up proportional to the size of the handled problem with respect to the algorithms known in the literature for STLS problems. A simulation experiment based on a medical application (whose kernel problem is the deconvolution problem) in renography is described. This example

is used to demonstrate the improved efficiency of the new algorithm compared to other algorithms for solving this type of STLS problems. By means of a Monte-Carlo simulation, using several noise levels, we show the improved statistical accuracy of the deconvolution results obtained with the STLS estimator as compared to other estimators such as the TLS estimator which does not impose a structure on the correction matrix $[\Delta A \ \ \Delta b]$.

The results presented in Ch. 4 and Ch. 5, obtained in collaboration with Philippe Lemmerling and Sabine Van Huffel, have been published in [80], [93], [94]. Morever `Fortran` implementations of these algorithms will be included in the near future into the STLS toolbox in the `SLICOT` [8] library[1].

In Ch. 6 a fast algorithm to compute the R factor of the QR factorization of a block–Hankel matrix H, based on the generalized Schur algorithm, is considered. This is a kernel problem arising in subspace identification. The proposed algorithm allows to handle the rank–deficient case. Moreover it is shown that in most cases, the considered algorithm is one order of magnitude faster than the classical QR algorithm. The results presented in this chapter, obtained in collaboration with Daniel Kressner, Vasile Sima, Paul Van Dooren and Sabine Van Huffel, have been published in [92], [96]. Moreover the algorithm described in this chapter is the kernel of the routine `IB01M1.f` included in the subspace identification toolbox of the `SLICOT` library.

Ch. 7 concerns a fast algorithm for computing the smallest eigenpair of symmetric positive definite Toeplitz matrices. Exploiting a recurrence relation for the principal minors of a symmetric positive definite Toeplitz matrix of order n, an $O(n^2)$ algorithm has been derived. Although its conceptual simplicity, the performances of the considered algorithm turn out to be comparable with those ones of the other proposed algorithms for solving such a problem. The results, obtained in collaboration with Daniel Boley, have been published in [89].

Ch. 8 outlines fast and stable algorithms for solving basic linear algebra problems involving diagonal plus semi–separable matrices. In particular, fast and stable algorithms for solving linear systems involving DSS matrices, for reducing symmetric DSS matrices into symmetric tridiagonal ones and for reducing DSS matrices into bidiagonal ones by means of orthogonal transformations are considered. The proposed algorithms are at least two times faster than stable algorithms known in the literature for solving such problems. The results, obtained in collaboration with Shivkumar Chandrasekaran and Sabine Van Huffel, have been published in [90], [91].

Finally, Ch. 9 summarizes the conclusions and outlines suggestions for further research.

[1] More detailed information on the `SLICOT` library is available on the website `http://www.win.tue.nl/wgs/niconet.html`.

Chapter 2

DISPLACEMENT STRUCTURE AND GENERALIZED SCHUR ALGORITHM

2.1 Introduction

The iterative methods for solving STLS problems proposed in Ch. 3 and Ch. 4 require to solve, at each iteration, a linear system

$$Ax = b, \tag{2.1.1}$$

where A is a symmetric Toeplitz–block matrix[1]. These matrices vary at each iteration and, in many applications, can be very large. In such cases, the computational burden required to solve these systems by means of classical algorithms (Gaussian elimination, QR algorithm, ...) can become prohibitively large.

These algorithms can be speeded up by exploiting the so–called *displacement structure* of the involved matrices.

The *displacement* of a Hermitian matrix was originally introduced in [65] and the *displacement theory* has been intensively studied in the last few decades (see [66] and the references therein).

In this chapter we introduce the displacement theory for strongly regular[2] Hermitian matrices. In STLS problems the involved matrices are strongly regular, indeed. However this assumption is not restrictive. Some look-ahead techniques, based on the displacement theory, have been developed to solve problems involving matrices that are not strongly regular [111]. Moreover the rank-deficient case is handled in Ch. 5. The extensions to the non–symmetric and the complex (Hermitian and non-Hermitian) case is quite straigthforward and can be found in [66].

This chapter is organized as follows.

In § 2.2 the basic concepts of displacement theory for Hermitian matrices are introduced. The equivalence between Schur reduction and Gaussian elimination is shown in § 2.3. The property that the Schur complement of a matrix and a matrix itself share the same displacement rank is proved in § 2.4. In this section the generalized Schur algorithm is derived. In § 2.5, the displacement rank of symmetric Toeplitz and Toeplitz–block matrices is considered. Finally, the QR factorization of matrices having low displacement rank is considered in § 2.6. The conclusions can be found at the end of the chapter.

2.2 Displacement Theory for symmetric matrices

Let R be a strongly regular symmetric matrix and Φ be a strictly lower triangular matrix of order n, respectively. The displacement of R, with respect to Φ, is defined as

$$\nabla_\Phi R \triangleq R - \Phi R \Phi^T \ . \tag{2.2.2}$$

[1] In Ch. 4 the solution, in the least squares sense, of a rectangular linear system where the coefficient matrix is a full-rank rectangular Toeplitz–block, has to be computed. However we will see that this problem could be transformed into one equivalent to (2.1.1).

[2] A square matrix is said to be strongly regular if all the leading principal minors are different from zero.

The matrix Φ, in the original definition (see [65]), is the lower shift matrix Z of order n, defined in (2.5.11). Indeed, the product ZRZ^T displaces R downwards along the main diagonal by one position, explaining the name *displacement* for $\nabla_Z R$ (see Fig. 2.1). The rank α of $\nabla_\Phi R$ is called *displacement rank*. If $\nabla_\Phi R$ has

$$\nabla_Z R = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{1,n} \\ \vdots & \cdots & \vdots & \vdots \\ r_{n,1} & r_{n,1} & \cdots & r_{n,n} \end{bmatrix} - \left[\begin{array}{c|ccc} & & & \\ \hline & r_{1,1} & \cdots & r_{1,n-1} \\ & \vdots & \cdots & \vdots \\ & r_{n-1,1} & \cdots & r_{n-1,n-1} \end{array}\right]$$

Figure 2.1. $\nabla_Z R$ is obtained by shifting R downwards along the diagonal.

low displacement rank, *independent* of n, then R is said to be *structured* with respect to the displacement ∇_Φ defined by (2.2.2), and α is called the *displacement rank* of R. We observe that $\nabla_\Phi R$ is symmetric too, and therefore has further structure: its eigenvalues are real and so we can define the *displacement inertia* of R as the pair $\{p, q\}$, where p (resp. q) is the number of strictly positive (resp. negative) eigenvalues of $\nabla_\Phi R$. Of course, the displacement rank is $\alpha = p + q$. Therefore, we can write

$$\nabla_\Phi R = R - \Phi R \Phi^T = GJG^T \ , \tag{2.2.3}$$

or, equivalently,

$$R = \Phi R \Phi^T + GJG^T. \tag{2.2.4}$$

where

$$J = J^T = (I_p \oplus -I_q), \tag{2.2.5}$$

is a signature matrix, and $G \in \mathbb{R}^{n \times \alpha}$. The pair $\{G, J\}$ is called a ∇_Φ*−generator* of R. This representation is clearly not unique; for example, $\{G\Theta, J\}$ is also a generator for any $J-$orthogonal matrix Θ (*i.e.*, for any Θ such that $\Theta J \Theta^T = J$). This is because

$$G \underbrace{\Theta J \Theta^T}_{J} G^T = GJG^T.$$

Moreover, the first p and the last q columns of G are called *positive* and *negative* $\nabla_\Phi-$*generators*, respectively, or, if there is no ambiguity, simply *positive* and *negative generators*. Let $\{G, J\}$ be a $\nabla_\Phi-$generator of R. Let h be the first nonzero row of G. $\{G, J\}$ is said to be in *proper form* if h has all the elements equal to zero with the exception of a single pivot element in position π. This pivot has to be in the first p positions when $hJh^T > 0$ (positive step) and in the last q positions when $hJh^T < 0$ (negative step) (note that the case $hJh^T = 0$ is ruled out by the strong regularity

assumption on R [66]). Without loss of generality, we can choose π either equal to 1 if $hJh^T > 0$, or equal to α if $hJh^T < 0$. Define $\Phi_i \triangleq \Phi^i$. It is straightforward to see that

$$\nabla_\Phi \Phi R \Phi^T = \Phi G J G^T \Phi^T.$$

Then

$$\Phi R \Phi^T = \Phi_2 R \Phi_2^T + \Phi G J G^T \Phi^T. \tag{2.2.6}$$

Replacing(2.2.6) in (2.2.4) it turns out

$$R = \Phi_2 R \Phi_2^T + \Phi G J G^T \Phi^T + G J G^T.$$

Proceeding recursively and using the fact that Φ is nilpotent, i.e., $\Phi_j = \mathbf{0}$, $j > n-1$, we can easily conclude that

$$R = \sum_{i=0}^{n-1} \Phi_i G J G^T \Phi_i^T. \tag{2.2.7}$$

2.3 Equivalence between Schur reduction and Gaussian elimination

The concept of Schur complements perhaps first arose in the context of triangular factorization of matrices. It was implicit in Gauss' work on linear equations and more explicit in the paper of Schur [113]. In this section we consider the problem of triangular factorization, which is basically affected by the Gaussian elimination technique, and show that adding displacement structure allows us to speed up the Gaussian elimination procedure. This leads us to what is called the Generalized Schur Algorithm.

We start by reviewing what we shall call the Gauss-Schur reduction procedure.

2.3.1 Schur Reduction (Deflation)

The triangular decomposition of a matrix $R \in \mathbb{R}^{n\times n}$ is denoted by

$$R = L D^{-1} L^T, \tag{2.3.1}$$

where $D = \mathrm{diag}\{d_1, d_2, \ldots, d_n\}$ is a diagonal matrix, and the lower triangular factor L is normalized in such a way that the $\{d_i\}$ appear on its main diagonal. The columns of L are denoted by l_i, $i = 1, \ldots, n$.

Algorithm 2.3.1 (Schur Reduction) *Given $R \in \mathbb{R}^{n\times n}$, start with $R_1 = R$ and repeat for $i = 1, 2, \ldots, n$:*

1. *Let l_i denote the first column of R_i and d_i denote the upper-left-corner element of R_i.*

2. Perform the Schur reduction step:

$$\begin{bmatrix} 0 & 0 \\ 0 & R_{i+1} \end{bmatrix} = R_i - l_i d_i^{-1} l_i^T . \tag{2.3.2}$$

The matrices R_i, $i = 2, \ldots, n$, are known as the Schur complements of the leading $i \times i$ blocks of R.

◊

We therefore see that the Schur reduction step (2.3.2) *deflates* the matrix R_i by subtracting a rank-one matrix from it leading to a new matrix R_{i+1}.

By successively repeating (2.3.2) we obtain the triangular factorization of R,

$$\begin{aligned} R_1 &= l_1 d_1^{-1} l_1^T + \begin{bmatrix} 0 & 0 \\ 0 & R_2 \end{bmatrix}, \quad R_2 \in \mathbb{R}^{(n-1)\times(n-1)} \\ &= d_1^{-1} l_1 l_1^T + d_2^{-1} l_2 l_2^T + \begin{bmatrix} 0 & 0 \\ 0 & R_3 \end{bmatrix}, \quad R_3 \in \mathbb{R}^{(n-2)\times(n-2)} \\ &\vdots \\ &= L D^{-1} L^T . \end{aligned}$$

2.3.2 Close Relation to Gaussian Elimination

The Schur reduction procedure is in fact the same as Gaussian elimination. To see this, consider the first step of (2.3.2):

$$\begin{bmatrix} 0 & 0 \\ 0 & R_2 \end{bmatrix} = R_1 - l_1 d_1^{-1} l_1^T . \tag{2.3.3}$$

If we partition the entries of l_1 into $l_1 \triangleq \mathrm{col}\{d_1, t_1\}$, where t_1 is also a column vector, then the above equality can be written as

$$R = \begin{bmatrix} 1 & 0 \\ t_1 d_1^{-1} & I_{n-1} \end{bmatrix} \begin{bmatrix} d_1 & \\ & R_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_1 d_1^{-1} & I_{n-1} \end{bmatrix}^T , \tag{2.3.4}$$

or, equivalently, as

$$\begin{bmatrix} 1 & 0 \\ -t_1 d_1^{-1} & I_{n-1} \end{bmatrix} R = \begin{bmatrix} d_1 & 0 \\ t_1 & R_2 \end{bmatrix}^T ,$$

where I_{n-1} is the identity matrix of dimension $n-1$. This relation shows why (2.3.2), which we called Schur reduction, is closely related to Gaussian elimination. Note also that the above Schur reduction procedure can readily be extended to strongly-regular non-symmetric matrices to yield the so-called LDU decompositions (see, *e.g.*, [67]).

2.4 Generalized Schur Algorithm and ∇_Φ–generators of the Schur complement

The Schur–Gauss reduction algorithm 2.3.1 requires the computation of the Schur complements $R_i, i = 1, \ldots, n$. We consider a recursive way to compute R_i based on the knowledge of a ∇_Φ–generator G of R_1.
Let $R_1 = R$, $G_1 = G$. Let h_1 be the first row of G_1 and suppose first $h_1 J h_1^T > 0$. The following lemma tells us that it is always possible to reduce a generator in proper form.

Lemma 2.4.1: *Consider two nonzero row vectors v_1 and v_2 of length α. If $v_1 J v_1^T = v_2 J v_2^T$, then there exists an $\alpha \times \alpha$ J–orthogonal matrix Θ such that $v_1 = v_2 \Theta$.*

Proof. (see [67]). ◊

Let Θ_1 be a J–orthogonal matrix such that

$$\hat{G}_1 = G_1 \Theta_1 = [\tilde{g}_1 | \tilde{G}_1] \tag{2.4.5}$$

is in proper form and $\tilde{g}_1$ is the first column of $\hat{G}_1$, with $\tilde{g}_1 \neq 0$. It turns out that

$$d_1^{-1} l_1 l_1^T = \tilde{g}_1 \tilde{g}_1{}^T, \tag{2.4.6}$$

hence, d_1 and l_1 can be directly computed. Let

$$\tilde{I}_{1,n} \triangleq \left[\begin{array}{c|c} 0 & \\ \hline & I_{n-1} \end{array}\right].$$

We now show that the Schur complement R_2 inherits the displacement structure of R_1, i.e., if R_1 has low displacement rank, let say α_1, the Schur complement R_2 has displacement rank $\alpha_2 \leq \alpha_1$. Taking (2.2.7) and (2.3.2) into account, R_2 can be written in the following way,

$$\begin{aligned}
\left[\begin{array}{c|c} 0 & \\ \hline & R_2 \end{array}\right] &= \tilde{I}_{1,n} \left(R - d_1^{-1} l_1 l_1^T\right) \tilde{I}_{1,n}^T \\
&= \tilde{I}_{1,n} \left(\sum_{i=0}^{n-1} \Phi_i \hat{G}_1 J \hat{G}_1^T \Phi_i^T \right) \tilde{I}_{1,n}^T - d_1^{-1} \tilde{I}_{1,n} l_1 l_1^T \tilde{I}_{1,n}^T \\
&= \tilde{I}_{1,n} \left[\hat{G}_1 J \hat{G}_1^T\right] \tilde{I}_{1,n}^T + \sum_{i=1}^{n-1} \Phi_i \hat{G}_1 J \hat{G}_1^T \Phi_i^T - d_1^{-1} \tilde{I}_{1,n} l_1 l_1^T \tilde{I}_{1,n}^T
\end{aligned}$$

We observe that

$$\tilde{I}_{1,n} \left[\hat{G}_1 J \hat{G}_1^T\right] \tilde{I}_{1,n}^T = \tilde{I}_{1,n} \tilde{g}_1 \tilde{g}_1{}^T \tilde{I}_{1,n}^T + [\mathbf{0}|\tilde{G}_1] J [\mathbf{0}|\tilde{G}_1]^T$$

Hence, taking (2.4.6) into account,

$$\left[\begin{array}{c|c} 0 & \\ \hline & R_2 \end{array}\right] = [\mathbf{0}|\tilde{G}_1]J[\mathbf{0}|\tilde{G}_1]^T + \sum_{i=1}^{n-1} \Phi_i \hat{G}_1 J \hat{G}_1^T \Phi_i^T.$$

Furthermore, since the first row of $\tilde{G}_1$ is $\mathbf{0}$,

$$\Phi_{n-1}[\mathbf{0}|\tilde{G}_1]J[\mathbf{0}|\tilde{G}_1]^T\Phi_{n-1}^T = \mathbf{0}.$$

Therefore, considering (2.4.5), we have

$$\begin{aligned}
\left[\begin{array}{c|c} 0 & \\ \hline & R_2 \end{array}\right] &= \sum_{i=0}^{n-2} \Phi_i[\mathbf{0}|\tilde{G}_1]J[\mathbf{0}|\tilde{G}_1]^T\Phi_i^T + \sum_{i=1}^{n-1} \Phi_i[\tilde{g}_1|\mathbf{0}]J[\tilde{g}_1|\mathbf{0}]^T\Phi_i^T \\
&= \sum_{i=0}^{n-2} \Phi_i[\mathbf{0}|\tilde{G}_1]J[\mathbf{0}|\tilde{G}_1]^T\Phi_i^T + \sum_{i=0}^{n-2} \Phi_i[\Phi\tilde{g}_1|\mathbf{0}]J[\Phi\tilde{g}_1|\mathbf{0}]^T\Phi_i^T \\
&= \sum_{i=0}^{n-2} \Phi_i[\Phi\tilde{g}_1|\tilde{G}_1]J[\Phi\tilde{g}_1|\tilde{G}_1]^T\Phi_i^T \\
&= \sum_{i=0}^{n-2} \Phi_i G_2 J G_2^T \Phi_i^T. \qquad (2.4.7)
\end{aligned}$$

Hence, the computation of a ∇_Φ–generator $\{G_2, J\}$ of $\left[\begin{array}{c|c} 0 & \\ \hline & R_2 \end{array}\right]$ can be summarized in the following steps, depicted in Fig. 2.2.

$$G_1 = \begin{bmatrix} * & * & * \\ * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \end{bmatrix} \xrightarrow{\text{Step 1}} \begin{bmatrix} * & 0 & 0 \\ * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \end{bmatrix} \xrightarrow{\text{Step 2}} \begin{bmatrix} 0 & 0 & 0 \\ * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \end{bmatrix} = G_2$$

Figure 2.2. Illustrating the proper form of the generator recursion (case $h_1 J h_1^T > 0$).

- Step 1: reduction of G_1 into proper form.
- Step 2: computation of the matrix–vector $\Phi\tilde{g}_1$.

The first step eliminates all entries in the first row of G_i except the pivot entry in the upper left corner position. The second step is needed to eliminate the pivot entry.

$$G_1 = \begin{bmatrix} * & * & * \\ * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \end{bmatrix} \xrightarrow{\text{Step 1}} \begin{bmatrix} 0 & 0 & * \\ * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \end{bmatrix} \xrightarrow{\text{Step 2}} \begin{bmatrix} 0 & 0 & 0 \\ * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \end{bmatrix} = G_2$$

Figure 2.3. Illustrating the proper form of the generator recursion (case $h_1 J h_1^T < 0$).

The case $h_1 J h_1^T < 0$ can be handled in a similar way and it is depicted in Fig. 2.3. Indeed, in this case there exists a J–orthogonal matrix Ξ_1 such that

$$\hat{G}_1 = G_1 \Xi_1 = [\tilde{G}_1 | \tilde{g}_1], \tag{2.4.8}$$

is in proper form, where, this time, $\tilde{g}_1$ is the α–th column of $\hat{G}_1$ with $\tilde{g}_1 \neq 0$. Then

$$d_1^{-1} l_1 l_1^T = -\tilde{g}_1 \tilde{g}_1^T.$$

Proceeding as previously, we have

$$\left[\begin{array}{c|c} & \\ \hline & R_2 \end{array}\right] = \sum_{i=0}^{n-2} \Phi_i [\tilde{G}_1 | \Phi \tilde{g}_1] J [\tilde{G}_1 | \Phi \tilde{g}_1]^T \Phi_i^T$$
$$= \sum_{i=0}^{n-2} \Phi_i G_2 J G_2^T \Phi_i^T. \tag{2.4.9}$$

In both cases $\{G_2, J\}$ is a ∇_Φ–generator of $\left[\begin{array}{c|c} & \\ \hline & R_2 \end{array}\right]$. Reducing G_2 in proper form, l_2 and d_2 can now be computed. Recursively proceeding, we obtain the LDL^T factorization of R.

The procedure just described is better known under the name of Generalized Schur algorithm.

Algorithm 2.4.1 (Generalized Schur Algorithm in Proper Form) *Given a matrix $R \in \mathbb{R}^{n \times n}$ that satisfies (2.2.3) for some full rank $G \in \mathbb{R}^{n \times \alpha}$. Start with $G_1 = G$ and perform the following steps*

`for` $i = 1, \ldots, n$

Let h_i be the i–th row of G_i.

transform G_i to proper form $\hat{G}_i$,

compute l_i *and* d_i,

update $\hat{G}_i$ *to* G_{i+1}.

end

$\diamond$

As already mentioned, the computational burden of each step of GSA is given by

1. reduction of the generator matrix in proper form,
2. application of the matrix Φ to the pivot generator vector.

The reduction of a generator matrix in proper form can be accomplished in $O(\alpha(n-i))$ flops by means of J–orthogonal transformations (see A.1). Hence, it turns out it is essential to consider a matrix Φ such that the displacement rank is $\alpha \ll n$, and the product

$$\Phi \tilde{g}_i, i = 1, \ldots, n \tag{2.4.10}$$

can be computed efficiently.

2.5 Displacement of symmetric Toeplitz and Toeplitz–block matrices

As already said in the introduction of the chapter, we are interested in fast algorithms for solving linear systems involving symmetric Toeplitz–block matrices.

2.5.1 Displacement of symmetric Toeplitz matrices

Choosing Φ equal to the shift matrix

$$Z \triangleq \begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ & 1 & 0 & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{bmatrix}. \tag{2.5.11}$$

it is easy to see that the displacement rank of symmetric Toeplitz matrices

$$T = \begin{bmatrix} c_0 & c_{-1} & c_{-2} & \cdots & c_{-n+1} \\ c_1 & c_0 & c_{-1} & \cdots & c_{-n+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix}, \quad c_k = c_{-k}, \tag{2.5.12}$$

is 2, except when either all $c_i = 0, i \neq 0$, a case we shall exclude. This situation is depicted in Fig. 2.4.

$$\nabla_Z T = T - ZTZ^T = \begin{bmatrix} c_0 & c_{-1} & c_{-2} & \dots & c_{-n+1} \\ c_1 & & & & \\ \vdots & & & & \\ c_{n-1} & & & & \end{bmatrix}, \quad c_k = c_{-k}$$

Figure 2.4. $\nabla_Z T$, T Toeplitz matrix.

Assuming $c_0 = 1$, a generator for T is $\{u_1, v_1, (1 \oplus -1)\}$, where

$$u_1 = \begin{bmatrix} 1 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix}, \quad \text{and} \quad v_1 = \begin{bmatrix} 0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix};$$

$$T - ZTZ^T = \begin{bmatrix} 1 & 0 \\ c_1 & c_1 \\ \vdots & \vdots \\ c_{n-1} & c_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ c_1 & c_1 \\ \vdots & \vdots \\ c_{n-1} & c_{n-1} \end{bmatrix}^T. \tag{2.5.13}$$

Therefore, $O(2(n-i))$ flops are required to reduce the generator G_i in proper form at the i–th iteration of GSA. Moreover (2.4.10) corresponds to shifting the entries of the vectors $\tilde{g}_i, i = 1, \dots, n$, downward one position. Hence the complexity of GSA is $O(n^2)$.

2.5.2 Displacement of symmetric Toeplitz–block matrices

A symmetric Toeplitz–block matrix is a symmetric block–matrix

$$T_b = \begin{bmatrix} T_{1,1} & T_{1,2} & \dots & T_{1,M} \\ T_{2,1} & T_{2,2} & \dots & T_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ T_{M,1} & T_{M,2} & \dots & T_{M,M} \end{bmatrix}$$

where $T_{i,j} \in \mathbb{R}^{n_i \times n_j}$ are Toeplitz matrices. Let $N = \sum_{j=1}^{M} n_j$.

Setting

$$\Phi = \begin{bmatrix} Z_{n_1} & & & \\ & Z_{n_2} & & \\ & & \ddots & \\ & & & Z_{n_M} \end{bmatrix}$$

where Z_{n_i}, $i = 1, \ldots, M$, are the shift matrices of order n_i, respectively, the displacement $\nabla_\Phi T_b = T_b - \Phi T_b \Phi^T$ is the sum of M rank 2 matrices (see Fig. 2.5).

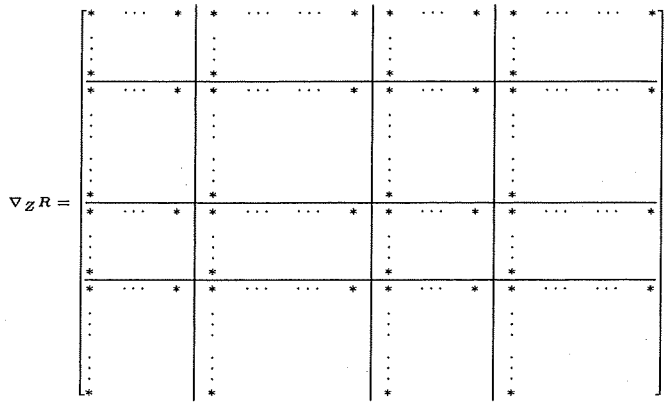


Figure 2.5. Nonzero pattern of $\nabla_\Phi T_b$, T_b Toeplitz–block matrix.

Assuming the element in the main diagonal of each Toeplitz matrix $T_{i,i}$, $i = 1, \ldots, M$, is equal to 1, a generator $\{[U, V], I_{M/2} \oplus -I_{M/2}\}$ can be recursively computed according to the following procedure, described in `matlab`–style.

Algorithm 2.5.1 (Generator of Toeplitz–block matrices)

```
for i = 1 : M,
    N_1 = 1 + sum_{j=1}^{i-1} n_j;
    U(:, i) = T_b(:, N_1);
    for j = 1 : i - 1,
        N_2 = 1 + sum_{k=1}^{j-2} n_k;
        U(N_2, i) = 0;
    end
    V(:, i) = U(:, i);
    V(N_1, i) = 0;
end
```

$\Diamond$

Indeed, let $x = [1, x_2, \ldots, x_N]^T$ be the first column of T_b. Then

$$M \triangleq \left[\begin{array}{c|ccc} 1 & x_2 & \cdots & x_N \\ \hline x_2 & & & \\ \vdots & & & \\ x_N & & & \end{array}\right] = \begin{bmatrix} 1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \begin{bmatrix} 1 & x_2 & \cdots & x_N \end{bmatrix} - \begin{bmatrix} 0 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \begin{bmatrix} 0 & x_2 & \cdots & x_N \end{bmatrix}.$$

Hence $[1, x_2, \ldots, x_N]^T$ and $[0, x_2, \ldots, x_N]^T$ are a couple of generators of T_b. In a similar way, the other generators can be computed from the matrix obtained subtracting M from $\nabla_\Phi T_b$. We observe that the matrix $[U, V]$ is directly obtained from T_b without any computation.
The i–th step of GSA requires $O(2M(N-i))$flops. Hence the computational complexity of GSA applied to T_b is $O(MN^2)$.

2.5.3 Displacement of symmetric block–Toeplitz matrices

A symmetric block–Toeplitz matrix is a symmetric block–matrix

$$T^{(b)} = \begin{bmatrix} T_1 & T_2 & \ddots & T_n \\ T_2^T & T_1 & \ddots & \ddots \\ \ddots & \ddots & \ddots & T_2 \\ T_n^T & \ddots & T_2^T & T_1 \end{bmatrix} \tag{2.5.14}$$

where $T_i \in \mathbb{R}^{M \times M}$ and T_1 is symmetric.
Setting

$$\Phi = \begin{bmatrix} \mathbf{0}_M & & & \\ Z_M & \ddots & & \\ & \ddots & \ddots & \\ & & Z_M & \mathbf{0}_M \end{bmatrix}$$

the displacement $\nabla_\Phi T^{(b)} = T^{(b)} - \Phi T^{(b)} \Phi^T$

$$\nabla_\Phi T^{(b)} = \begin{bmatrix} T_1 & T_2 & \ldots & T_n \\ T_2^T & & & \\ \vdots & & & \\ T_n^T & & & \end{bmatrix} \tag{2.5.15}$$

is a $2M$ rank matrix. A generator of $T^{(b)}$ can easily be derived considering (2.5.15) as a sum of M rank 2 matrices. Hence the computational complexity of GSA applied to block–Toeplitz matrices (2.5.14) is $0(MN^2)$, where $N = M \times n$.

2.6 QR Factorization of Structured Matrices

We now show how the displacement ideas can be used to suggest a fast algorithm for the QR factorization of structured matrices (with Q orthogonal and R upper triangular). This has been a much studied problem (see, for instance, [15], [29], [30], [122].) The displacement approach described below is much simpler, conceptually and algebraically.

Let X be an $n \times n$ matrix[3] that has low displacement rank with respect to the displacement $X - ZXZ^T$. Form the displacement of

$$M = \begin{bmatrix} -I & X & 0 \\ X^T & 0 & X^T \\ 0 & X & 0 \end{bmatrix} ,$$

with $\Phi = Z \oplus Z \oplus Z$, and find a generator for M. A general procedure for doing this has been given in [64]; when X is Toeplitz, one can obtain a generator by means of algorithm 2.5.1.

After n steps of the generalized Schur algorithm applied to a generator of M, we have a generator of

$$M_1 = \begin{bmatrix} X^TX & X^T \\ X & 0 \end{bmatrix} .$$

After another n steps, we have the partial triangularization (where L is $n \times n$ lower triangular, and U is an $n \times n$ matrix)

$$M_1 = \begin{bmatrix} L \\ U \end{bmatrix} D^{-1} \begin{bmatrix} L^T & U^T \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -I \end{bmatrix} .$$

By equating terms on both sides of the above equality we conclude that

$$X^TX = (LD^{-T/2})(LD^{-T/2})^T \ , \ (UD^{-T/2})(LD^{-T/2})^T = X \ ,$$

and $(UD^{-T/2})(UD^{-T/2})^T = I$. Therefore, we can identify

$$Q = UD^{-T/2} \quad , \quad R = (LD^{-T/2})^T \ ,$$

as the Q and R factors in the QR factorization of X. Here, D is a positive-definite diagonal matrix and $D^{1/2}$ denotes a diagonal matrix whose entries are the square-roots of the diagonal entries of D. In summary, the QR factors of the structured matrix X can be obtained by applying the Schur recursion to a properly defined extended structured matrix M.

The above procedure may encounter numerical difficulties in finite precision implementations. However, in [23], it has been shown how to employ such embedding constructions in order to develop the first provably backward-stable algorithm for the solution of linear systems of equations with structured coefficient matrices.

[3] The argument also applies to rectangular matrices, say $m \times n$.

In a similar way, we show how to compute the R factor of the QR factorization of a full-rank[4] matrix $X \in \mathbb{R}^{m \times n}$, with low displacement rank with respect to the displacement $\nabla_Z X = X - ZXZ^T$, without computing the orthogonal factor Q. It is well known that the R^T factor of the QR factorization of a full rank matrix X is equivalent to the Cholesky factor L of the correlation matrix $X^T X$, apart from a multiplication by a sign matrix. Form the displacement of the structured matrix

$$M_2 = \begin{bmatrix} -I & X \\ X^T & \mathbf{0} \end{bmatrix}.$$

After m steps of GSA applied to a generator of M_2 we have a generator for the Schur complement $M_3 = \mathbf{0} + X^T I^{-1} X = X^T X$. After another n steps, we have the Cholesky factor $LD^{-T/2}$ and, therefore, the matrix R. Also in this case, D is a positive-definite diagonal matrix and $D^{1/2}$ denotes a diagonal matrix whose entries are the square-roots of the diagonal entries of D.

2.7 Conclusions

Classical algorithms for linear systems, like Gaussian elimination and the QR algorithm can be prohibitively expensive when the size of the involved data are large.

Given the equivalence between the Schur reduction and Gaussian elimination, the main idea of displacement theory, introduced in this chapter, is to describe a matrix R of order n by a $n \times \alpha$ generator matrix G, where α is the displacement rank of R. If $\alpha \ll n$, a computational gain of one order of magnitude can in general be expected from the generalized Schur algorithm, since it operates directly on the generators. The displacement rank of (symmetric) Toeplitz matrices is 2. Therefore the computational complexity of the generalized Schur algorithm for (symmetric) Toeplitz matrices is $O(n^2)$. We have also shown that the QR factorization of structured matrices can efficiently be computed by means of the generalized Schur algorithm.

We conclude by observing that the displacement theory introduced in this chapter can be regarded as a special case of the generalized displacement structure [56], [112], [67]:

$$\Omega R \Delta^T - F R A^T = G B^T , \tag{2.7.16}$$

where $\{\Omega, \Delta, F, A\}$ are $n \times n$ and $\{G, B\}$ are $n \times \alpha$. Such equations uniquely define R when the diagonal entries $\{\omega_i, \delta_i, f_i, a_i\}$ of the displacement matrices $\{\Omega, \Delta, F, A\}$ satisfy

$$\omega_i \delta_j^T - f_i a_j^T \neq 0 \quad \text{for all } i, j . \tag{2.7.17}$$

The restriction that $\{\Omega, \Delta, F, A\}$ are lower triangular is the most general one that allows recursive triangular factorization [83].

[4]The case of a rank deficient structured matrix will be handled in Chap. 6.

Chapter 3

GENERALIZED SCHUR ALGORITHM: IMPLEMENTATIONS AND STABILITY

3.1 Introduction

The generalized Schur algorithm allows to compute a fast Cholesky decomposition of a wide variety of symmetric positive definite structured matrices, i.e., block–Toeplitz and Toeplitz–block matrices, matrices of the form T^TT, where T is a rectangular Toeplitz matrix [64], [28] and to compute a fast LDL^T factorization of *strongly regular* [13] structured matrices, where L is a triangular matrix and D is a diagonal matrix. The main computation at each iteration of GSA is the reduction of the generator matrix in proper form. This can be accomplished in different ways (see, e.g., [119]).

The stability property of the GSA depends on the way it is implemented [67], [23]. In [119] the GSA was shown to be stable, provided one hyperbolic rotation in factored form [16] is performed at each iteration. Similar results were obtained in [23], using the *OD procedure* or the *H procedure* instead of using the hyperbolic rotations in factored form. The computational complexity of GSA is $\mathcal{O}(\alpha N^2)$, where N is the order of the involved matrix and α is its displacement rank (see Ch.2).

The kernels of the algorithms proposed in Ch. 4 and Ch. 5 are the solution of a least squares problem, where the coefficient matrix is a rectangular Toeplitz–block matrix, and the solution of a symmetric linear system with Toeplitz-block coefficient matrix, respectively. These problems are solved by means of GSA. In order to further reduce the computational complexity of GSA applied to these problems by fully exploiting the sparsity of the generators, it is essential not to destroy the particular structure of the corresponding generators. Taking this into account, we managed to design fast and stable algorithms requiring only one hyperbolic rotation at each iteration[1]. Furthermore, modifying these algorithms by considering two hyperbolic rotations at each iteration, the computational complexity is further halved (see Ch. 4 and Ch. 5). The aim of this chapter is to prove that these implementations are still stable provided the hyperbolic rotations are performed in a *"stable"* way [97], [119], [23], [66].

The chapter is organized as follows. In § 3.2 we show how a generator matrix can be reduced in proper form, by means of a product of Givens and hyperbolic transformations. In particular, the implementation of the GSA when using two hyperbolic rotations is also considered. The stability analysis of this implementation of GSA is considered in in § 3.3. The conclusions are considered in § 3.4.

3.2 Reduction of a generator matrix in proper form

The main step of each iteration of GSA is the reduction of a generator matrix in proper form. This can always be achieved by means of a J–orthogonal transformation Θ (see lemma 2.4.1). The matrix Θ can be chosen in different ways [66], [23], [17]:

- a Householder hyperbolic transformation [105];

[1] The corresponding `matlab` code can be obtained from the authors upon request.

- a stabilized Householder hyperbolic transformation [17];
- a product of orthogonal (Householder or Givens matrices) and hyperbolic transformations [66].

Since implementations of GSA involving (stabilized) Householder (hyperbolic) transformations don't preserve the structure of the involved generators, we consider only those involving Givens and hyperbolic rotations. From a stability point of view, it is advisable to perform as less as possible hyperbolic rotations at each iteration of GSA [119], [66].

3.2.1 Reduction in proper form by means of elementary transformations

Let $G = [U\ V]$, where U and V are the matrices of the positive and negative generators, respectively. A way to reduce a generator in proper form, can be accomplished by first reducing both U and V in proper form. Then, the generator matrix in proper form is obtained by applying a suitable hyperbolic rotation to the pivot columns of U and V. Without loss of generality, we choose the first column of U and of V as pivot columns for the positive and the negative generators, respectively.

Taking into account the definition of J in (2.2.5), we observe that, considering Givens rotations Q_1 and Q_2, of order p and q, respectively, the matrices

$$Q_1 \oplus I_q = \begin{bmatrix} Q_1 & \\ & I_r \end{bmatrix}, \quad I_p \oplus Q_2 = \begin{bmatrix} I_p & \\ & Q_2 \end{bmatrix}, \quad Q_1 \oplus Q_2 = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix}, \tag{3.2.1}$$

are J–orthogonal. Moreover, considering the $J_{1,1}$–orthogonal matrix (see (A.3.3), (A.3.4))

$$\Theta_{1,1} = \frac{1}{\sqrt{1-\rho^2}} \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$

with $|\rho| < 1$, and

$$J_{1,1} = \begin{bmatrix} 1 & \\ & -1 \end{bmatrix},$$

the matrix of order $\alpha = p + q$

$$\frac{1}{\sqrt{1-\rho^2}} \begin{bmatrix} & & \text{i} & & \text{j} & \\ & & \downarrow & & \downarrow & \\ 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & \rho & \leftarrow \text{i} \\ & & & \ddots & & \\ & & \rho & & 1 & \leftarrow \text{j} \\ & & & & & \ddots \\ & & & & & & 1 \end{bmatrix}$$

is J–orthogonal, with $i \in \{1,\dots,p\}$ and $j \in \{p+1,\dots,q\}$. Moreover, the product of J–orthogonal matrices Θ_1 and Θ_2 is a J–orthogonal matrix too. Indeed, let $\Theta = \Theta_1\Theta_2$,

$$\begin{aligned}\Theta J\Theta^T &= \Theta_1\Theta_2 J(\Theta_1\Theta_2)^T \\ &= \Theta_1\Theta_2 J\Theta_2^T\Theta_1^T \\ &= \Theta_1 J\Theta_1^T \\ &= J.\end{aligned}$$

The reduction in proper form of U and V, is accomplished by applying $p-1$ and $q-1$ Givens rotations, respectively.

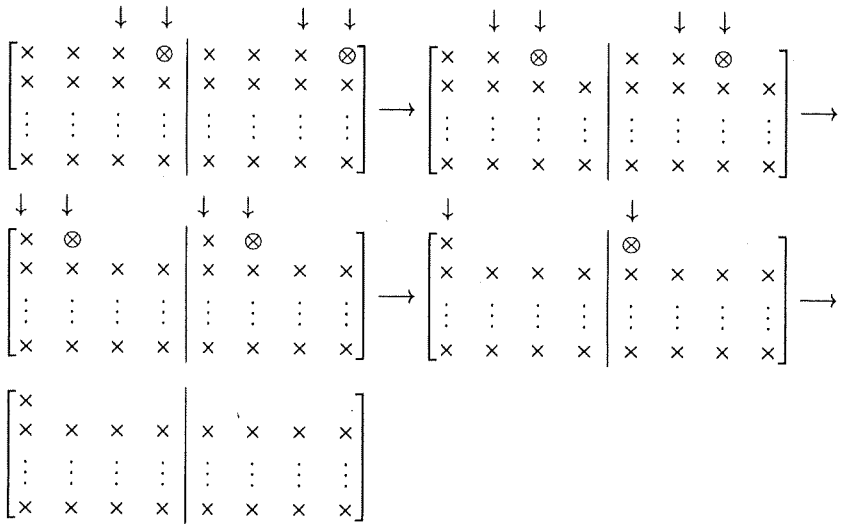


Figure 3.1. Reduction of a generator in proper form by means of a sequence of Givens and one hyperbolic rotation (positive step).

Finally, a generator in proper form is obtained by applying a hyperbolic rotation to the pivot column of U and V. The reduction is depicted in Fig. 3.1 and summarized in the following algorithm.

Algorithm 3.2.1 (Generalized Schur Algorithm) *Given* $G \in \mathbb{R}^{n\times\alpha}$,

`for` $i=1,2,\dots,n$:

1. `for` $j=p-1,p-2,\dots,1$

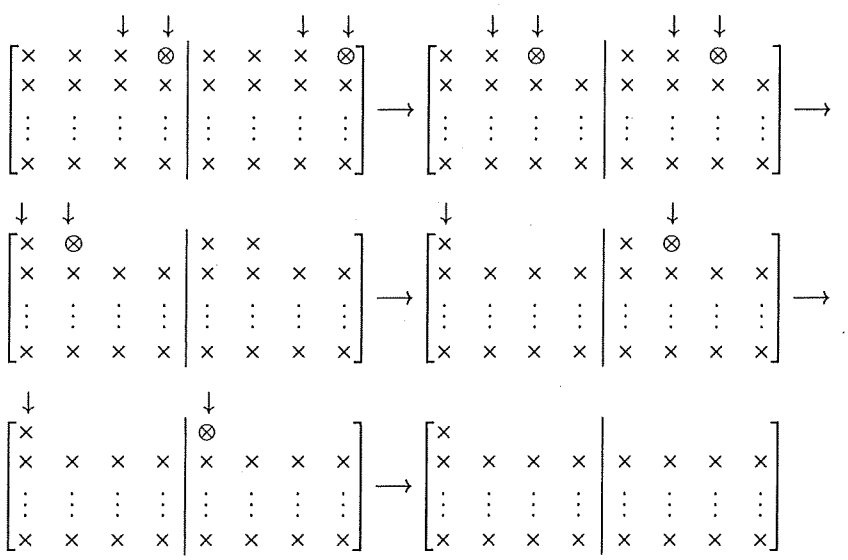


Figure 3.2. Reduction of a generator in proper form by means of a sequence of Givens and two hyperbolic rotations (positive step).

Apply a Givens rotation to the j–th and $(j+1)$th columns of G in order to annihilate $G(i, j+1)$.

end

2. for $j = \alpha - 1, \alpha - 2, \ldots, p+1$

   Apply a Givens rotation to the j–th and $(j+1)$th columns of G in order to annihilate $G(i, j+1)$.

   end

3. *Apply a hyperbolic rotation to the pivot columns* 1 *and* $p+1$ *in order to annihilate*

   either

   $$G(i, p+1) \qquad \text{if} \quad |G(i,1)| > |G(i,p+1)| \quad \text{(positive step)}$$

   or

   $$G(i,1) \qquad \text{if} \quad |G(i,1)| < |G(i,p+1)| \quad \text{(negative step).}$$

◇

We observe that the Givens rotations can be applied in a different order. Moreover, Givens transformations not involving the same columns can be performed in parallel. The stability of such an implementation has been studied in [119], [23]. It turns out that if the hyperbolic rotation is carefully implemented (see App. A), GSA is backward stable.
The reduction of a generator in proper form by means of a sequence of Givens matrices and two hyperbolic rotations is depicted in Fig. 3.2. Such an implementation of GSA is considered in Ch. 4 and Ch. 5 to preserve the structure of the involved generators, thereby further reducing the computational complexity. In the next section a proof of the stability of this particular implementation of GSA is given.

3.3 Stability analysis

The numerical stability analysis of GSA for computing the Cholesky decomposition of symmetric positive definite matrices of displacement rank 2 has been studied in [13]. In this particular case, each iteration of GSA involves only one hyperbolic rotation to the couple of generators. The stability of GSA, applied to matrices of displacement rank 2, follows directly from the numerical properties of hyperbolic rotations and it is shown that the GSA is backward stable. Moreover the stability of GSA is not dependent on the implementation of the hyperbolic transformations. This analysis can straightforwardly be extended to GSA for computing the LDL^T factorization of strongly regular matrices of displacement rank 2.

The numerical stability analysis of GSA to factorize strongly regular matrices of displacement rank $\alpha > 2$ has been studied in [119]. The conclusion drawn there is that GSA is backward stable provided it is implemented with hyperbolic transformations in factored form (which is well known to provide numerical stability in the context of Cholesky downdating [16]). If a more direct implementation of the hyperbolic transformations is used, then it is unstable. In this respect, the algorithm is analogous to Cholesky downdating; the details of implementation of the hyperbolic transformations are essential for stability.

Similar results have been derived in [23], where the hyperbolic rotations occurrring in GSA are computed by either the OD procedure (see App. A) or the H procedure. We are going to prove that GSA is still backward stable even when two hyperbolic rotations are performed at each iteration in a stable way, i.e., either in factored form [16], by means of the OD procedure or by means of the H procedure (see App. A) [23], [66].

3.3.1 Definitions and notations

Without loss of generality, we consider GSA to compute the LL^T factorization of a symmetric positive definite matrix

$$A \in \mathbb{R}^{n \times n} \tag{3.3.2}$$

of low displacement rank with respect to the shift matrix Z (2.5.11), where L is lower triangular. The same stability results can easily be derived for GSA for computing the LDL^T factorization of a strongly regular matrix. Let G be a generator for A. Define

$$G = \left[\begin{array}{cc|ccc} u_{1,1}^{(1)} & u_{1,2}^{(1)} & v_{1,1}^{(1)} & z_{1,1}^{(1)} & v_{1,2}^{(1)} \\ u_{2,1}^{(1)} & U_{2,2}^{(1)} & v_{2,1}^{(1)} & z_{2,1}^{(1)} & V_{2,2}^{(1)} \end{array}\right], \quad J = I_p \oplus -I_q, \; q = \alpha - p, \tag{3.3.3}$$

where $u_{1,1}^{(1)}, v_{1,1}^{(1)}, z_{1,1}^{(1)} \in \mathbb{R}$, $u_{2,1}^{(1)}, v_{2,1}^{(1)}, z_{2,1}^{(1)} \in \mathbb{R}^{n-1}$, $u_{1,2}^{(1)} \in \mathbb{R}^{1\times(p-1)}$, $v_{1,2}^{(1)} \in \mathbb{R}^{1\times(q-2)}$, $U_{2,2}^{(1)} \in \mathbb{R}^{(n-1)\times(p-1)}$, and $V_{2,2}^{(1)} \in \mathbb{R}^{(n-1)\times(q-2)}$.

The GSA requires n iterations to compute the factor L. Let $G_{0,Z} \triangleq G$. As already mentioned in the previous section, at the ith iteration, $i = 1, \ldots, n$, a J–orthogonal matrix Θ_i is chosen such that the ith row of $G_i = G_{i-1,Z}\Theta_i$ has all the elements equal to zero with the exception of a single pivot element in the first column (the first $i-1$ rows of G_i are zero). Then the first column of G_i becomes the ith column of L. The generator matrix $G_{i,Z}$ at the next iteration is given by

$$G_{i,Z}(:,1) = ZG_i(:,1),$$

$$G_{i,Z}(:,[2:\alpha]) = G_i(:,[2:\alpha]).$$

Without loss of generality, the matrices Θ_i, $i = 1, \ldots, n$, can be factored as the product of two hyperbolic rotations and an orthogonal one, i.e.,

$$\Theta_i = Q_i H_{i,2} H_{i,1},$$

where

$$Q_i = \left[\begin{array}{c|c} Q_{i,1} & \\ \hline & Q_{i,2} \end{array}\right],$$

with $Q_{i,1}$ and $Q_{i,2}$ orthogonal matrices of order p and q, such that

$$G_{i-1,Z}Q_i = \left[\begin{array}{cc|ccc} 0_{i-1} & 0_{i-1,p-1} & 0_{i-1} & 0_{i-1} & 0_{i-1,q-2} \\ u_{1,1}^{(i,1)} & 0_{1,p-1} & v_{1,1}^{(i,1)} & z_{1,1}^{(i,1)} & 0_{1,q-2} \\ u_{2,1}^{(i,1)} & U_{2,2}^{(i,1)} & v_{2,1}^{(i,1)} & z_{2,1}^{(i,1)} & V_{2,2}^{(i,1)} \end{array}\right] \tag{3.3.4}$$

respectively, and

$$H_{i,1} = \frac{1}{\sqrt{1-\rho_{i,1}^2}} \left[\begin{array}{cccc} 1 & 0_{p-1}^T & \rho_{i,1} & 0_{q-1}^T \\ 0_{p-1} & I_{p-1} & 0 & 0_{p-1,q-1} \\ \rho_{i,1} & 0_{p-1}^T & 1 & 0_{q-1}^T \\ 0_{q-1} & 0_{q-1,p-1} & 0_{q-1} & I_{q-1} \end{array}\right],$$

$$H_{i,2} = \frac{1}{\sqrt{1-\rho_{i,2}^2}} \left[\begin{array}{cccc} 1 & 0_{p}^T & \rho_{i,2} & 0_{q-2}^T \\ 0_{p} & I_{p} & 0 & 0_{p,q-2} \\ \rho_{i,2} & 0_{p}^T & 1 & 0_{q-2}^T \\ 0_{q-2} & 0_{q-2,p} & 0_{q-2} & I_{q-2} \end{array}\right],$$

with $|\rho_{i,1}|, |\rho_{i,2}| < 1$, such that

$$G_i = G_{i-1,Z} Q_i H_{i,2} H_{i,1} = \left[\begin{array}{cc|ccc} 0_{i-1} & 0_{i-1,p-1} & 0_{i-1} & 0_{i-1} & 0_{i-1,q-2} \\ u_{1,1}^{(i,2)} & 0_{1,p-1} & 0 & 0 & 0_{1,q-2} \\ u_{2,1}^{(i,2)} & U_{2,2}^{(i,2)} & v_{2,1}^{(i,2)} & z_{2,1}^{(i,2)} & V_{2,2}^{(i,2)} \end{array}\right], \quad (3.3.5)$$

where $0_{r,s}$ denotes the rectangular zero matrix with r rows and s columns. As mentioned in §1, the computation of the hyperbolic rotations in a stable way is crucial for the stability of the algorithm. For implementation details of hyperbolic rotations in factored form see [16], [119]. In the next section we will show that this variant of the GSA is stable, provided in each iteration the J–orthogonal matrix is computed as previously described, and the hyperbolic rotations are implemented in factored form. Similar stability results hold considering either the H–procedure or the OD–procedure to implement the hyperbolic rotations [23], [66].

3.3.2 Stability Analysis of GSA

A stability analysis of the GSA with a single hyperbolic rotation in factored form per iteration is presented in [119]. The stability analysis for the algorithm described in the previous section can be done in a similar way. It is split up into two parts : one which shows how local errors propagate through the algorithm and one which bounds the local errors. We consider the same notation as introduced in §2 but denote by the superscript the corresponding quantities as stored in the computer. Hence

$$\tilde{G}_i = \left[\tilde{u}_i \quad \tilde{U}_i \,\middle|\, \tilde{v}_i \quad \tilde{z}_i \quad \tilde{V}_i\right].$$

The local errors, generated by computing $\tilde{G}_{i+1}$ by means of orthogonal and hyperbolic transformations, are given by

$$\epsilon F_i = \tilde{G}_{i+1} J \tilde{G}_{i+1}^T - \tilde{G}_{i,Z} J \tilde{G}_{i,Z}^T + O(\epsilon^2), \quad i = 1, \ldots, n-1, \quad (3.3.6)$$

where ϵ is the machine precision.

In [119] it is proved that

$$A - \tilde{L}\tilde{L}^T = \sum_{j=0}^{n-1} Z_j (GJG^T - \tilde{G}J\tilde{G}^T) Z_j^T - \epsilon \sum_{j=0}^{n-1} \sum_{k=1}^{n-j-1} Z_j F_k Z_j^T + O(\epsilon^2), \quad (3.3.7)$$

where $Z_j \triangleq Z^j$ and $\tilde{R}$ is the computed Cholesky factor. This means that if the errors in the computation of the initial generator matrix and the local errors are bounded, the algorithm is stable. The error in the initial generator matrix is not a problem, since often it is explicitly known or can be computed in a backward stable way [53]. Below, we assume that the initial generator matrix is computed exactly and restrict ourselves to the effects of local errors due to the orthogonal and hyperbolic transformations.

Because any bounds on the errors produced by the transformations will depend on the norm of the generators, it is essential to bound the generators.

Theorem 3.3.1: *Let A be the symmetric positive definite matrix defined in (3.3.2) and let G the generator matrix defined in (3.3.3). When the generators are computed by applying a block diagonal orthogonal matrix and two hyperbolic transformations, they satisfy*

$$\|G_i\|_F \leq 2\sqrt{i-1}\|A\|_F + \|G\|_F, \tag{3.3.8}$$

with G_i defined in (3.3.5).

Proof: Let $\hat{u}_i, \hat{v}_i$ and $\hat{z}_i$ be the generator vectors in (3.3.4) that will be modified by the two hyperbolic rotations $H_{i,2}$ and $H_{i,1}$,

$$\begin{aligned}
\begin{bmatrix} u_i & v_i & z_i \end{bmatrix} &= \tfrac{1}{\delta_{i,1}\delta_{i,2}} \begin{bmatrix} \hat{u}_i & \hat{v}_i & \hat{z}_i \end{bmatrix} \begin{bmatrix} 1 & \rho_{i,1} & 0 \\ \rho_{i,1} & 1 & 0 \\ 0 & 0 & \delta_{i,1} \end{bmatrix} \begin{bmatrix} 1 & 0 & \rho_{i,2} \\ 0 & \delta_{i,2} & 0 \\ \rho_{i,2} & 0 & 1 \end{bmatrix} \\
&= \tfrac{1}{\delta_{i,1}\delta_{i,2}} \begin{bmatrix} \hat{u}_i & \hat{v}_i & \hat{z}_i \end{bmatrix} \begin{bmatrix} 1 & \rho_{i,1}\delta_{i,2} & \rho_{i,2} \\ \rho_{i,1} & \delta_{i,2} & \rho_{i,1}\rho_{i,2} \\ \rho_{i,2}\delta_{i,1} & 0 & \delta_{i,1} \end{bmatrix}
\end{aligned}$$

where $\delta_{i,k} = \sqrt{1-\rho_{i,k}^2}$, $k = 1, 2$. Then we have

$$\begin{aligned}
\begin{bmatrix} u_i & v_i & z_i \end{bmatrix} &= \frac{1}{\delta_{i,1}\delta_{i,2}} \begin{bmatrix} \hat{u}_i & \hat{v}_i & \hat{z}_i \end{bmatrix} \begin{bmatrix} 1 & \rho_{i,1}\delta_{i,2} & \rho_{i,2} \\ \rho_{i,1} & \rho_{i,1}^2\delta_{i,2} & \rho_{i,1}\rho_{i,2} \\ \rho_{i,2}\delta_{i,1} & \rho_{i,1}\rho_{i,2}\delta_{i,1}\delta_{i,2} & \rho_{i,2}^2\delta_{i,1} \end{bmatrix} \\
&\quad + \frac{1}{\delta_{i,1}\delta_{i,2}} \begin{bmatrix} \hat{u}_i & \hat{v}_i & \hat{z}_i \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \delta_{i,1}^2\delta_{i,2} & 0 \\ 0 & -\rho_{i,1}\rho_{i,2}\delta_{i,1}\delta_{i,2} & \delta_{i,1}\delta_{i,2}^2 \end{bmatrix} \\
&= \begin{bmatrix} u_i & \rho_{i,1}\delta_{i,2}u_i & \rho_{i,2}u_i \end{bmatrix} + \begin{bmatrix} \hat{u}_i & \hat{v}_i & \hat{z}_i \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \delta_{i,1} & 0 \\ 0 & -\rho_{i,1}\rho_{i,2} & \delta_{i,2} \end{bmatrix}.
\end{aligned}$$

Consider the Givens rotations

$$U = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_l & s_l \\ 0 & -s_l & c_l \end{bmatrix}, \quad \begin{cases} c_l = \rho_{i,2}/\sqrt{1-\delta_{i,1}^2\delta_{i,2}^2} \\ \\ s_l = \rho_{i,1}\delta_{i,2}/\sqrt{1-\delta_{i,1}^2\delta_{i,2}^2} \end{cases}$$

$$V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_r & s_r \\ 0 & -s_r & c_r \end{bmatrix}, \quad \begin{cases} c_r = \rho_{i,2}\delta_{i,1}/\sqrt{1-\delta_{i,1}^2\delta_{i,2}^2} \\ \\ s_r = \rho_{i,1}/\sqrt{1-\delta_{i,1}^2\delta_{i,2}^2} \end{cases}.$$

Then

$$\begin{bmatrix} u_i & v_i & z_i \end{bmatrix} U = \begin{bmatrix} u_i & \rho_{i,1}\delta_{i,2}u_i & \rho_{i,2}u_i \end{bmatrix} U$$

$$
\begin{aligned}
&+ \begin{bmatrix} \hat{u}_i & \hat{v}_i & \hat{z}_i \end{bmatrix} VV^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & \delta_{i,1} & 0 \\ 0 & -\rho_{i,1}\rho_{i,2} & \delta_{i,2} \end{bmatrix} U \\
&= \begin{bmatrix} u_i & 0 & \sqrt{1-\delta_{i,1}^2\delta_{i,2}^2}u_i \end{bmatrix} + \begin{bmatrix} \tilde{u}_i & \tilde{v}_i & \tilde{z}_i \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \delta_{i,1}\delta_{i,2} \end{bmatrix} \\
&= \begin{bmatrix} u_i & \tilde{v}_i & \sqrt{1-\delta_{i,1}^2\delta_{i,2}^2}u_i + \delta_{i,1}\delta_{i,2}\tilde{z}_i \end{bmatrix},
\end{aligned}
$$

where

$$
\begin{bmatrix} \tilde{u}_i & \tilde{v}_i & \tilde{z}_i \end{bmatrix} = \begin{bmatrix} \hat{u}_i & \hat{v}_i & \hat{z}_i \end{bmatrix} V.
$$

Then

$$
\begin{aligned}
\left\| \begin{bmatrix} u_i & v_i & z_i \end{bmatrix} U \right\|_F^2 &= \left\| \begin{bmatrix} u_i & v_i & z_i \end{bmatrix} \right\|_F^2 \\
&= \left\| \begin{bmatrix} u_i & \tilde{v}_i & \sqrt{1-\delta_{i,1}^2\delta_{i,2}^2}u_i + \delta_{i,1}\delta_{i,2}\tilde{z}_i \end{bmatrix} \right\|_F^2 .
\end{aligned}
$$

Applying the inequality

$$
\left\| \begin{bmatrix} u_i & \tilde{z}_i \end{bmatrix} \begin{bmatrix} \sqrt{1-\alpha^2} \\ \alpha \end{bmatrix} \right\|_2^2 \leq \left\| \begin{bmatrix} u_i \; \tilde{z}_i \; \end{bmatrix} \right\|_2^2
$$

with $\alpha = \delta_{i,1}\delta_{i,2}$, $|\alpha| \leq 1$, we finally obtain

$$
\begin{aligned}
\left\| \begin{bmatrix} u_i & v_i & z_i \end{bmatrix} \right\|_F^2 &\leq 2\|u_i\|_2^2 + \left\| \begin{bmatrix} \tilde{v}_i & \tilde{z}_i \end{bmatrix} \right\|_F^2 \\
&= 2\|u_i\|_2^2 + \left\| \begin{bmatrix} \hat{v}_i & \hat{z}_i \end{bmatrix} \right\|_F^2 \\
&\leq 2\|u_i\|_2^2 + \left\| \begin{bmatrix} \hat{u}_i & \hat{v}_i & \hat{z}_i \end{bmatrix} \right\|_F^2 .
\end{aligned}
$$

Since the orthogonal transformations do not affect the norm of the generators and $\|Z\|_2 = 1$, then

$$
\|G_i\|_F^2 \leq 2\|u_i\|^2 + \|G_{i-1}\|_F^2,
$$

and recursively we have

$$
\begin{aligned}
\|G_i\|_F^2 &\leq 2\sum_{j=1}^{i} \|u_j\|_2^2 + \|G\|_F^2 \\
&= 2\|R(1:k,:)\|_F^2 + \|G\|_F^2.
\end{aligned}
$$

Then (3.3.8) follows since, for an arbitrary positive semi–definite, rank $i-1$ matrix with a factorization $A = LL^T$, (see [119]),

$$
\|R\|_F^2 \leq \sqrt{i}\|A\|_F^2
$$

This follows from the fact that the squared Frobenius norm equals the sum of the squares of the singular values and from a standard inequality relating the vector 2–norm and the vector 1–norm.

$\Diamond$

To complete the stability analysis we need to show that the orthogonal and hyperbolic transformations, applied in factored form, produce a local error, ϵF_i, which is proportional to the norm of the generator matrix. An error analysis of hyperbolic transformations applied in factored form (see App. A) is given in [16]. Denoting by

$$H_{i,j} = \begin{bmatrix} 1 & 0 \\ \rho_{i,j} & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{1-\rho_{i,j}^2}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \rho_{i,j} \\ 0 & \sqrt{1-\rho_{i,j}^2} \end{bmatrix}, \quad j = 1, 2,$$

the hyperbolic transformations applied in factored form, yield:

$$\begin{bmatrix} \tilde{u}_{1,i+1} + \widehat{\Delta u} & \tilde{v}_{1,i+1} & \tilde{z}_{1,i+1} \end{bmatrix} = H_{i,2} H_{i,1} \begin{bmatrix} \hat{u}_{1,i} & \hat{v}_{1,i} + \widehat{\Delta v} & \hat{z}_{1,i} + \widehat{\Delta z} \end{bmatrix}. \quad (3.3.9)$$

The mixed error vectors $\widehat{\Delta u}$, $\widehat{\Delta v}$ and $\widehat{\Delta z}$ satisfy

$$\left\| \begin{bmatrix} \widehat{\Delta u} & \widehat{\Delta v} & \widehat{\Delta z} \end{bmatrix} \right\|_F \leq 12.5\epsilon \left\| \begin{bmatrix} \tilde{u}_{1,i+1} & \hat{v}_{1,i} & \hat{z}_{1,i} \end{bmatrix} \right\|_F, \quad (3.3.10)$$

where ϵ is the roundoff unit. Furthermore, concerning the application of the orthogonal transformations, it can be proved (see [119], [135]) that there exist orthogonal matrices $\hat{Q}_{i,1}$ and $\hat{Q}_{i,2}$ such that

$$\begin{aligned} &\left[\tilde{u}_{1,i} + \Delta u_1 \quad \tilde{U}_{2,i} + \Delta U_2 \;\middle|\; \tilde{v}_{1,i} + \Delta v_1 \quad \tilde{z}_{1,i} + \Delta z_1 \quad \tilde{V}_{2,i} + \Delta V_2 \right] \left[\begin{array}{c|c} \hat{Q}_{i,1} & \\ \hline & \hat{Q}_{i,2} \end{array} \right] \\ &\quad = \left[\hat{u}_{1,i} \quad \hat{U}_{2,i} \;\middle|\; \hat{v}_{1,i} \quad \hat{z}_{1,i} \quad \hat{V}_{2,i} \right] \\ &\quad = \hat{G}_i, \end{aligned} \quad (3.3.11)$$

where, for $m = \max\{p, q-1\}$,

$$\left\| \begin{bmatrix} \Delta u_1 & \Delta U_2 \end{bmatrix} \right\|_F \leq 6m\epsilon \left\| \begin{bmatrix} Z\tilde{u}_{1,i} & \tilde{U}_{2,i} \end{bmatrix} \right\|_F$$

and

$$\left\| \begin{bmatrix} \Delta v_1 & \Delta z_1 & \Delta V_2 \end{bmatrix} \right\|_F \leq 6m\epsilon \left\| \begin{bmatrix} Z\tilde{v}_{1,i} & Z\tilde{z}_{1,i} & \tilde{V}_{2,i} \end{bmatrix} \right\|_F.$$

Let

$$\Delta G_i = [\Delta u_1 \; \Delta U_2 \; \Delta v_1 \; \Delta z_1 \; \Delta V_2],$$

then

$$\|\Delta G_i\|_F \leq 6m\epsilon \|G_{i,Z}\|_F \leq 6m\epsilon \|G_i\|_F.$$

Analogously, let

$$\widehat{\Delta G_i} = \begin{bmatrix} \widehat{\Delta u} & \widehat{\Delta v} & \widehat{\Delta z} \end{bmatrix},$$

then the error bounds (3.3.9) and (3.3.11) can be used to show that

$$\hat{G}_i J \hat{G}_i^T = (\tilde{G}_{i,Z} + \Delta G_i) J (\tilde{G}_{i,Z} + \Delta G_i)^T,$$

and

$$\begin{aligned}(\tilde{G}_{i+1}\widehat{\Delta u}\, e_1^T) J (\tilde{G}_{i+1} + \widehat{\Delta u}\, e_1^T)^T \;=\; & (\hat{G}_i + e_{p+1}\widehat{\Delta v}^T + \widehat{\Delta z}\, e_{p+2}^T) J \\ & \times (\hat{G}_i + \widehat{\Delta v}\, e_{p+1}^T + \widehat{\Delta z}\, e_{p+2}^T)^T,\end{aligned}$$

where e_1, e_{p+1} and e_{p+2} are standard basis vectors. Then

$$\begin{aligned}\epsilon F_i \;=\; & \tilde{G}_{i,Z} J \Delta G_i^T + \Delta G_i J \tilde{G}_{i,Z}^T \\ & - \begin{bmatrix}\tilde{u}_{1,i+1} & \hat{v}_{1,i} & \hat{z}_{1,i}\end{bmatrix} \widehat{\Delta G_i}^T - \widehat{\Delta G_i} \begin{bmatrix}\tilde{u}_{1,i+1} & \hat{v}_{1,i} & \hat{z}_{1,i}\end{bmatrix}^T,\end{aligned}$$

corresponding to the bound

$$\begin{aligned}\|\epsilon F_i\|_F \;&\leq\; 2\|\tilde{G}_{i,Z}\|_F \|\Delta G_i\|_F + 2\left(\|\hat{G}_i\|_F + \|\tilde{G}_{i+1}\|_F\right)\|\widehat{\Delta G_i}\|_F \\ &\leq\; 12m\epsilon\|\tilde{G}_{i,Z}\|_F^2 + 25\epsilon\left(\|\hat{G}_i\|_F + \|\tilde{G}_{i+1}\|_F\right)^2 \\ &\leq\; 12m\epsilon\|G_i\|_F^2 + 25\epsilon(\|G_i\|_F + \|G_{i+1}\|_F)^2 + O(\epsilon^2).\end{aligned}$$

By Theorem 3.3.1, $\|G_i\|_F^2, \|G_{i+1}\|_F^2 \leq 2\sqrt{i}\|A\|_F + \|G\|_F^2$, the following bound holds

$$\|\epsilon F_i\|_F \leq (12m + 100)\epsilon\left(2\sqrt{i}\|A\|_F + \|G\|_F^2\right).$$

From (3.3.7), we have $\|A - \tilde{R}^T\tilde{R}\|_F \leq (6m + 50)(n-1)n\epsilon\left(2\sqrt{n}\|A\|_F + \|G\|_F^2\right)$.

3.4 Conclusions

Fast and efficient algorithms for solving Structured Total Least Squares problems [93], [80] are based on a particular implementation of the GSA requiring two hyperbolic transformations at each iteration.

In this chapter the stability of such an implementation is discussed. The stability result is dependent on the particular implementation of the hyperbolic transformations. It is proved that if the hyperbolic transformations are performed in factored form, the considered implementation is as stable as the implementation studied in [119] which requires only one hyperbolic transformation at each iteration.

Chapter 4

FAST ALGORITHM FOR SOLVING THE HANKEL/TOEPLITZ STRUCTURED TOTAL LEAST SQUARES PROBLEM

4.1 Introduction

The structured total least squares problem has already been introduced in Ch. 1. This chapter focuses on fast and reliable algorithms for the following STLS problem,

$$\begin{aligned} &\min_{\Delta A, \Delta b, x} \|[\Delta A \ \Delta b]\|_X^2 \\ &\text{such that } (A + \Delta A)x = b + \Delta b, \\ &\text{and } [\Delta A \ \Delta b] \text{ has the same linear structure as } [A \ b], \end{aligned} \tag{4.1.1}$$

where $\|.\|_X$ is an appropriate 2-norm defined on the entries of $[\Delta A \ \Delta b]$ and

$$[A \ b] = \begin{bmatrix} s_{n+1} & s_n & \cdots & s_2 & s_1 \\ s_{n+2} & s_{n+1} & \ddots & \ddots & s_2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ s_{m+n} & s_{m+n-1} & \cdots & s_{m+1} & s_m \end{bmatrix}. \tag{4.1.2}$$

At the same time, this covers the STLS problem in which $[A \ b]$ and $[\Delta A \ \Delta b]$ have a Hankel structure, since inverting the row order of $[A \ b]$ and $[\Delta A \ \Delta b]$ turns a Hankel (Toeplitz) STLS problem[1] into a Toeplitz (Hankel) STLS problem, without changing the solution for x. As already seen in Ch. 1, the latter structure occurs frequently in many signal processing applications such as NMR spectroscopic data quantitation [128], analysis of acoustic resonances [2], autoregressive modelling [129],...

Fast algorithms for the STLS problem

$$\begin{aligned} &\min_{\Delta A, \Delta b, x} \|[\Delta A \ \Delta b]\|_X^2 \\ &\text{such that } (A + \Delta A)x = b + \Delta b, \\ &\text{and } \Delta A \text{ has the same linear structure as } A \ , \end{aligned}$$

will be handled in Ch. 5.

The chapter is organized as follows. In the next section we describe two basic algorithms for solving general STLS problems: STLS1 and STLS2. A fast implementation of algorithm STLS2 is developed in section 4.3 for the particular STLS problem under consideration, namely a Toeplitz STLS problem with a frequently used 2-norm $\|.\|_X$, that will be specified in the next section. By exploiting the low displacement rank of the matrices involved in the basic algorithm STLS2 and by taking advantage of the sparsity of the corresponding generators, we obtain an implementation that requires $O(mn + n^2)$ flops per iteration, compared to $O(m^3)$ or $O((m + n)^3)$ flops per iteration for straightforward implementations of STLS1

[1] This is a shorthand notation indicating a STLS problem in which $[A \ b]$ and $[\Delta A \ \Delta b]$ are Hankel (Toeplitz) matrices.

and STLS2.
In section 4.4 the increased efficiency of the proposed fast implementation is compared to the efficiency of the two straightforward implementations (which do not exploit the structure present at the algorithmic level in the problem at hand). The comparison is done on a recently introduced speech compression scheme in which the considered STLS problem constitutes one of the kernel problems. By changing the parameters of the compression scheme, the theoretically obtained dependence of the number of flops on the dimensions m and n is verified for the fast implementation.

4.2 Basic algorithms

Before describing the 2 basic algorithms, we reformulate (4.1.1) as follows:

$$\min_{\Delta s, x} 0.5 \Delta s^T W \Delta s \quad \text{such that } (A + \Delta A)x = b + \Delta b, \tag{4.2.3}$$

where $\Delta s \in \mathbb{R}^{k \times 1}$ contains the k *different* elements of $[\Delta A \ \Delta b]$ and $W \in \mathbb{R}^{k \times k}$ is a weighting matrix. Since in this chapter the Toeplitz STLS problem is considered, we have –in addition to the Toeplitz structure of $[A \ b]$– that

$$[\Delta A \ \Delta b] = \begin{bmatrix} \Delta s_{n+1} & \Delta s_n & \dots & \Delta s_2 & \Delta s_1 \\ \Delta s_{n+2} & \Delta s_{n+1} & \ddots & \ddots & \Delta s_2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \Delta s_{m+n} & \Delta s_{m+n-1} & \dots & \Delta s_{m+1} & \Delta s_m \end{bmatrix}, \tag{4.2.4}$$

where $\Delta s \in \mathbb{R}^{(m+n) \times 1}$ contains the corrections to the samples in the observed data vector s.
In most signal processing applications $\|\Delta s\|_2^2$ has to be minimized and therefore we choose $W = I_{m+n}$ in (4.2.3), where $I_k \in \mathbb{R}^{k \times k}$ represents the $k \times k$ identity matrix. We have thus completely specified $\|.\|_X$ in (4.1.1).
As can be seen from (4.2.3) the STLS problem is a constrained optimization problem with a quadratic objective function and nonlinear constraints. Therefore, taking into account the nature of the STLS problem, it should come as no surprise that all the algorithms for solving it will be iterative.
The first basic algorithm described here follows the same lines as the heuristic algorithm developed in [107] for a similar (namely A Toeplitz but b unstructured) STLS problem: in each iteration the equality constraints of (4.2.3) are linearized around the current solution point (a solution point being determined by Δs and x). The Toeplitz STLS problem considered here is also treated in [107], but in that paper a penalty function approach is proposed. The latter means that the constrained optimization problem (4.2.3) is transformed into an unconstrained optimization problem, simply by adding the weighted constraints to the original objective function, resulting in an unconstrained optimization problem (see e.g. [51],

[46]). However, for the unconstrained problem to be equivalent to the constrained problem, large weights need to be introduced, yielding ill-conditioned matrices and thus inaccurate results. Moreover, a good choice of the weight is not trivial, since it depends very much on the numerical values of the problem at hand, making an automated approach difficult. We therefore stick to the constrained optimization formulation of (4.2.3) and the above mentioned linearization of the constraints.
Before describing the algorithm, we introduce some notation. Let us represent small perturbations on Δs and x by $\Delta\tilde{s} \in \mathbb{R}^{(m+n)\times 1}$ and $\Delta\tilde{x} \in \mathbb{R}^{n\times 1}$ respectively. Furthermore let

$$r(\Delta s, x) = (A + \Delta A)x - b - \Delta b$$

and $X \in \mathbb{R}^{m\times(m+n)}$ is defined by

$$X\Delta s = [\Delta A\ \Delta b] \begin{bmatrix} x \\ -1 \end{bmatrix}.$$

When $[\Delta A\ \Delta b]$ is Toeplitz this yields

$$X = \begin{bmatrix} -1 & x_n & \cdots & x_1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & x_n & \cdots & x_1 & 0 & & \vdots \\ \vdots & & \ddots & \ddots & & \ddots & & \vdots \\ 0 & & & \ddots & \ddots & & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & -1 & x_n & \cdots & x_1 \end{bmatrix}.$$

The iterative algorithm is obtained by replacing in (4.2.3) Δs and x by $\Delta s + \Delta\tilde{s}$ and $x + \Delta\tilde{x}$ respectively, followed by a linearization of the constraints around the current solution point $[\Delta s^T\ x^T]^T$ (simply omit second order terms that occur). We then obtain the following algorithm:

Algorithm 4.2.1 (Algorithm STLS1)

Input: $[A\ b] \in \mathbb{R}^{m\times(n+1)}$, $W \in \mathbb{R}^{(m+n)\times(m+n)}$
Output: *the parameter vector* $x \in \mathbb{R}^{n\times 1}$ *and* $\Delta s \in \mathbb{R}^{(m+n)\times 1}$ *(i.e. the minimal representation of the matrix* $[\Delta A\ \Delta b]$*)*

Step 1: Initialize Δs *and* x

Step 2: `while` *stopcriterion not satisfied*

Step 2.1: Solve the following equality constrained LS problem:

$$\min_{\Delta\tilde{s},\Delta\tilde{x}} \|W^{1/2}(\Delta s + \Delta\tilde{s})\|_2^2$$

$$\textit{such that } r(\Delta s, x) + J \begin{bmatrix} \Delta\tilde{s} \\ \Delta\tilde{x} \end{bmatrix} = 0$$

Step 2.2: $\Delta s \leftarrow \Delta s + \Delta\tilde{s}$

$$x \leftarrow x + \Delta\tilde{x}$$

end

$\Diamond$

where $J = [X \;\; A + \Delta A]$ is the Jacobian of the constraints $r(\Delta s, x)$ w.r.t. $v \triangleq [\Delta s^T \;\; x^T]^T$. A discussion on the initialization of Δs and x is beyond the scope of this chapter. It should be noted however that problem (4.2.3) is a nonlinear optimization problem that suffers from many local minima, a problem that becomes more important with increasing dimensions of the problem. Therefore, a good initialization procedure is of utmost importance [78], especially when a qualitive result means finding the global minimum. Since this chapter is mainly about the computational aspects of Step 2 of the above algorithm, we use a simple initialization method, namely the initial parameter vector is obtained as the LS solution of $Ax \approx b$ and Δs is set to 0.
The choice of the stop criterion depends on the application at hand. In the remainder of the chapter the following stop criterion is used:

$$||\Delta s||_2 < tol.$$

For solving the equality constrained LS problem in Step 2.1 of Algorithm STLS1, we make use of the Generalized RQ (GRQ) factorization [6], [5]. The latter GRQ factorization is a 2 step procedure consisting of one RQ factorization followed by a QR factorization. For the RQ factorization a fast implementation exploiting the low displacement rank of the involved matrices could be developed. However, the consecutive QR factorization does not operate on one of the originally structured matrices but on one of the original matrices multiplied with an orthogonal matrix to the right. Since the latter matrix is typically unstructured, it is not possible to improve the performance of this second step. We will however use this algorithm in the comparison of section 4.4, since this algorithm is in fact a general algorithm that with only minor modifications can deal with any type of linearly structured STLS problem. The price to pay for this generality is the decrease in efficiency.
In order to be able to exploit the structure of the Toeplitz STLS problem we follow a different approach. Instead of eliminating the constraints of the STLS problem (4.2.3), we apply the Newton method for unconstrained optimization to the Lagrangian L of problem (4.2.3):

$$L(\Delta s, x, \lambda) = 0.5\Delta s^T W \Delta s - \gamma^T (b - Ax - X\Delta s),$$

where $\gamma \in \mathbb{R}^{m\times 1}$ is a vector of Lagrange multipliers. The straightforward application of the Newton method (see e.g. [46], [51]) on the Lagrangian L (for $W = I_{m+n}$)

yields the second basic algorithm:

Algorithm 4.2.2 (Algorithm STLS2)

Input: $[A\ b] \in \mathbb{R}^{m\times(n+1)}$
Output: *the parameter vector* $x \in \mathbb{R}^{n\times 1}$ *and* $\Delta s \in \mathbb{R}^{(m+n)\times 1}$ *(i.e. the minimal representation of the Toeplitz matrix* $[\Delta A\ \Delta b]$*)*

Step 1: Initialize Δs,x *and* γ

Step 2: `while` *stopcriterion not satisfied*

Step 2.1: Solve the following system of equations:

$$\begin{bmatrix} S & J^T \\ J & 0 \end{bmatrix} \begin{bmatrix} \Delta\tilde{s} \\ \Delta\tilde{x} \\ \Delta\tilde{\gamma} \end{bmatrix} = - \begin{bmatrix} g + J^T\gamma \\ r(\Delta s, x) \end{bmatrix}$$

Step 2.2: $\Delta s \leftarrow \Delta s + \Delta\tilde{s}$

$x \leftarrow x + \Delta\tilde{x}$

$\gamma \leftarrow \gamma + \Delta\tilde{\gamma}$

`end`

$\Diamond$

where $g = \begin{bmatrix} W & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta s \\ \Delta x \end{bmatrix} \in \mathbb{R}^{(m+2n)\times 1}$ is the gradient of the objective function in (4.2.3) and $J = [X\ A+\Delta A]$ is the Jacobian of the constraints $r(\Delta s, x)$ in (4.2.3), all w.r.t. $v \triangleq [\Delta s^T\ x^T]^T$. Furthermore, for optimal convergence rate (superlinear) the matrix S should be set to $\nabla^2_{vv}L(\Delta s, x, \gamma)$. The latter means that in this case

$$S = \begin{bmatrix} I_{m+n} & 0 \\ 0 & 0 \end{bmatrix} - \sum_{i=1}^{m} \gamma_i \nabla^2_{vv} r. \tag{4.2.5}$$

As shown in e.g. [46], S can also be chosen to be a positive definite approximation of $\nabla^2_{vv}L$, without changing the final solution of problem (4.2.3). Inclusion of the second term in (4.2.5) would render the structure of S rather complicated. We therefore only retain the first term in (4.2.5):

$$S = \begin{bmatrix} I_{m+n} & 0 \\ 0 & 0 \end{bmatrix}. \tag{4.2.6}$$

It is interesting to notice that Step 2.1 of algorithm STLS1 is basically the same as Step 2.1 of algorithm STLS2 (when the above approximation of S (see (4.2.6))

is used), although both algorithms are derived from a different starting point. The proof is easily obtained by applying the method of Lagrange multipliers to the subproblem described in Step 2.1 of algorithm STLS1. The system of equations that results from it is exactly the same as the system of equations that needs to be solved in Step 2.1 of algorithm STLS2 with the approximation of S described in (4.2.6). The latter observation also yields some additional insight in the convergence properties that can be expected[2] for algorithms STLS1 and STLS2. By omitting the curvature of the constraints in (4.2.5) the convergence rate is no longer superlinear but linear (see [46], [51]). However, this is largely compensated by the fact that one iteration can be implemented in a very fast way, exploiting the low displacement rank structure of the matrices involved in Step 2.1 of Algorithm STLS2 and by taking advantage of the sparsity of the corresponding generators.

4.3 Fast implementation of STLS2

In this section we consider the structure at the second level, i.e., the algorithmic level. A fast implementation will be developed for the kernel problem (i.e. Step 2.1) of algorithm STLS2. The latter problem is a system of linear equations:

$$Mz = b_1, \tag{4.3.7}$$

with

$$M = \begin{bmatrix} I_{(m+n)\times(m+n)} & 0_{(m+n)\times n} & X^T \\ 0_{n\times(m+n)} & 0_{n\times n} & \Lambda^T \\ X & \Lambda & 0_{m\times m} \end{bmatrix}, \tag{4.3.8}$$

where $\Lambda = A + \Delta A \triangleq toeplitz(\lambda(n : m+n-1), \lambda(n : -1 : 1))$, with $toeplitz(c, r)$ a shorthand notation for the Toeplitz matrix having c as its first column and r as its first row. The solution of (4.3.7) can be obtained by computing the LDL^T factorization of M, where L is lower triangular and D is a signature matrix[3] of the LDL^T factorization. The solution is then found by solving the following linear systems:

$$\begin{aligned} Lz_2 &= b_1 \\ Dz_1 &= z_2 \\ L^T z &= z_1. \end{aligned} \tag{4.3.9}$$

The latter factorization can be obtained in a fast way by an appropriate implementation of the generalized Schur algorithm. As will be described in the next subsection, a high computational efficiency is obtained by exploiting the low displacement rank of the Toeplitz-block-like matrix M and by taking advantage of the sparsity of the corresponding generators.

[2]The same conclusion applies to the similar but different STLS problem described in [107].

[3]In Ch. 2 we have described the generalized Schur algorithm for computing the LDL^T factorization of a structured matrix, with the elements in the main diagonal of L equal to 1. The generalized Schur algorithm for computing the LDL^T factorization of a structured matrix, where D is the signature matrix, can straightforwardly be derived from algorithm 2.3.1.

4.3.1 The generalized Schur algorithm applied to M

Before applying the generalized Schur algorithm to M we observe that the matrix M is not strongly regular. In fact $\det(M(1:i,1:i)) = 0, i = m+n+1,\ldots,m+2n$. Hence a permutation matrix P is considered in order to transform M into the Toeplitz–block matrix $\tilde{K}$, i.e.,

$$\tilde{K} = PMP^T = \begin{bmatrix} I_{(m+n)\times(m+n)} & X^T & 0_{(m+n)\times n} \\ X & 0_{m\times m} & \Lambda \\ 0_{n\times(m+n)} & \Lambda^T & 0_{n\times n} \end{bmatrix}.$$

It is easy to prove that $\tilde{K}$ is strongly regular. Let

$$\Phi \triangleq Z_{m+n} \oplus Z_m \oplus Z_n,$$

where Z_* is the shift matrix of order $*$. The displacement of $\tilde{K}$ with respect to Φ,

$$\nabla_\Phi \tilde{K} = \tilde{K} - \Phi\tilde{K}\Phi^T$$

is the sum of 5 rank–one matrices (see Fig. 4.1).

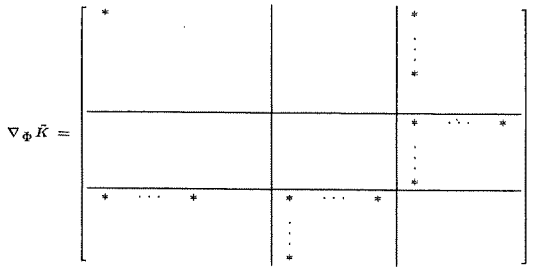


Figure 4.1. Nonzero pattern of $\nabla_\Phi \tilde{K}$.

Considering the Schur complement of $I_{(m+n)\times(m+n)}$ in $\tilde{K}$ we can obtain the following partial LDL^T decomposition of $\tilde{K}$ without any additional cost (of course the product XX^T is not explicitly computed),

$$\tilde{K} = \begin{bmatrix} I_{(m+n)\times(m+n)} & & \\ X & I & \\ 0_{n\times(m+n)} & & I \end{bmatrix} \begin{bmatrix} I_{(m+n)\times(m+n)} & & \\ & \hat{K} & \end{bmatrix} \begin{bmatrix} I_{(m+n)\times(m+n)} & X^T & 0_{(m+n)\times n} \\ & I & \\ & & I \end{bmatrix},$$

where the matrix

$$\hat{K} = \begin{bmatrix} -XX^T & \Lambda \\ \Lambda^T & 0_{n\times n} \end{bmatrix}$$

of order $m+n$ is the Schur complement of $I_{(m+n)\times(m+n)}$ in the matrix $\tilde{K}$. Then the problem is reduced to computing the LDL^T decomposition of $\hat{K}$.

Let

$$Z \triangleq Z_m \oplus Z_n.$$

Since the Schur complement inherits the displacement structure (see Ch. 2, § 2.4), then

$$\operatorname{rank}\left(\hat{K} - Z\hat{K}Z^T\right) \leq 5.$$

However, by inspection (see [64]), it can be seen that the displacement rank of $\hat{K}$ with respect to Z is 4. Denote by $v_1 = X(1,:)^T$, and $v = v_1/\|v_1\|_2$. Let $y = -Xv$ and $w = [\lambda(n), \lambda(n-1), \ldots, \lambda(1)]^T/\|v_1\|_2$. Then the generators of $\hat{K}$ are defined in the following way.

$$\begin{aligned}
g_1 &= [y^T, w^T]^T \\
g_2 &= [0, y(2:m)^T, w^T]^T \\
g_3 &= [0, \lambda(m+n-1), \lambda(m+n-2), \ldots, \lambda(n+1), .5, 0, \ldots, 0]^T \\
g_4 &= [0, \lambda(m+n-1), \lambda(m+n-2), \ldots, \lambda(n+1), -.5, 0, \ldots, 0]^T,
\end{aligned}$$

where g_2 and g_3 are positive, g_1 and g_4 are negative. We remark that the order of the columns in the generator matrix defined in Ch. 2 (the positive and the negative generators are in the first and in the last columns of the matrix, respectively) can easily be retrieved by permuting the columns in (4.3.10). Similarly the diagonal elements of the matrix J defined in (4.3.11) must be exchanged.
Define

$$G_{Z,0} \triangleq \begin{bmatrix} g_1 & g_2 & g_3 & g_4 \end{bmatrix} \tag{4.3.10}$$

and let

$$J \triangleq \begin{bmatrix} -1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -1 \end{bmatrix}. \tag{4.3.11}$$

Then

$$\hat{K} = G_{Z,0} J G_{Z,0}^T.$$

Since the order of the matrix $\hat{K}$ is $m+n$ the computational cost of the generalized Schur algorithm should be proportional to $(m+n)^2$. In the next section we will show that, exploiting the particular structure of the generators of $\hat{K}$ the computational cost of the generalized Schur algorithm can be reduced to $O(mn+n^2)$.

We observe that the matrix $\hat{K}$ is indefinite. However, analyzing the generators and the Schur complement of $-XX^T$ in $\hat{K}$ we are able to say a priori that the steps of the algorithm for $i = 1, \ldots, m$ are negative, the steps for $i = m+1, \ldots, m+n$, are positive.

Remark 4.3.1: *We observe that $g_1(1:m), g_2(1:m)$ are the generators for the symmetric negative definite Toeplitz matrix $-XX^T$. If we denote by*

$$\hat{G}_1 \triangleq \left[\begin{array}{cc} g_1(1:m) & g_2(1:m) \end{array}\right],$$

the generator matrix for $-XX^T$, by $\hat{G}_i$ and $\hat{f}_i$ the updated generator matrix and the ith row of $\hat{G}_i$, respectively, obtained at the ith iteration for the computation of the LDL^T factorization of $-XX^T$, we have that $\hat{f}_i \left[\begin{array}{cc} -1 & 0 \\ 0 & 1 \end{array}\right] \hat{f}_i^T < 0$.

Description of the algorithm

As introduced in Ch. 2 and Ch. 3, at each iteration i of GSA, we look for a J–orthogonal matrix Θ_i in order to eliminate all elements of f_i,the i–th row of G_i with exception of one element. This can be done by choosing J–orthogonal matrices Θ such that

$$\left[\begin{array}{cc} * & 0 \end{array}\right] = \left[\begin{array}{cc} f_i(j) & f_i(k) \end{array}\right] \Theta.$$

Θ can be either a Givens rotation if $\{j,k\} \in \{1,4\}$ or $\{j,k\} \in \{2,3\}$, or a hyperbolic rotation elsewhere. Proceeding in this way we can eliminate all the entries of f_i with exception of a single pivot element. Since the Givens and the hyperbolic rotations are used in updating and downdating a Cholesky decomposition [16], [11] respectively, we shall also refer to an application of a Givens rotation to a pair of generators as a *updating* and an application of an hyperbolic rotation to a pair of generators as a *downdating*. We choose the index of the pivot element equal to 1 if $f_i J f_i^T < 0$ and we set $D(i,i) = -1$ (negative step), equal to 2 in the other case and we set $D(i,i) = 1$ (positive step). The case $f_i J f_i^T = 0$ is ruled out by the strong regularity assumption on $\hat{K}$ [66]. We perform the downdating step by means of a *mixed* hyperbolic rotation[4] [16], [119].

To exploit the *structure* of the generators we divide the algorithm into 4 phases:

- 1st phase: iteration `for` $i = 1$,
- 2th phase: iteration `for` $i = 2, \ldots, m-n$,
- 3th phase: iteration `for` $i = m-n+1, \ldots, m$,
- 4th phase: iteration `for` $i = m+1, \ldots, m+n$.

1st phase: iteration for $i = 1$

$g_1^{(0)}$ is the only vector with the first entry different from zero. Then we set $L(:,1) = g_1^{(0)}, D(1,1) = -1, g_1^{(1)}(2:m+n) = g_1^{(0)}(1:m+n-1), g_1^{(1)}(1) = 0, g_1^{(1)}(m+1) = 0$.

[4] In the literature this kind of hyperbolic rotation is also called *hyperbolic rotation in factored form* or *stabilized hyperbolic rotation.*

2nd phase: iterations for $i = 2 : m - n$

Before describing this phase we observe that the vectors $g_3^{(i-1)}$ and $g_4^{(i-1)}$ differ only for the $(m+1)$th entry. This implies the updating of $g_1^{(i-1)}$ with $g_4^{(i-1)}$ and the downdating with $g_3^{(i-1)}$ modifies only the $(m+1)$th entry of $g_1^{(i-1)}$. Hence $g_1^{(i-1)}(1:m)$ and $g_2^{(i-1)}(1:m)$ continue to be the generator vectors at the beginning of the ith step for the LDL^T factorization of $-XX^T$. Thus each iteration of this phase is a negative step since $f_i J f_i^T < 0$. Now we describe how the generators are modified at each iteration of this phase. We have to update $g_1^{(i-1)}$ with $g_4^{(i-1)}$ and downdate with $g_3^{(i-1)}$. These vectors are

$$g_1^{(i-1)} = [\underbrace{0,\ldots,0}_{i-1}, \xi_i, \ldots, \xi_{n+i}, \underbrace{0,\ldots,0}_{m-n-i}, \xi_{m+1}, \ldots, \xi_{m+n}]^T \tag{4.3.12}$$

$$g_4^{(i-1)} = [\underbrace{0,\ldots,0}_{i-1}, \zeta_i, \ldots, \zeta_m, \zeta_{m+1}, \zeta_{m+2} \ldots, \zeta_{m+n}]^T$$

$$g_3^{(i-1)} = [\underbrace{0,\ldots,0}_{i-1}, \zeta_i, \ldots, \zeta_m, \mu_{m+1}, \zeta_{m+2} \ldots, \zeta_{m+n}]^T.$$

The Givens rotation used to update $g_1^{(i-1)}$ with $g_4^{(i-1)}$ is

$$Q_{i-1} \triangleq \begin{bmatrix} c_G^{(i-1)} & -s_G^{(i-1)} \\ s_G^{(i-1)} & c_G^{(i-1)} \end{bmatrix},$$

with

$$c_G^{(i-1)} = \frac{\xi_i}{\sqrt{\xi_i^2 + \zeta_i^2}} \text{ and } s_G^{(i-1)} = \frac{\zeta_i}{\sqrt{\xi_i^2 + \zeta_i^2}}.$$

The updated vectors $\tilde{g}_1^{(i-1)}$ and $\tilde{g}_4^{(i-1)}$ are

$$\tilde{g}_1^{(i-1)} = c_G^{(i-1)} g_1^{(i-1)} + s_G^{(i-1)} g_4^{(i-1)} \tag{4.3.13}$$

$$\tilde{g}_4^{(i-1)} = -s_G^{(i-1)} g_1^{(i-1)} + c_G^{(i-1)} g_4^{(i-1)}, \tag{4.3.14}$$

with $\tilde{g}_1^{(i-1)} = [\underbrace{0,\ldots,0}_{i-1}, \tilde{\xi}_i, \ldots, \tilde{\xi}_{m+n}]^T \quad \left(\tilde{\xi}_i = \sqrt{\xi_i^2 + \zeta_i^2}\right)$. Moreover,

$$\tilde{g}_4^{(i-1)}(n+i+1:m) = c_G^{(i-1)} g_4^{(i-1)}(n+i+1:m), \tag{4.3.15}$$

since $g_1^{(i-1)}(j) = 0, j = n+i+1, \ldots, m$. The next step in the ith iteration is downdating $\tilde{g}_1^{(i-1)}$ with $g_3^{(i-1)}$ by means of a mixed hyperbolic rotation

$$H_{i-1} \triangleq \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{1-\rho^2}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \rho \\ 0 & 1 \end{bmatrix},$$

where ρ is such that

$$\begin{bmatrix} \hat{\xi}_i & 0 \end{bmatrix} = \begin{bmatrix} \tilde{\xi}_i & \zeta_i \end{bmatrix} H_{i-1}^T.$$

Taking (4.3.13) into account, it is straightforward to see that

$$\rho = -s_G^{(i-1)} \text{ and } \sqrt{1-\rho^2} = c_G^{(i-1)}.$$

The downdated vectors $\hat{g}_1^{(i-1)}$ and $\tilde{g}_3^{(i-1)}$ are

$$\hat{g}_1^{(i-1)} = \frac{\tilde{g}_1^{(i-1)} - s_G^{(i-1)} g_3^{(i-1)}}{c_G^{(i-1)}} = g_1^{(i-1)} - s_G^{(i-1)} \frac{g_3^{(i-1)} - g_4^{(i-1)}}{c_G^{(i-1)}} \tag{4.3.16}$$

and

$$\tilde{g}_3^{(i-1)} = -s_G^{(i-1)} \hat{g}_1^{(i-1)} + c_G^{(i-1)} g_3^{(i-1)}. \tag{4.3.17}$$

$$\begin{bmatrix} u_1 & v_1 & v_1 \\ u_2 & v_2 & v_2 \\ \vdots & \vdots & \vdots \\ u_n & v_n & v_n \end{bmatrix} \longrightarrow \begin{bmatrix} u_1 & 0 & 0 \\ u_2 & c_G v_2 - s_G u_2 & c_G v_2 - s_G u_2 \\ \vdots & \vdots & \vdots \\ u_n & c_G v_n - s_G u_n & c_G v_n - s_G u_n \end{bmatrix}$$

Figure 4.2. Updating and downdating of the vector u with the same vector v. The result is that the vector u is still the same, and the second and the third vectors are $c_G v - s_G u$, wher c_G and s_G are the *Givens coefficients* used in the updating.

Hence,

$$\begin{aligned} \tilde{g}_3^{(i-1)} &= -s_G^{(i-1)} \hat{g}_1^{(i-1)} + c_G^{(i-1)} g_3^{(i-1)} \\ &= -s_G^{(i-1)} g_1^{(i-1)} + \frac{{c_G^{(i-1)}}^2 g_3^{(i-1)} - {s_G^{(i-1)}}^2 (g_4^{(i-1)} - g_3^{(i-1)})}{c_G^{(i-1)}} \\ &= -s_G^{(i-1)} g_1^{(i-1)} + \frac{g_3^{(i-1)} - (1 - {c_G^{(i-1)}}^2) g_4^{(i-1)}}{c_G^{(i-1)}}. \end{aligned} \tag{4.3.18}$$

Hence, from (4.3.17) and (4.3.14), $\tilde{g}_3^{(1)}$ and $\tilde{g}_4^{(1)}$ continue to be equal, except for the $(m+1)$th entry. Furthermore, from (4.3.16), we observe that $g_1^{(i-1)}$ and $\hat{g}_1^{(i-1)}$ differ in their $(m+1)$th entry. Since $g_1^{(i-1)}(m+1) = 0$, then

$$\hat{g}_1^{(i-1)}(m+1) = -s_G^{(i-1)} (g_3^{(i-1)}(m+1) - g_4^{(i-1)}(m+1)) / c_G^{(i-1)}.$$

We observe that it is not necessary to compute the whole vector in (4.3.15) since, at the next iteration, the corresponding entries of $g_1^{(i)}(n+i+2:m+1)$ are equal to 0. We need only to store the partial product

$$c_G^{(i-2)} \cdots c_G^{(2)} c_G^{(1)} \tag{4.3.19}$$

into a temporary variable, and multiply $g_4^{(i)}(n+i+1)$ with this variable at the beginning of the ith iteration. The i–th iteration of this phase is illustrated in Fig. 4.3.

To finish the iteration $\hat{g}_1^{(i-1)}$ has to be downdated with $g_2^{(i-1)}$. This computation does not destroy the structure of the vectors since

$$\begin{aligned}
\hat{g}_1^{(i-1)} &= [\underbrace{0,\ldots,0}_{i-1},\underbrace{*,\cdots,*,*}_{n+1},\underbrace{0,\ldots,0}_{m-n-i},\underbrace{*,\cdots,*}_{n}]^T \\
g_2^{(i-1)} &= [\underbrace{0,\ldots,0}_{i-1},\underbrace{*,\cdots,*}_{n},\underbrace{0,\ldots,0}_{m-n-i+1},\underbrace{*,\cdots,*}_{n}]^T.
\end{aligned}$$

Let $\tilde{H}_{i-1}$ be the stabilized hyperbolic rotation such that

$$\left[\check{g}_1^{(i-1)}(i:m+n) \quad \check{g}_2^{(i-1)}(i:m+n)\right] = \left[\hat{g}_1^{(i-1)}(i:m+n) \quad g_2^{(i-1)}(i:m+n)\right] \tilde{H}_{i-1}^T$$

with $\check{g}_2^{(i-1)}(i) = 0$. Then $\check{g}_1^{(i-1)}$ becomes the ith column of L, $D(i,i) = -1$, and, for the next iteration, the updated vectors are

$$\begin{aligned}
g_1^{(i)} &= [0, \hat{g}_1^{(i-1)}(i+1:m+n)],\ g_1^{(i)}(m+1) = 0 \\
g_2^{(i)} &= \check{g}_2^{(i-1)} \\
g_4^{(i)} &= \tilde{g}_4^{(i-1)} \\
g_3^{(i)} &= [\tilde{g}_4^{(i-1)}(1:m); \gamma; \tilde{g}_4^{(i-1)}(m+2:m+n)],
\end{aligned}$$

where $\gamma = c_G^{(i-1)} g_3^{(i-1)}(m+1) - s_G^{(i-1)} \hat{g}_3^{(i-1)}(m+1)$. The number of flops of this phase is $18mn - 18n^2$.

3rd phase: iterations for $i = m-n+1:m$

The iterations of this phase are very similar to those of the previous one. However we do not need to store the product of the Givens coefficients $c_G^{(i-1)}$ into a temporary variable, since $g_1^{(i-1)}(k) \neq 0, k = i,\ldots,m+n$. We recall that $g_3^{(i-1)}$ and $g_4^{(i-1)}$ continue to be equal with exception of the $(m+1)$th entry. Thus $g_1^{(i-1)}(i:m)$ and $g_2^{(i-1)}(i:m)$ are the generator vectors at the ith iteration of the LDL^T factorization of $-XX^T$. Hence each iteration of this phase continues to be a negative step ($D(i,i) = -1$). The number of flops of this phase is $13.5n^2$.

4th phase: iterations for $i = m+1 : m+n$

In this phase the vectors $g_3^{(i-1)}(m+1 : m+n)$ and $g_4^{(i-1)}(m+1 : m+n)$ are different. Now we observe that the vectors $g_i^{(m)}, i = 1, \ldots, 4$ are the generators for the Schur complement of $-XX^T$ in the matrix $\hat{K}$ [67], that is, the generators for $\Lambda^T(XX^T)^{-1}\Lambda$, a symmetric and positive definite matrix. Then each iteration of this phase is a positive step, meaning that $D(i,i) = 1$ and the ith column of L is $\check{g}_2^{(i-1)}$. The number of flops of this phase is $9n^2$.

The corresponding `matlab`–like program `FastLD` can be found in App. B.

4.3.2 Stability of LDL^T factorization

The stability of the proposed generalized Schur algorithm is studied in Ch. 3. The stability properties of the generalized Schur algorithm for the considered problem depend on the implementation of the hyperbolic rotations.

In Ch. 3 it is proved that the following results hold for the LDL^T factorization of $\hat{K}$, provided the hyperbolic rotations are implemented in a stable way [16], [23].

Theorem 4.3.1: *Let G be the generator matrix in (4.3.10) of $\hat{K}$. Let L and D be the matrices of the LDL^T factorization of $\hat{K}$ computed by means of the generalized Schur algorithm applying a sequence of Givens rotations and two mixed hyperbolic rotations per iteration. Then*

$$\|\hat{K} - LDL^T\|_F \leq 74(m+n-1)(m+n)\varepsilon \left(2\sqrt{m+n}\|\hat{K}\|_F + \|G\|_F^2\right).$$

Hence the proposed algorithm is backward stable.

4.3.3 Solution of the linear systems

In this subsection we evaluate the computational cost of the solution of the linear system of equations (4.3.9).

Having computed the following factorization of $\tilde{K}$ in $O(mn+n^2)$ flops,

$$\tilde{K} = \begin{bmatrix} I & & \\ X & I & \\ 0_{n\times(m+n)} & & I \end{bmatrix} \begin{bmatrix} I & \\ & R^T \end{bmatrix} \begin{bmatrix} I & \\ & D \end{bmatrix} \begin{bmatrix} I & \\ & R \end{bmatrix} \begin{bmatrix} I & X^T & 0_{(m+n)\times n} \\ & I & \\ & & I \end{bmatrix}$$
$$= L_1 L_2 D_1 L_2^T L_1^T,$$

we need now to solve 5 linear systems, with coefficient matrices $L_1, L_2, D_1, L_2^T, L_1^T$, respectively. The solution of the systems with coefficient matrix L_1 and L_1^T can be computed in $O(mn)$ flops. The solution of the linear system with coefficient matrix D_1 is obtained changing the sign of the entries $m+n+1, \ldots, 2m+n$, of b_1. Furthermore, the solution of the linear systems with coefficient matrix L_2 and L_2^T

can be computed in $0(mn + n^2)$ flops since

$$R = \begin{bmatrix} * & * & * & * & & & * & * & * \\ & * & * & * & * & & * & * & * \\ & & * & * & * & * & * & * & * \\ & & & * & * & * & * & * & * \\ & & & & * & * & * & * & * \\ & & & & & * & * & * & * \\ & & & & & & * & * & * \\ & & & & & & & * & * \\ & & & & & & & & * \end{bmatrix},$$

where the first row is $R(1,:) = [\underbrace{*,\ldots,*}_{n+1},\underbrace{0,\ldots,0}_{m-n-1},\underbrace{*,\ldots,*}_{n}]^T$.

Hence the solution of the linear system (4.3.9) has the same computational complexity as the LDL^T factorization of $\hat{K}$.

4.4 Application

The STLS problem involving Toeplitz (or Hankel) matrices constitutes the kernel problem of many signal processing applications. In this section the fast implementation of the STLS2 algorithm described in the previous section is applied to the speech compression application introduced in Ch. 1, § 1.3.2 (see also [79], [77]).
Subsection 4.4.1 gives a short description of the latter application. It is shown that the low-frequency part of the compression scheme is a Toeplitz STLS problem.
Subsection 4.4.2 first briefly describes the two straightforward implementations of algorithm STLS1 and STLS2 that will be used in comparison with the fast implementation of algorithm STLS2. The two alternative implementations do solve the Toeplitz STLS problem (thus the structure at the first level is taken into account) but they do not exploit the particular structure of matrices that appear in the algorithms described (thus the structure at the second level is not exploited). Next, the computational efficiency of the three implementations is compared by means of the speech compression application.
Furthermore, the dependence of the number of flops (per iteration) on the size of the problem (i.e. the frame length and the estimated number of components) is verified for the different implementations.

4.4.1 Description

For the sake of clarity, let us shortly repeat the description of the subband based speech compression scheme already presented in Ch. 1, § 1.3.2. The incoming speech signal is processed on a frame by frame basis. In a first step a frame of the incoming signal is fed to a lowpass filter. This yields the low-frequency (LF) part. By subtracting the latter from the incoming signal, we obtain the high-frequency (HF), non-smoothed part. The motivation behind splitting the speech signal into

two bands is that the LF part of a speech signal can efficiently be modeled by means of a linear model (Auto Regressive (AR) model). Instead of determining a classical AR model, leading to a vector of model parameters and a residual vector [85], we apply a minimal correction to the measured data (i.e. the LF part of the frame) such that the corrected measurements exactly satisfy an AR model. As shown in [77], [79], the latter procedure corresponds to solving a Toeplitz STLS problem (4.2.3) (with $W = I_{m+n}$) in which the observed data vector s in (4.1.2) corresponds to the LF part of the incoming frame, the corrections we apply correspond to the vector Δs in (4.2.4) and the model for that particular frame is represented by the vector x. The compression comes about as follows. Due to the Toeplitz structure of $[A + \Delta A \; b + \Delta b]$ and the equality constraint in (4.2.3), it should be clear that we only need $s(1:n) + \Delta s(1:n)$ and $x \in \mathbb{R}^{n\times 1}$ in order to reconstruct the entire frame $s + \Delta s \in \mathbb{R}^{(m+n)\times 1}$. Thus, loosely speaking, we obtain a compression ratio of $2n/(m+n)$. For further implementation details we refer to [77], [79].

4.4.2 Numerical tests

In this subsection we compare the efficiency of three implementations. The first one is the fast implementation of the STLS2 algorithm as described in section 4.3. It will be referred to as STLS2f. We also consider a straightforward implementation of algorithm STLS2, referred to as STLS2s, in which Step 2.1 of algorithm STLS2 is solved by Gaussian elimination with partial pivoting [53]. No use is made of the particular structure of the matrix involved in this system of equations. The third implementation, referred to as STLS1s, is a straightforward implementation of the algorithm STLS1, without any optimization with respect to the structure of the involved matrices. This means that we use a standard GRQ algorithm[5] to solve the equality constrained LS problem in step 2.1 of algorithm STLS1. As shown in the previous section, the computational complexity of STLS2f is $O(mn + n^2)$. A more rigid flop count based on the program that can be found in appendix A, yields a theoretical flop count of $40mn + 71m - 36n - 13n^2 + 90$. The implementations STLS1s and STLS2s obviously[6] have a computational complexity of respectively $O(m^3)$ and $O((m+n)^3)$ per iteration.

The speech signal (see left hand side of Fig. 4.4) we compress is 14616 samples long. It is processed on a frame by frame basis, each frame being 504 samples long. As mentioned before each frame is split in a LF and a HF part. It is the LF part that is compressed using the STLS problem formulation. The LF part of the frame can be modelled sufficiently well using a 4th order model, thus yielding the following dimensions of the STLS problem (4.2.3): $m = 500$, $n = 4$. The right hand side of

[5] As mentioned before, the first step (RQ) of the GRQ algorithm applied to the Toeplitz STLS problem can be made more efficient. However, all structure is lost in the second step (QR), which will dominate the computational cost.

[6] For STLS1s the computational cost is mainly due to the Householder reflections used in the RQ and QR factorizations, whereas for STLS2s the computational cost is that of the Gaussian elimination scheme with partial pivoting.

Fig. 4.4 shows the LF part of the original 6th frame (i.e. the signal s, full line) and the reconstructed LF part of the frame obtained after compression (i.e. the signal $s + \Delta s$, dashed line). The latter frame can be reconstructed using only the first 4 samples of $s + \Delta s$ and the parameter vector $x \in \mathbb{R}^{4\times 1}$.
Per frame, the three implementations STLS1s, STLS2s and STLS2f require the same (but varying) number of iterations. The important number is thus the number of flops per iteration. For this particular problem these numbers are displayed in table 4.1 (see first line of the table). We clearly see the drastically increased computational performance obtained with STLS2f. Also note that the cells in the column of the fast STLS2f implementation are split in two parts: the upper part contains the flop count as obtained by the Matlab function *flops* whereas the lower part (bold number) contains the theoretically obtained flop count (i.e. flop count based on the program described in the appendix B). We see that there is a strong correspondence between both numbers.
To investigate the dependence of the computational cost of the different implementations on the size of the problem, we change the parameters of the compression scheme. The three implementations are applied using the following framelengths and estimated orders:

- framelength=254, estimated order=4 (this corresponds to $m = 250$, $n = 4$)
- framelength=508, estimated order=8 (this corresponds to $m = 500$, $n = 8$).

The results are shown on the second and third line of table 4.1 and 4.2. Going from the second to the first line of the tables, m is doubled. As could be expected from the theoretical flop count, the number of flops per iteration for STLS1s and STLS2s is approximately multiplied by 8, whereas the number of flops per iteration for STLS2f is only doubled. Going from the first to the third line of the table, n is doubled and m remains constant. As could be expected from the theoretical flop count estimate, doubling n does not really change the number of flops per iteration for implementations STLS1s and STLS2s. For the fast implementation we see that the number of flops is not really doubled, because of the large linear term $(71m)$ in the theoretical flop count. Theoretically we would expect the flop count to be multiplied by 1.67, which is very close to the factor 1.74 found when using the results obtained by the matlab command *flops*.

4.5 Conclusion

In this chapter a fast implementation of the Toeplitz (Hankel) STLS problem (i.e. an STLS problem in which the Toeplitz (Hankel) structure needs to be preserved) is described. This type of STLS problem occurs frequently in signal processing applications.
An algorithm is developed by applying the method of Lagrange multipliers to this specific STLS problem. The kernel problem that needs to be solved in each iteration of this iterative algorithm is the solution of a system of equations. By exploiting

Table 4.1. This table shows the increased performance of the implementation STLS2f compared to the straightforward implementations STLS1s and STLS2s, for different problem sizes. The bold numbers represent the theoretical flop count whereas the other numbers represent the flop count obtained with the Matlab function *flops*.

frame length	estimated order	$m \times n$	STLS1s	STLS2s	STLS2f
504	4	500×4	9.624×10^8	6.916×10^8	117854 **116162**
254	4	250×4	1.262×10^8	8.968×10^7	59104 **57488**
508	8	500×8	1.002×10^9	7.080×10^8	2.056×10^5 **194470**

Table 4.2. This table shows the increased performance of the implementation STLS2f compared to the straightforward implementations STLS1s and STLS2s, for different problem sizes. $flops_{STLS1s}$, $flops_{STLS2s}$ and $flops_{STLS2f}$ represents the number of flops per iteration for respectively the implementations STLS1s, STLS2s and STLS2f.

frame length	estimated order	$m \times n$	$\frac{flops_{STLS1s}}{flops_{STLS2f}}$	$\frac{flops_{STLS2s}}{flops_{STLS2f}}$
504	4	500×4	8166	5868
254	4	250×4	2135	1517
508	8	500×8	4874	3444

the low displacement rank of the involved matrices a fast implementation can be developed. By taking advantage of the sparsity of the corresponding generators we are able to even further improve the computational efficiency. The computational complexity for each iteration is $O(mn + n^2)$ whereas straightforward implementations have a computational complexity of $O(m^3)$ or $O((m+n)^3)$. These numbers are confirmed by applying the different implementations in a recently proposed speech compression scheme that contains a Toeplitz STLS problem as kernel problem.

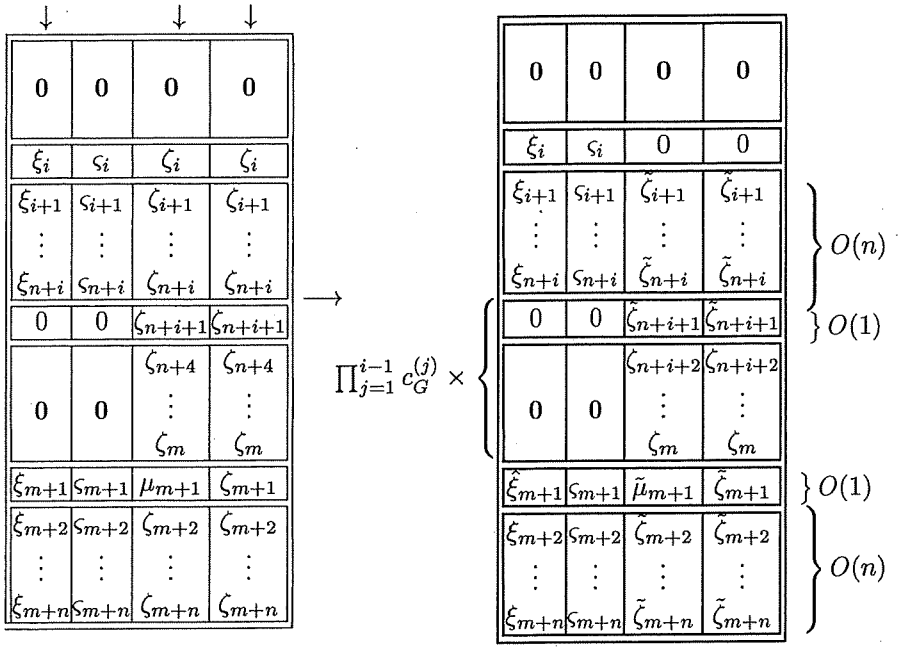


Figure 4.3. To better understand the effects of the right–multiplication of Q_{i-1} and H_{i-1} to the generator matrix G_{i-1} at the ith iteration of 2nd phase, the generator matrix G_{i-1} is divided into seven *"windows"*, each of those with $i-1, 1, n, 1, m-n-i-1, 1, n-1$ rows, respectively. We remark that the second, the third and the fourth windows move downward one position at each iteration. Since the element in the third and the fourth columns of G_{i-1} are equal with the exception of the entries in the fifth window, it turns out that Q_{i-1} and H_{i-1} modify only the first column of the fifth window, in a way similar to that one depicted in Fig. 4.2. In addition the third and the fourth columns, continue to be equal, with the exception in the fifth window. The rows of $\hat{G}_{i-1} \triangleq G_{i-1}Q_{i-1}H_{i-1}$ in the fourth and fifth windows are equal to the corresponding rows in G_0 multiplied by $\prod_{j=1}^{i-1} c_G^{(j)}$. Since the downdating of the first and second columns in $\hat{G}_{i-1}$ does not fill the first, fourth and fifth windows and since at the end of the i–th iteration the first column is moved downward by Φ, it is only necessary to compute the entries in the fourth window, with $O(1)$ flops, without computing the entries in the fifth window.

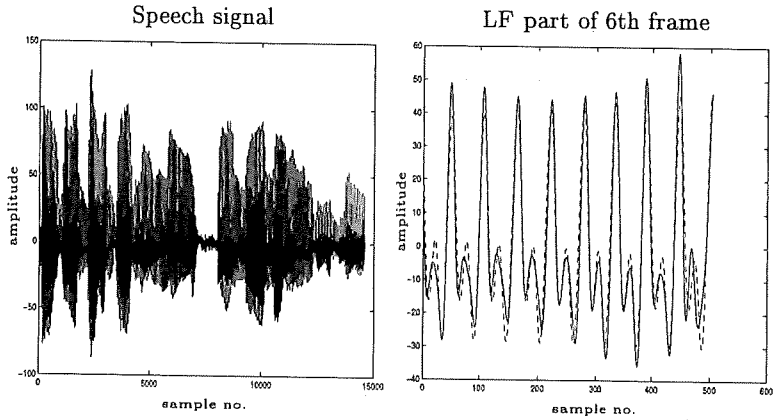


Figure 4.4. The left hand side of this figure shows the original speech signal. The right hand side shows the LF part of the 6th frame of the original signal (i.e. s, the full line) together with the reconstructed (after compression) LF part of the 6th frame (dashed line).

Chapter 5

FAST ALGORITHM FOR THE BASIC DECONVOLUTION PROBLEM

5.1 Introduction

The basic deconvolution problem has been described in Ch. 1, § 1.3.3. In this chapter we develop a fast algorithm for the basic deconvolution problem. The latter problem is depicted in Ch. 1, Fig. 1.1, where $u_k + \Delta u_k$ represents the measured input (whereas u_k is the true but unmeasured input) and $y_k + \Delta y_k$ represents the measured output (whereas y_k is the true but unmeasured output) at time k; Δu_k and Δy_k are i.i.d. white Gaussian measurement noise added, respectively, to the input and to the output. The system, represented by its transfer function $X(z)$, is a linear time-invariant system with impulse response $x \in \mathbb{R}^{n \times 1}$. The basic deconvolution problem can now be formulated as follows:

> Given the noisy (we assume Gaussian i.i.d. additive noise of equal variance) measurements $u_k + \Delta u_k$, $k = 1, \ldots, m+n-1$ and $y_k + \Delta y_k$, $k = 1, \ldots, m$,
>
> of the linear system, find a maximum likelihood (ML) estimate for the system impulse response x_i, $i = 1, \ldots, n$.

In the next section we show that the ML estimator for the basic deconvolution problem is a structured total least squares (STLS) problem. The basic deconvolution problem, formulated as $Ax \approx y$ and described in § 5.2, is solved using the 2-norm STLN algorithm described in [107], with A Toeplitz-structured and y unstructured. This iterative STLN algorithm solves as kernel problem in each iteration step a least squares problem involving a higher-dimensional structured and sparse data matrix. However, the STLN algorithm described in [107], requiring $O\left((m+n)^3\right)$ operations, does not exploit these matrix features.

In this chapter we present the fast implementation presented in [106], requiring $O(mn^2 + m^2)$ operations, which partially exploits the matrix structure and a faster algorithm of $O(mn + n^2)$ computational complexity, based on the generalized Schur algorithm (see, i.e., Ch. 2, [67], [66]). The improved efficiency is obtained by fully exploiting the low displacement rank of the involved matrices (displacement rank 5) and the sparsity of the corresponding generators throughout all computations. In addition, we prove that this STLS estimator provides statistically better estimates of the impulse response given input and output data affected by i.i.d. zero mean noise with equal variance, compared to the ordinary TLS estimator.

The chapter is organized as follows. § 5.2 describes the basic deconvolution problem and also outlines the 2-norm STLN algorithm with unstructured right-hand side, as described in [107], for solving this problem. § 5.3 and § 5.4 describe two fast algorithms for solving the kernel problem of the STLN approach applied to the basic deconvolution problem: a least squares (LS) problem involving structured matrices. §5.5 describes some examples of typical deconvolution problems in order to demonstrate the dependency of the number of flops on the problem size and to show the increased computational efficiency w.r.t. existing implementations. To illustrate

the statistical properties of the STLS estimator, a simulation experiment based on a medical application in renography is described. By means of a Monte-Carlo simulation, using several noise levels, we show the improved statistical accuracy of the deconvolution results obtained with the STLS estimator as compared to other estimators such as the TLS estimator that do not impose a structure on the correction matrix $[\Delta A \;\; \Delta y]$.

5.2 The basic deconvolution problem

Starting from the problem formulation of the basic deconvolution problem in section 5.1, it is straightforward to show that a ML estimate can be found as the solution of the following problem (for a proof, see [3]):

$$\begin{aligned} &\min_{E,x} \alpha^T\alpha + \beta^T\beta \\ &\text{s.t. } (A+E)x = y + \beta, \end{aligned} \tag{5.2.1}$$

with

$$A = \begin{bmatrix} u_n & u_{n-1} & \cdots & u_1 \\ u_{n+1} & u_n & \cdots & u_2 \\ \vdots & \ddots & & \vdots \\ u_{m+n-1} & u_{m+n-2} & \cdots & u_m \end{bmatrix} \in \mathbb{R}^{m\times n},$$

$$E = \begin{bmatrix} \alpha_n & \alpha_{n-1} & \cdots & \alpha_1 \\ \alpha_{n+1} & \alpha_n & \cdots & \alpha_2 \\ \vdots & \ddots & & \vdots \\ \alpha_{m+n-1} & \alpha_{m+n-2} & \cdots & \alpha_m \end{bmatrix} \in \mathbb{R}^{m\times n},$$

$$\beta = (A+E)x - y \in \mathbb{R}^{m\times 1},$$

with E the correction applied to A, β the correction applied to y, $y \in \mathbb{R}^{m\times 1}$ the output, and $x \in \mathbb{R}^{n\times 1}$ the impulse response. Problem (5.2.1) is a STLS problem, since corrections can be applied to the left-hand side matrix A of the constraints in (5.2.1) (implying that it is a *total* LS type problem) and in addition the corresponding correction matrix E is structured (implying that we have to deal with a *structured* TLS problem). As already mentioned in the introduction, we will apply the STLN approach, implying that we solve (5.2.1) as an optimization problem. Using the zeroth and first order terms of the Taylor series expansion of $\beta = (A + E(\alpha))x - y$ (where we use the notation $E(\alpha)$ to denote the dependence of E on α) around $[\alpha^T \;\; x^T]^T$, we obtain the Gauss–Newton method for solving (5.2.1) (for a proof, see [107]). The outline of the basic deconvolution algorithm, which is equivalent to the 2-norm STLN algorithm with unstructured right-hand side [107] for A Toeplitz, is then as follows:

Algorithm 5.2.1 (Basic Deconvolution Algorithm)
Input: *extended data matrix* $[A \;\; y] \in \mathbb{R}^{m\times(n+1)}$ *(*$m > n$*) of full rank* $n+1$.

Output: *correction vector* α *and parameter vector* x *s.t.* $\alpha^T\alpha + \beta^T\beta$ *is as small as possible and* $\beta = (A + E(\alpha))x - y$.

Step 1: $\alpha \leftarrow 0$

$x \leftarrow A\backslash y$

Step 2: `while` *stop criterion not satisfied*

Step 2.1: $\min_{\Delta x, \Delta\alpha} \left\| M \begin{bmatrix} \Delta\alpha \\ \Delta x \end{bmatrix} + \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \right\|_2$

with $M = \begin{bmatrix} X & A+E \\ I & 0 \end{bmatrix} \in \mathbb{R}^{(2m+n-1)\times(m+2n-1)}$

Step 2.2: $x \leftarrow x + \Delta x$

$\alpha \leftarrow \alpha + \Delta\alpha$

`end`

$\Diamond$

with $X \in \mathbb{R}^{m\times(m+n-1)}$ defined such that $X\alpha = Ex$. Taking E as defined previously in the deconvolution problem (5.2.1), X becomes

$$X = \begin{bmatrix} x_n & x_{n-1} & \cdots & x_1 & 0 & \cdots & \cdots & 0 \\ 0 & x_n & x_{n-1} & \cdots & x_1 & 0 & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \ddots & 0 \\ 0 & \cdots & 0 & x_n & x_{n-1} & \cdots & \cdots & x_1 \end{bmatrix}.$$

Note that $A\backslash y$ in Step 1 is a shorthand notation for the LS solution of the overdetermined system of equations

$$Ax \approx y. \tag{5.2.2}$$

As described in [76], more advanced initialization steps are possible [79]. They yield better starting values in the sense that convergence takes place in fewer iterations and to a better local minimum. However the price to be paid is an increase in computational complexity of the initialization. Due to the nature of the problem we consider, the simple least squares estimate will turn out to be sufficient in the considered application. The method of seminormal equations [11], with one step of iterative refinement [53], [118], [98], can be used to find the least squares solution. Due to the Toeplitz structure of A, the R factor of the QR decomposition of A can

be computed in $O(mn + n^2)$ flops [28], [15]. Hence the least squares solution of (5.2.2) can be computed in $O(mn + n^2)$ flops.

We use the following stop criterion in our implementation of the algorithm:

$$\|[\Delta\alpha^T \ \Delta x^T]\|_2 < 10^{-6}.$$

In the next two sections we describe two algorithms for solving the LS problem in Step 2.1 of algorithm 5.2.1. They basically consist of a fast triangularization of the matrix M, followed by the solution of the normal equations

$$M^T M[\Delta\alpha^T \ \Delta x^T]^T = R^T R[\Delta\alpha^T \ \Delta x^T]^T = -M^T[\beta^T \ \alpha^T]^T,$$

with $M = Q[R^T \ 0]^T$ the QR factorization of M. The triangularization of the matrix M can be obtained by means of the QR decomposition of M with a computational complexity of $O((m+n)^3)$. The algorithm considered in [107] and described in § 5.3 requires $O(mn^2 + m^2)$ flops. In § 5.4 we propose an algorithm for computing the matrix R, based on GSA. Since $M \in \mathbb{R}^{(2m+n-1)\times(m+2n-1)}$ the computational complexity of GSA should be $O((m+n)^2)$. Preserving the particular structure of the generator matrix at each iteration, we will be able to consider a particular implementation of GSA with $O(mn + n^2)$ computational complexity.

5.3 Park–Rosen–Glick (RPG) algorithm

The standard QR decomposition algorithm requires $O((m+n)^3)$ flops to compute the R factor of the QR decomposition of M. The algorithm we are going to describe, and proposed in [106], has $O(mn^2 + m^2)$ computational complexity.
Partition

$$X = \begin{bmatrix} X_1 & X_2 \end{bmatrix} \quad \text{and} \quad I = \begin{bmatrix} I_m & \\ & I_{n-1} \end{bmatrix},$$

the matrix M can be written

$$M = \begin{bmatrix} X_1 & X_2 & A+E \\ I_m & & \\ & I_{n-1} & \end{bmatrix}.$$

The algorithm can be divided into two steps:

- step 1: reduce the submatrix $M(1:m,:)$ to upper triangular,

$$W^{(0)} = \begin{bmatrix} X_1 & X_2 & A+E \\ I_m & & \end{bmatrix} \Rightarrow W^{(m)} = \begin{bmatrix} X_1^{(m)} & X_2^{(m)} & A^{(m)} + E^{(m)} \\ & W_{2,2}^{(m)} & W_{2,3}^{(m)} \end{bmatrix},$$

with $X_1^{(m)}$ and $W_{2,2}^{(m)}$ upper triangular, by means of orthogonal transformations exploiting the Toeplitz structure of the involved matrices;

- step 2: reduce the submatrix

$$\begin{bmatrix} W_{2,2}^{(m)} & W_{2,3}^{(m)} \\ I_{n-1} & \end{bmatrix}$$

to upper triangular form by means of orthogonal transformations.

Step 1 is accomplished in m iterations. At the first iteration, the elements in the main diagonal of I_m in $W^{(0)}$ are annihilated by means of a Givens rotation. Due to the Toeplitz structure of X_1 and I_m, a single Givens rotation $G^{(1)}$ is needed to annihilate the whole diagonal of I_m (see Fig. 5.1). This is achieved in $O(n+m)$ flops.

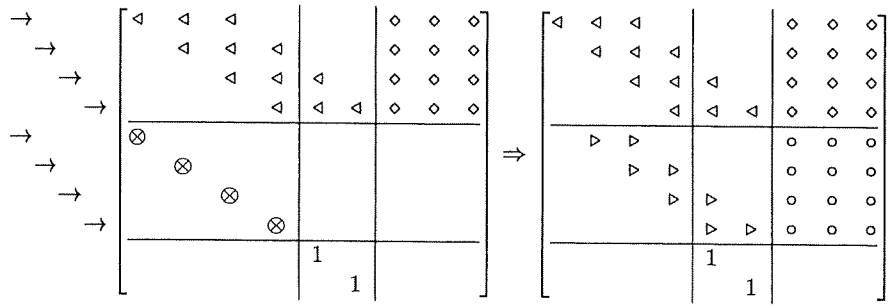


Figure 5.1. First iteration of step 1 of RPG algorithm. The symbols $\triangleright, \diamond, \triangleleft$, and $\circ$ are used to represent elements belonging to submatrices having a Toeplitz structure. The symbol $\otimes$ indicate the elements to be annihilated in the matrix by means of Givens rotations. The arrows on the left of each matrix indicate the pairs of rows on which the same Givens rotation is applied.

After the diagonal elements of I_m are annihilated, $W^{(0)}$ is transformed in

$$W^{(1)} \triangleq G^{(1)}W^{(1)} = \begin{bmatrix} X_1^{(1)} & X_2^{(1)} & A^{(1)}+E^{(1)} \\ W_{2,1}^{(1)} & W_{2,2}^{(1)} & W_{2,3}^{(1)} \end{bmatrix},$$

where $\begin{bmatrix} X_1^{(1)} & X_2^{(1)} \end{bmatrix}$, $A^{(1)}+E^{(1)}$, and $W_{2,3}^{(1)}$ are Toeplitz matrices, $X_1^{(1)}$ is upper triangular, and $\begin{bmatrix} W_{2,1}^{(1)} & W_{2,2}^{(1)} \end{bmatrix}$ is strictly upper triangular. At the second iteration, a Givens rotation is applied to annihilate the first superdiagonal of $W_{2,1}^{(1)}$. Since the first and last row are not involved, this computation is accomplished in $O(n+m-1)$ flops. Proceeding in this way, the i–th iteration will require $O(n+m-i+1)$ flops, $i=1,\ldots,m$ (see Fig. 5.2). Hence the computational complexity of step 1 is $O(m(m+n))$. Furthermore we observe that the submatrices

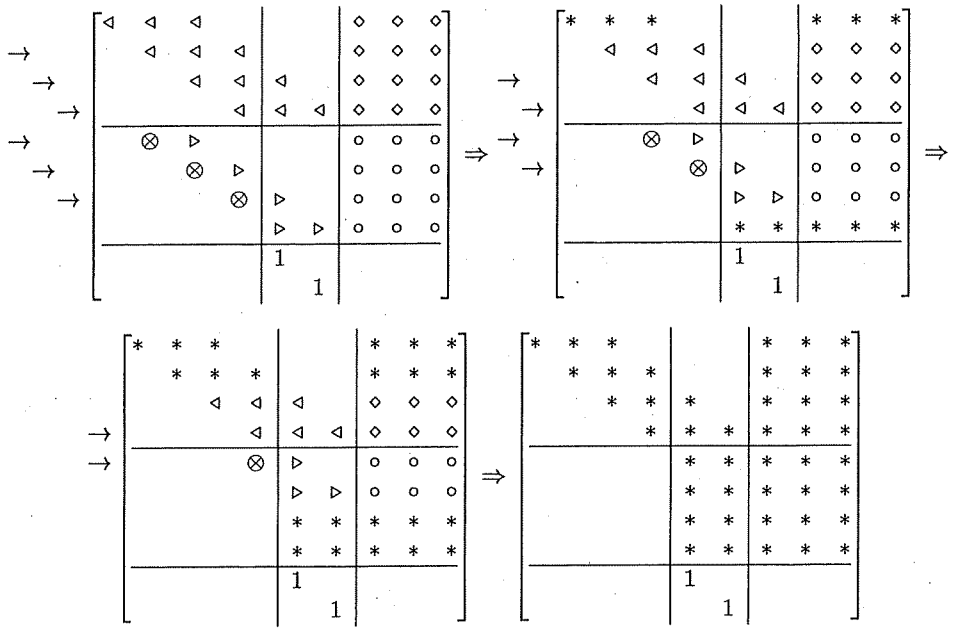


Figure 5.2. iterations $2, \ldots, m$ of step 1 of RPG algorithm. The symbols $\triangleright, \diamond, \triangleleft$, and $\circ$ are used to represent elements belonging to submatrices having a Toeplitz structure. The symbol $\otimes$ indicate the elements to be annihilated in the matrix by means of Givens rotations. The symbol $*$ denotes those elements in the matrix that are not more related to the others by any relation. The arrows on the left of each matrix indicate the pairs of rows on which the same Givens rotation is applied.

in $W^{(m)}$ don't have any structure. The computational complexity of step 2 by a standard QR decomposition is $O(mn^2)$. Therefore the computational complexity of the whole algorithm is $O(m^2 + mn^2)$.

5.4 Fast generalized Schur algorithm

In this section the triangular factor R of the QR decomposition of M is computed by means of GSA, exploiting its low displacement structure. R^T is, up to a multiplication by a sign matrix, also the Cholesky factor of $M^T M$. Moreover $M^T M$ is

the Schur complement of $-I_{2m+n-1}$ in the matrix

$$M_b = \begin{bmatrix} -I_{2m+n-1} & M \\ M^T & \end{bmatrix}.$$

Let

$$\Phi = Z_m \oplus Z_{m+n-1} \oplus Z_{m+n-1} \oplus Z_n,$$

where Z_* is the shift matrix of order $*$. Due to the structure of M it is easy to verify that,

$$\nabla_\Phi M_b = M_b - \Phi M_b \Phi^T$$

can be written as the sum of 6 rank–one matrices (see Fig. 5.3).

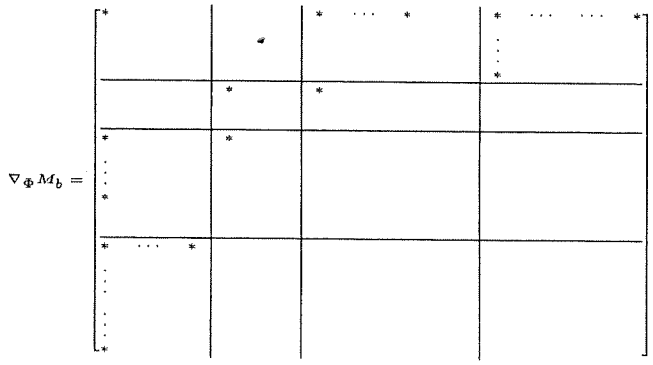


Figure 5.3. Nonzero pattern of $\nabla_\Phi M_b$.

Since the Schur complement inherits the displacement structure (see Ch. 2, § 2.4), then

$$\operatorname{rank}\left(M^T M - Z_{m+n-1} \oplus Z_n M^T M Z_{m+n-1}^T \oplus Z_n^T\right) \leq 6.$$

However, by inspection (see [64]), it can be seen that the displacement rank of $M^T M$ with respect to the block-shift matrix

$$Z = Z_{m+n-1} \oplus Z_n \tag{5.4.3}$$

is 5, and the following lemma holds.

Lemma 5.4.1: *Let* $w = M(2 : 2m+n-1, m+n)/\|M(2 : 2m+n-1, m+n)\|_2$ *and* $t = M(2 : 2m+n-1, 2 : m+2n-1)^T w$. *The following vectors form a generator of*

M^TM:

$$\begin{aligned} g_1 &= M(1,:)^T, \\ g_2 &= e_1, \\ g_3 &= [0, t^T]^T, \\ g_4 &= [0, t(1:m+n-2)^T, 0, t(m+n:m+2n-2)^T]^T, \\ g_5 &= [0, M(m, 1:m+n-2)^T, 0, M(m, m+n:m+2n-2)^T]^T, \end{aligned}$$

where $e_1 = [1, \underbrace{0, \ldots, 0}_{m+2n-2}]^T$. *Define*

$$G \triangleq [\, g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_5 \,].$$

Then

$$M^TM = GJG^T. \tag{5.4.4}$$

where

$$J = I_3 \oplus -I_2. \tag{5.4.5}$$

Proof. Construct M^TM and ZM^TMZ^T, where Z is the block–shift matrix (5.4.3). then straightforward manipulations show that $M^TM - ZM^TMZ^T$ can be expressed as a sum of five rank–one matrices. ◊

Following the technique described in Ch. 3, we can easily consider a GSA implementation for the computation of R with computational complexity of $O(m^2 + mn + n^2)$.

In the following section we consider a fast version of the GSA that requires only $18mn + 34.5n^2$ flops, taking into account the *"sparsity"* of the vectors $g_i, i = 1, \ldots, 5$.

5.4.1 Description of the fast generalized Schur algorithm

Let $G_{0,Z} \triangleq G$. For the sake of clarity, let us recall the basic steps of an iteration of GSA. Let J be the J–orthogonal matrix defined in (5.4.5). At the ith iteration, $i = 1, \ldots, m+2n-1$, a J–orthogonal matrix Θ_i, given by a sequence of Givens and hyperbolic rotations, is chosen such that $G_i \triangleq G_{i-1,Z}\Theta_i$ is in proper form, that is, the ith row of G_i has all the elements equal to zero with the exception of a single pivot element in the first column (the first $i-1$ rows of G_i are zero). Then the first column of G_i becomes the ith column of L. The generator matrix $G_{i,Z}$ at the next iteration is given by

$$G_{i,Z}(:,1) = ZG_i(:,1),$$

$$G_{i,Z}(:,[2:5]) = G_i(:,[2:5]).$$

Since the Givens and the hyperbolic rotations are used in updating and downdating a Cholesky decomposition [16], [11] respectively, we shall also refer to an application

of a Givens rotation to the columns of $G_{i-1,Z}$ as an *updating* and an application of a hyperbolic rotation to the columns of $G_{i-1,Z}$ as a *downdating* .

To preserve the "sparsity" of the generator matrix, we divide the algorithm into four phases:

(1) initialization: $i = 1$,

(2) the iterations for $i = 2 : m$,

(3) the iterations for $i = m + 1 : m + n - 1$,

(4) the iterations for $i = m + n : m + 2n - 1$.

Initialization: $i = 1$

The only vectors with the first entry different from 0 are g_1 and g_2. The new vectors $\tilde{g}_1$ and $\tilde{g}_2$ are computed as

$$[\tilde{g}_1 \quad \tilde{g}_2] = [g_1 \quad g_2]\, Q_0,$$

where Q_0 is the Givens rotation chosen to annihilate the first element of g_2. Hence, the first column of L is $\tilde{g}_1$. In the following we denote by $g_k^{(i)}, k = 1, \ldots, 5$, the vectors at the ith iteration. Then we define

$$\begin{aligned} g_1^{(1)} &= Z\tilde{g}_1 = [0, \tilde{g}_1(1 : m + n - 2)^T, 0, \tilde{g}_1(m + n : m + 2n - 2)^T]^T, \\ g_2^{(1)} &= \tilde{g}_2, \\ g_3^{(1)} &= g_3, \\ g_4^{(1)} &= g_4, \\ g_5^{(1)} &= g_5. \end{aligned}$$

The number of flops for this phase is $4n$.

Iterations for $i = 2 : m$

In each iteration of this phase $g_1^{(i-1)}$ is updated with $g_2^{(i-1)}$, and $g_3^{(i-1)}$, $g_4^{(i-1)}$ is updated with $g_5^{(i-1)}$, and $g_1^{(i-1)}$ is downdated with $g_4^{(i-1)}$. Since g_5 has m initial zeros, $g_5^{(1)}$ does not contribute to this phase.

Moreover, the structure of the vectors $g_k^{(i-1)}, k = 1, 2$, at the beginning of the ith iteration is

$$\begin{aligned} g_1^{(i-1)} &= [\underbrace{0, \ldots, 0}_{i-1}, \underbrace{*, \cdots, *}_{n}, *, \underbrace{0, \ldots, 0}_{m-i+1}, \underbrace{*, \cdots, *}_{n-1}]^T, \\ g_2^{(i-1)} &= [\underbrace{0, \ldots, 0}_{i-1}, \underbrace{*, \cdots, *}_{n-1}, \underbrace{0, \ldots, 0, 0}_{m-i+1}, \underbrace{*, \cdots, *}_{n}]^T, \end{aligned}$$

where the entries $*$ are in general different from 0. Then updating $g_1^{(i-1)}$ with $g_2^{(i-1)}$ fills in only the $(m + n)$th entry of the updated $g_1^{(i-1)}$. To complete the ith

iteration, $g_1^{(i-1)}$ must be updated with $g_3^{(i-1)}$ and downdated with $g_4^{(i-1)}$. To explain this computation, we describe the iteration for $i = 2$, recalling that the vectors $g_3^{(1)}$ and $g_4^{(1)}$ are equal, except for the $(m+n)$th element (the entries of these vectors are generally different from 0). Let

$$\begin{aligned} g_1^{(1)} &= [0, \xi_2, \ldots, \xi_{n+1}, \underbrace{0, \ldots, 0}_{m-2}, \xi_{m+n}, \ldots, \xi_{m+2n-1}]^T, \\ g_3^{(1)} &= [0, \zeta_2, \ldots, \zeta_{m+n-1}, \zeta_{m+n}, \zeta_{m+n+1}, \ldots, \zeta_{m+2n-1}]^T, \\ g_4^{(1)} &= [0, \zeta_2, \ldots, \zeta_{m+n-1}, \mu_{m+n}, \zeta_{m+n+1}, \ldots, \zeta_{m+2n-1}]^T. \end{aligned} \tag{5.4.6}$$

We observe that the $(m+n)$th entry of $g_4^{(1)}$ is equal to 0. The Givens rotation used during the updating is

$$Q_1 \triangleq \begin{bmatrix} c_G^{(1)} & -s_G^{(1)} \\ s_G^{(1)} & c_G^{(1)} \end{bmatrix},$$

with

$$c_G^{(1)} = \frac{\xi_2}{\sqrt{\xi_2^2 + \zeta_2^2}} \text{ and } s_G^{(1)} = \frac{\zeta_2}{\sqrt{\xi_2^2 + \zeta_2^2}}$$

such that

$$\tilde{g}_1^{(1)} = c_G^{(1)} g_1^{(1)} + s_G^{(1)} g_3^{(1)}, \tag{5.4.7}$$

$$\tilde{g}_3^{(1)} = -s_G^{(1)} g_1^{(1)} + c_G^{(1)} g_3^{(1)}, \tag{5.4.8}$$

where $\tilde{g}_1^{(1)}$ and $\tilde{g}_3^{(1)}$ denote the updated vectors with $\tilde{g}_1^{(1)} = [0, \tilde{\xi}_2, \ldots, \tilde{\xi}_{m+2n-1}]^T$ $\left(\tilde{\xi}_2 = \sqrt{\xi_2^2 + \zeta_2^2}\right)$. Moreover, we point out that, for (5.4.6),

$$\tilde{g}_3^{(1)}(n+2 : n+m-1) = c_G^{(1)} g_3^{(1)}(n+2 : n+m-1). \tag{5.4.9}$$

To finish the iteration for $i = 2$, $\tilde{g}_1^{(1)}$ must be downdated with $g_4^{(1)}$ by means of the stabilized hyperbolic rotation

$$H_1 \triangleq \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{1-\rho^2}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \rho \\ 0 & 1 \end{bmatrix},$$

where ρ is such that

$$\begin{bmatrix} \hat{\xi}_2 & 0 \end{bmatrix} = \begin{bmatrix} \tilde{\xi}_2 & \zeta_2 \end{bmatrix} H_1^T.$$

Taking (5.4.7) into account, it is straightforward to see that

$$\rho = -s_G^{(1)} \text{ and } \sqrt{1-\rho^2} = c_G^{(1)}.$$

The downdated vectors $\hat{g}_1^{(1)}$ and $\tilde{g}_4^{(1)}$ are

$$\hat{g}_1^{(1)} = \frac{\tilde{g}_1^{(1)} - s_G^{(1)} g_4^{(1)}}{c_G^{(1)}} = g_1^{(1)} + \frac{s_G^{(1)} \left(g_3^{(1)} - g_4^{(1)} \right)}{c_G^{(1)}} \tag{5.4.10}$$

and

$$\tilde{g}_4^{(1)} = -s_G^{(1)} \hat{g}_1^{(1)} + c_G^{(1)} g_4^{(1)}. \tag{5.4.11}$$

Hence,

$$\begin{aligned} \tilde{g}_4^{(1)} &= -s_G^{(1)} \hat{g}_1^{(1)} + c_G^{(1)} g_4^{(1)} \\ &= -s_G^{(1)} g_1^{(1)} + \frac{{c_G^{(1)}}^2 g_4^{(1)} - {s_G^{(1)}}^2 (g_3^{(1)} - g_4^{(1)})}{c_G^{(1)}} \\ &= -s_G^{(1)} g_1^{(1)} + \frac{g_4^{(1)} - (1 - {c_G^{(1)}}^2) g_3^{(1)}}{c_G^{(1)}}. \end{aligned} \tag{5.4.12}$$

From (5.4.8) and (5.4.12), $\tilde{g}_3^{(1)}$ and $\tilde{g}_4^{(1)}$ continue to be equal, except for the $(m+n)$th entry. Furthermore, from (5.4.10), we observe that $g_1^{(1)}$ and $\hat{g}_1^{(1)}$ differ in their $(n+m)$th entry. $\hat{g}_1^{(1)}$ now becomes the 2nd column of L, and, for the next iteration, the updated vectors are

$$\begin{aligned} g_1^{(2)} &= Z \hat{g}_1^{(1)} = \left[0, 0, \hat{g}_1^{(1)}(2 : m+n-2)^T, 0, \hat{g}_1^{(1)}(m+n : m+2n-2)^T \right]^T, \\ g_5^{(2)} &= g_5^{(1)}, \\ g_2^{(2)} &= g_2^{(1)}, \\ g_3^{(2)} &= \tilde{g}_3^{(1)}, \\ g_4^{(2)} &= [\tilde{g}_3^{(1)}(1 : m+n-1)^T, \gamma, \tilde{g}_3^{(1)}(m+n+1 : m+2n-1)^T]^T, \end{aligned}$$

where $\gamma = -s_G^{(1)} \hat{g}_1^{(1)}(m+n) + c_G^{(1)} g_4^{(1)}(m+n)$. To reduce the computational cost of this phase, we observe that it is not necessary to calculate $g_3^{(2)}(n+3 : m+n-1)$ since at the next iteration the corresponding entries of the vector $g_1^{(2)}$ are equal to 0. Hence for the vector $g_3^{(3)}(n+4 : m+n-1)$ the following relation holds:

$$g_3^{(3)}(n+4 : m+n-1) = c_G^{(2)} c_G^{(1)} g_3^{(1)}(n+4 : m+n-1)$$

and, at the ith iteration,

$$g_3^{(i-1)}(n+i : m+n-1) = c_G^{(i-2)} \cdots c_G^{(2)} c_G^{(1)} g_3^{(1)}(n+i : m+n-1).$$

Hence it is sufficient to store the partial product

$$c_G^{(i-2)} \cdots c_G^{(2)} c_G^{(1)} \tag{5.4.13}$$

into a temporary variable and multiply $g_3^{(1)}(n+i-1)$ with this variable at the beginning of the ith iteration. At the end of each iteration we set

$$L(i:m+2n-1,i) = \hat{g}_1^{(i-1)}(i:m+2n-1), \quad g_1^{(i)} = Z\hat{g}_i^{(i-1)}.$$

Hence the number of flops of this phase is $18mn$.

Iterations for $i = m+1 : m+n-1$

The only difference of this phase with the previous one is that $g_5^{(i-1)}$ must also be downdated from $g_1^{(i-1)}$. Let $\hat{g}_1^{(i-1)}$ be the first generator at the end of the ith iteration. We set

$$L(i:m+2n-1,i) = \hat{g}_1^{(i-1)}(i:m+2n-1), \quad g_1^{(i)} = Z\hat{g}_i^{(i-1)}.$$

The number of flops of this phase is $22.5n^2$.

Iterations for $i = m+n : m+2n-1$

This phase is similar to the previous one. The only difference is that the vector $g_4^{(i-1)}$ must also be computed, since now it differs from $g_3^{(i-1)}$. Let $\hat{g}_1^{(i-1)}$ be the first generator at the end of the ith iteration. Also after each iteration of this phase we set

$$L(i:m+2n-1,i) = \hat{g}_1^{(i-1)}(i:m+2n-1), \quad g_1^{(i)} = Z\hat{g}_i^{(i-1)}.$$

The number of flops of this phase is $12n^2$.

The Matlab-like code to compute the L factor by means of the described algorithm can be found in Appendix B.

5.4.2 Modified problem

We will now consider a slightly modified problem (5.2.1). The modification consists of the introduction of an error-free zero upper triangular part in the matrix A and by consequence E also has a zero upper triangular part. This modified problem typically arises if the system is assumed to be causal with zero initial state, implying that its inputs u_t are zero for $t \leq 0$. In this case, the first $n-1$ inputs are zero followed by m nonzero values. The latter means that $A \in \mathbb{R}^{m\times n}$ and $E \in \mathbb{R}^{m\times n}$ are as follows:

$$A = \begin{bmatrix} u_1 & 0 & \dots & 0 \\ u_2 & u_1 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & u_1 \\ \vdots & & & \vdots \\ u_m & u_{m-1} & \dots & u_{m-n+1} \end{bmatrix},$$

$$E = \begin{bmatrix} \alpha_1 & 0 & \dots & 0 \\ \alpha_2 & \alpha_1 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \alpha_1 \\ \vdots & & & \vdots \\ \alpha_m & \alpha_{m-1} & \dots & \alpha_{m-n+1} \end{bmatrix}.$$

By consequence, X becomes (remember that X is defined by $X\alpha = Ex$)

$$X = \begin{bmatrix} x_1 & & & & & \\ x_2 & x_1 & & & & \\ \vdots & \ddots & \ddots & & & \\ x_n & \ddots & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & x_n & \cdots & x_2 & x_1 \end{bmatrix} \in \mathbb{R}^{m\times m}.$$

Again we define M as

$$M = \begin{bmatrix} X & A+E \\ I & 0 \end{bmatrix} \in \mathbb{R}^{2m\times(m+n)}, \tag{5.4.14}$$

with $I \in \mathbb{R}^{m\times m}$ the identity matrix and $0 \in \mathbb{R}^{m\times n}$ the null matrix. The displacement rank of M^TM with respect to the block shift matrix $Z = Z_m \oplus Z_n$ is 5. Let $w_1 = M(:,1)/\|M(:,1)\|_2$ and $t_1 = M^Tw_1$. Let $w_2 = M(:,m+1)/\|M(:,m+1)\|_2$ and $t_2 = M^Tw_2$. Then the generators of M^TM with respect to Z are

$$\begin{aligned} g_1 &= t_1, \\ g_2 &= [0, t_2(2:m+n)^T]^T, \\ g_3 &= [0, t_2(2:m)^T, 0, t_2(m+2:m+n)^T]^T, \\ g_4 &= [0, t_1^T(2:m+n)]^T, \\ g_5 &= [0, M(m, 1:m-1)^T, 0, M(m, m+1:m+n-1)^T]^T, \end{aligned}$$

and

$$M^TM = \sum_{i=1}^{2} g_i g_i^T - \sum_{i=3}^{5} g_i g_i^T.$$

Also in this case, taking into account the sparsity of the vectors, g_1, g_4, g_5, and since g_2 and g_3 differ only in the $(m+1)$th entry, following the same technique described in section 5.4.1, it is possible to construct an algorithm for the fast triangularization of M requiring $18mn + 16.5n^2$ flops. As a matter of fact, the algorithm for solving this modified deconvolution problem[1] (which will be referred to as the *modified*

[1]Note that the problem described here is basically also a basic deconvolution problem, with a specific input sequence. However, a specific name is given to this problem and its corresponding algorithm so that they can easily be referred to.

deconvolution algorithm) has the same outline as described in section 5.2, when the appropriate matrices A, E, M, and X are used. The Matlab-like code of GSA to compute the L factor of the modified deconvolution problem can be found in Appendix B.

5.4.3 Stability of the computation of the R factor

The stability of the proposed generalized Schur algorithm is studied in Ch. 3. In phase 1 and phase 2 of the algorithm two hyperbolic rotations are performed in a stable way. In Ch. 3 it is proved that also in this case the generalized Schur algorithm performs reliably and the following result holds.

Theorem 5.4.1: *Let G be a generator matrix of M. Let L be the matrix of the L factor of the Cholesky decomposition of M^TM computed by means of the generalized Schur algorithm applying a sequence of Givens rotations and two mixed hyperbolic rotations per iteration. Then*

$$||M^TM - LL^T||_F \leq 80(m+2n-1)(m+2n-2)\varepsilon \left(2\sqrt{m+2n-1}||M||_F + ||G||_F^2\right).$$

Proof. See Ch. 3, § 3.3.2. ◊

A similar result holds for the L factor of the Cholesky decomposition of M^TM, computed by a GSA, implemented by means of the technique described in § 5.4, where M is the matrix defined in (5.4.14).

5.5 Numerical experiments

In this section we illustrate by means of two examples the efficiency of the algorithms described in section 5.4. To illustrate the statistical properties of the STLS estimator, a simulation experiment based on a medical application in renography (see Ch. 1, §1.3.3) is introduced. The next subsection describes the three examples that will be used in this section.

5.5.1 Examples

The first example is a typical deconvolution problem. Referring to Figure 1.1, we start from the exact impulse response $x = [1.9\ 3.3\ 4.4\ 5.4\ 5.9\ 6.2\ \ 6.3\ 6.4\ 6.5\ 6.45\ 6.3\ 6.2\ 6.1\ 5.9\ 5.8\ 5.7\ 5.6\ 5.4\ 5.3\ 5.0\ 4.85\ 4.6\ 4.0\ 3.4\ 2.8\ 1.7\ 1.3\ 1.0\ 0.2\ \ 0.0]^T \in \mathbb{R}^{30\times 1}$. The true but unmeasured input u is i.i.d. Gaussian noise of unit variance, except for the first 29 entries which equal 0. The latter implies that the first example is an illustration of the modified problem described in S 5.4.2. The true but unmeasured output y is calculated as a convolution of x and u. The measured input $u + \Delta u$ and output $y + \Delta y$ are obtained by perturbing u and y with i.i.d. Gaussian noise of variance $1e-4$.

The second example is similar to the previous one but now the exact impulse response x equals [1.9 3.3 4.4 5.4 5.9 6.2 6.3 6.1 5.8 5.6 5.3 5.0 4.85 4.6 4.0 3.4 1.8 1.0

$0.2\ 0.0]^T \in \mathbb{R}^{20\times 1}$. Again u and y are perturbed by i.i.d. Gaussian noise of unit variance, but this time there are no initial zeros. Thus the latter serves as an example for the basic deconvolution problem (and not the modified one). The measured input $u + \Delta u$ and output $y + \Delta y$ are obtained by perturbing u and y with i.i.d. Gaussian noise of variance $1e-4$.

The third example is a deconvolution problem that occurs in renography [44]. The goal here is to determine via deconvolution the so-called renal retention function of the kidney, which in system theoretic terms corresponds to the impulse response x of the system in Ch. 1, Figure 1.1. This retention function visualizes the mean whole kidney transit time of one unit of a tracer injected into the patient, and it enables a physician to evaluate the renal function and renal dysfunction severity after transplantation. In order to obtain this impulse response, the following experiment is conducted. A radioactive tracer is injected in an artery of the patient. The measured input of the system $u + \Delta u$ is the by noise perturbed arterial concentration of the radioactive tracer as a function of time. This concentration is measured by means of a gamma camera, and thus in discretized time $u_k + \Delta u_k$ represents the number of counts registered in the vascular region at the entrance of the kidney under study during the kth sampling interval. The measured output $y_k + \Delta y_k$ (the so-called renogram) represents the (by noise perturbed) number of counts registered in the whole kidney region by the gamma camera in the kth sampling interval. Deconvolution analysis of the renogram is based on modeling the kidney as a linear time-invariant causal system with zero-initial state. This is why we consider the modified problem, described in §5.4.2. Since this third example will be used to investigate the statistical properties of the STLS estimator, we have to use a simulation example instead of a real in vivo measurement. We will use the same simulation example as described in [130]. The noiseless input is described as follows:

$$u_{k+1} = Ae^{(-a_1 k\Delta t)} + Be^{(-a_2 k\Delta t)} + Ce^{(-a_3 k\Delta t)}, k = 0, 1, 2, \ldots, (t_{obs}/\Delta t) - 1,$$

with $A = 40.3$, $B = 45.2$, $C = 15.2$, $a_1 = 1.8$, $a_2 = 0.43$, and $a_3 = 0.035$. Δt represents the sampling interval, expressed in minutes, and t_{obs} is the total observation time. The exact impulse response is characterized by the following function:

$$x_{k+1} = \begin{cases} 1 & k = 0, 1, 2, \ldots, (t_1/\Delta t) - 1 \\ (k\Delta t - t_2)/(t_1 - t_2) & k = t_1/\Delta t, (t_1/\Delta t) + 1, \ldots, (t_2/\Delta t) - 1 \end{cases},$$

with $t_1 = 3$ minutes, $t_2 = 5$ minutes, and $t_{obs} = 20$ minutes. The noiseless output y is then obtained by convolution of the noiseless input u with the proposed impulse response x. In matrix format we have that

$$Ax = y,$$

where

$$A = \begin{bmatrix} u_1 & 0 & \dots & 0 \\ u_2 & u_1 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_1 \\ \vdots & \ddots & \ddots & \vdots \\ u_m & u_{m-1} & \dots & u_{m-n+1} \end{bmatrix} \in \mathbb{R}^{m \times n}.$$

For $\Delta t = \frac{1}{3}$ minutes, we have that $m \triangleq 3t_{obs} = 60$ and $n \triangleq 3t_2 = 15$. As described in [45], these functions and constants are a realistic simulation of real in vivo measurements.

5.5.2 Efficiency

In this subsection we first consider the dependence of the computational cost of the modified deconvolution algorithm for solving the modified problem described in §5.4.2 on the problem size (i.e., the dimensions $m \times n$ of the matrix A). Given the iterative nature of the algorithm, the total number of floating point operations (flops) is a multiple of the flops necessary to execute Step 2.1 of the modified deconvolution algorithm. Therefore, we will analyze this step in further detail.

In Step 2.1 we solve a LS problem by solving the corresponding normal equations

$$R^T \left(R \begin{bmatrix} \Delta\alpha \\ \Delta x \end{bmatrix} \right) = -M^T \begin{bmatrix} \beta \\ \alpha \end{bmatrix}, \tag{5.5.15}$$

where R is the triangular factor of the QR factorization of M. From (5.5.15), the description of the fast QR algorithm in §5.4.2 and the specific structure of M and R, we obtain the following theoretical number of computations for Step 2.1:

part 1: Calculation of the generators: $4mn + 9m + 0.5n^2 + 12.5n + 18$.

part 2: Construction of the right-hand side (see (5.5.15)): $4mn + 4m - n^2 - n + 8$.

part 3: Fast QR: $18mn + 16.5n^2$.

part 4: Solving a lower triangular system (see (5.5.15)): $4mn + 3m + n + 1$.

part 5: Solving an upper triangular system (see (5.5.15)): $4mn + 5m + n + 6$.

This means that overall the algorithm is $O(mn)$, when $m \gg n$, as is mostly the case. To illustrate the dependence on m and n, we consider three different cases based on the first example described in subsection 5.5.1. The first case (first row of Table 5.1) is constructed by generating $u \in \mathbb{R}^{600 \times 1}$. The second case (second row of Table 5.1) is constructed by generating $u \in \mathbb{R}^{1200 \times 1}$. The third case (third row of Table 5.1) is similar to the first case (and thus also derived from the first example described in subsection 5.5.1), but now only the first 15 elements of x are used to generate y with $u \in \mathbb{R}^{600 \times 1}$. In Table 5.1 we give the number of flops of the different parts of Step 2.1, as well as their sum. Flops are calculated using the Matlab command *flops*. Note that each cell of the table is divided into two parts: the upper part (bold number) contains the flop count obtained using the Matlab command *flops*,

Table 5.1. This table shows the number of flops for the different parts of Step 2.1 of the modified deconvolution algorithm, as well as their sum. The bold numbers show the flop count as measured by the Matlab *flops* command, whereas the other numbers represent the theoretical flop count.

$m \times n$	part 1	part 2	part 3	part 4	part 5	total
600 × 30	**78243**	**73478**	**352652**	**73831**	**75036**	**653240**
	78243	73478	338850	73831	75036	639438
1200 × 30	**155643**	**147878**	**690452**	**147631**	**150036**	**1291640**
	155643	147878	662850	147631	150036	1264038
600 × 15	**41718**	**38168**	**179507**	**37816**	**39021**	**336230**
	41718	38168	165713	37816	39021	322436

whereas the lower part gives the theoretical number of flops as determined at the beginning of this subsection. Note the close resemblance between the upper and lower part of each cell. Furthermore, from the table we can clearly see that the computational cost of Step 2.1 is $O(mn)$.

Table 5.2. This table shows the following ratios: the number of flops for the calculation of the R factor in alg_2 w.r.t. the number of flops for the calculation of the R factor in alg_1 ($flops_{alg_2}/flops_{alg_1}$) and the number of flops for the calculation of the R factor in alg_3 w.r.t. the number of flops for the calculation of the R factor in alg_1 ($flops_{alg_3}/flops_{alg_1}$), for matrices A of dimension $m \times n$. The bold numbers shows the flop count as measured by the Matlab *flops* command, whereas the other numbers come from the theoretical flop count.

$m \times n$	$flops_{alg_2}/flops_{alg_1}$	$flops_{alg_3}/flops_{alg_1}$
300 × 20	**1327.67**	**10.41**
	1432.27	10.93
600 × 20	**4949.15**	**13.19**
	5362.55	13.94
300 × 10	**2290.15**	**8.70**
	2671.12	9.76

To illustrate the better computational efficiency of the newly presented implementation of the basic deconvolution algorithm (referred to as alg_1), we compare its efficiency with that of the standard STLN approach [106], [131] (referred to as alg_2) and the faster STLN algorithm for Toeplitz STLS problems described in § 5.3 and [106] (referred to as alg_3). As an example we take the second example described in the previous subsection because this is a basic deconvolution problem (see also [131]). We use an example different from the previous paragraph since the algorithm alg_3 solves the basic deconvolution problem and not the modified one.

As a consequence alg_1 is the algorithm described in section 5.4.1. Table 5.2 shows three cases. The first ($m \times n = 300 \times 20$) and the second case ($m \times n = 600 \times 20$) are generated using the second example described in subsection 5.5.1, by generating, respectively, $u \in \mathbb{R}^{319\times 1}$ and $u \in \mathbb{R}^{619\times 1}$. The third case is also based on the second example described in subsection 5.5.1, but now $u \in \mathbb{R}^{309\times 1}$ and only the first 10 elements of the impulse response x are used to generate y. Since the algorithms only differ in the part where the fast (Q)R factorization is performed, Table 5.2 shows only the following figures: the number of flops for the calculation of the R factor in alg_2 w.r.t. the number of flops for the calculation of the R factor in alg_1 ($flops_{alg_2}/flops_{alg_1}$) and the number of flops for the calculation of the R factor in alg_3 w.r.t. the number of flops for the calculation of the R factor in alg_1 ($flops_{alg_3}/flops_{alg_1}$). Note that from section 5.4 and from the implementation in [106] we have the following theoretical flop counts for the (Q)R factorization:

alg_1: $18mn + 34.5n^2$,

alg_2: $3p^2(p - q/3), p = 2m + n - 1, q = m + 2n - 1$,

alg_3: $17mn + 3m^2 + 8mn^2$.

Again each cell in Table 5.2 is subdivided into an upper and a lower part. The upper part (bold numbers) contains the above–mentioned ratios obtained using the Matlab command *flops*, whereas the lower part contains the above–mentioned ratios based on the theoretical flop counts. We clearly see the computational advantage of alg_1 over alg_2 and alg_3, for different sizes $m \times n$ of the matrix A. Also notice the close resemblance between the theoretical and Matlab-based flop count.

5.5.3 Accuracy

We compare the statistical accuracy of the STLS estimator with that of the TLS estimator which in [130] was shown to outperform other estimators by far. To this end, we perform for each noise standard deviation σ_ν a Monte-Carlo simulation consisting of 100 runs. In every run, we add a different realization of i.i.d. white Gaussian noise with standard deviation σ_ν to the noiseless input u and the noiseless output y of the previously described medical simulation example. The obtained noisy vectors $u + \Delta u$ and $y + \Delta y$ are used as input to the modified deconvolution algorithm described at the beginning of §5.4.2. To compare the performance of both estimators at a noise level σ_ν, we average for both estimators the relative error $\frac{\|\Delta x\|_2}{\|x\|_2}$ over the different runs. Table 5.3 shows that in the case of the STLS estimator, the relative errors are 9% to 14% lower than in the case of the TLS estimator, confirming the statistical superior performance of the STLS estimator.

5.6 Conclusions

We have proposed new fast implementations of the algorithms that solve the basic and modified deconvolution problem in a ML sense. The algorithms solve the corresponding STLS problems in $O(mn)$ flops by exploiting the low displacement rank of the matrices involved in the basic deconvolution problem **and** the sparsity of the

Table 5.3. This table shows the relative error $\frac{\|\Delta x\|_2}{\|x\|_2}$ for the TLS and STLS estimator, averaged over 100 runs, for different noise standard deviations σ_ν.

σ_ν	TLS	STLS	σ_ν	TLS	STLS
0.05	0.0010	0.0009	0.5	0.0104	0.0091
0.071	0.0015	0.0013	0.707	0.0138	0.0123
0.087	0.0018	0.0016	0.866	0.0177	0.0153
0.1	0.0021	0.0018	1	0.0204	0.0180
0.158	0.0032	0.0029	1.58	0.0313	0.0279
0.224	0.0046	0.0041	2.24	0.0454	0.0402
0.274	0.0056	0.0049	2.74	0.0576	0.0513
0.316	0.0064	0.0057	3.16	0.0660	0.0601

corresponding generators.

By means of a deconvolution example, we illustrate the improved efficiency of our implementation of the basic deconvolution algorithm as compared to other implementations of algorithms for solving this type of STLS problem. We use a medical example in renography to illustrate the superior statistical performance of the STLS estimator as compared to other estimators. The proposed algorithm is faster than the algorithms known in the literature for solving STLS problems, even for small examples. It is evident that large speed-up will be achieved and be very attractive in large scale deconvolution problems.

Chapter 6

A FAST ALGORITHM FOR SUBSPACE STATE-SPACE SYSTEM IDENTIFICATION

6.1 Introduction

Two recent approaches [133], [134] in subspace identification problems require the computation of the R factor of the QR factorization of a block–Hankel matrix H, which, in general has a huge number of rows. Since the data are perturbed by noise, the involved matrix H is, in general, full rank. It is well known that, from a theoretical point of view, the R factor of the QR factorization of H is equivalent to the Cholesky factor of the correlation matrix $H^T H$, apart from a multiplication by a sign matrix. In [116] a fast Cholesky factorization of the correlation matrix, exploiting the block–Hankel structure of H, is described. In this chapter we consider a fast algorithm to compute the R factor based on the generalized Schur algorithm. The proposed algorithm allows to handle the rank–deficient case.

Subspace based system identification has become very popular in the last decade [39]. The success of this state–space identification approach is mainly due to the fact that it relies on a simple matrix decomposition for which reliable numerical algorithms are available. Its major drawback, on the other hand, is that large "data" matrices are involved and that this may lead to high computing and storage costs. We now briefly recall the basic formulation of the problem. Let u_k and y_k be the m–dimensional input vector and the l–dimensional output vector, respectively, of the linear time–invariant state–space model

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + w_k, \\ y_k &= Cx_k + Du_k + v_k, \end{aligned}$$

where x_k is the n–dimensional state vector at time k, $\{w_k\}$ and $\{v_k\}$ are state and output disturbances or noise sequences, and A, B, C and D are unknown real matrices of appropriate dimensions.

For non–sequential data processing, one chooses $N \gg 2(m+l)s$ and constructs the $N \times 2(m+l)s$ matrix $H = [U_{2s,N}^T Y_{2s,N}^T]$, where $U_{2s,N}$ and $Y_{2s,N}$ are block–Hankel matrices defined in terms of the input and output data, respectively :

$$U_{2s,N} = \begin{bmatrix} u_1 & u_2 & u_3 & \cdots & u_N \\ u_2 & u_3 & u_4 & \cdots & u_{N+1} \\ u_3 & u_4 & u_5 & \cdots & u_{N+2} \\ \vdots & \vdots & \vdots & & \vdots \\ u_{2s} & u_{2s+1} & u_{2s+2} & \cdots & u_{N+2s-1} \end{bmatrix},$$

$$Y_{2s,N} = \begin{bmatrix} y_1 & y_2 & y_3 & \cdots & y_N \\ y_2 & y_3 & y_4 & \cdots & y_{N+1} \\ y_3 & y_4 & y_5 & \cdots & y_{N+2} \\ \vdots & \vdots & \vdots & & \vdots \\ y_{2s} & y_{2s+1} & y_{2s+2} & \cdots & y_{N+2s-1} \end{bmatrix}.$$

Then the R factor of a QR factorization $H = QR$ is used for data compression. In [116] a fast Cholesky factorization of the correlation matrix, exploiting the block–Hankel structure of H, is described. In this chapter we consider a fast algorithm to

compute the R factor of the QR factorization of H based on the generalized Schur algorithm, exploiting its *displacement structure.*

The chapter does not deal with sequential or recursive processing of the input–output data; a detailed treatment of a recursive on–line identification technique, and comparisons with other techniques are included, e.g., in [26]. However, the software we developed for the fast QR approach can also cope with sequential data processing.

The chapter is organized as follows. In § 6.2 a stable computation of a generator matrix of H and the computational complexity of the generalized Schur algorithm for computing the R factor of the QR factorization of H are described. The rank–deficient case is described in §6.3 and some numerical experiments are reported in §6.4.

6.2 Fast computation of the R factor of the QR factorization of H

We show here how to compute a generator G of $H^T H$ where $H \in \mathbb{R}^{N,2(m+l)s}$ is the block Hankel matrix described in the first section with blocks of sizes $1 \times m$ and $1 \times l$:

$$H = \left[\begin{array}{cccc|cccc} u_1^T & u_2^T & \dots & u_{2s}^T & y_1^T & y_2^T & \dots & y_{2s}^T \\ u_2^T & \iddots & \iddots & \vdots & y_2^T & \iddots & \iddots & \vdots \\ \vdots & \iddots & \iddots & \vdots & \vdots & \iddots & \iddots & \vdots \\ u_N^T & \dots & \dots & u_{N+2s-1}^T & y_N^T & \dots & \dots & y_{N+2s-1}^T \end{array}\right].$$

The shift matrices used in this context are the matrices

$$Z_m \triangleq \left[\begin{array}{ccccc} 0_m & & & & \\ I_m & \ddots & & & \\ & \ddots & 0_m & & \\ & & I_m & 0_m & \end{array}\right], Z_l \triangleq \left[\begin{array}{ccccc} 0_l & & & & \\ I_l & \ddots & & & \\ & \ddots & 0_l & & \\ & & I_l & 0_l & \end{array}\right], \qquad (6.2.1)$$

$$Z \triangleq Z_m \oplus Z_l.$$

The following theorem then gives a construction of a generator for A.

Theorem 6.2.1: *Given the QR factorization of the first block columns :*

$$\left[\begin{array}{c|c} u_1^T & y_1^T \\ u_2^T & y_2^T \\ \vdots & \vdots \\ u_N^T & y_N^T \end{array}\right] = \left[\begin{array}{c} q_1^T \\ q_2^T \\ \vdots \\ q_N^T \end{array}\right] R_1 \qquad (6.2.2)$$

where R_1 can be assumed upper trapezoidal of row rank $k \leq m+l$ and $q_i \in \mathbb{R}^k$; define the product

$$\left[\begin{array}{c} C_{u,1}^T \\ \vdots \\ \hline C_{u,2s}^T \\ C_{y,1}^T \\ \cdots \\ C_{y,2s}^T \end{array}\right] \triangleq H^T \left[\begin{array}{c} q_1^T \\ q_2^T \\ \vdots \\ q_N^T \end{array}\right]. \tag{6.2.3}$$

Then a generator $\{G,\ J\}$ for $H^T H$ is given by

$$G \triangleq \left[\frac{G_u}{G_y}\right] = \left[\begin{array}{cc|cc} C_{u,1}^T & 0 & 0 & 0 \\ C_{u,2}^T & u_{N+1} & C_{u,2}^T & u_1 \\ \vdots & \vdots & \vdots & \vdots \\ C_{u,2s}^T & u_{N+2s-1} & C_{u,2s}^T & u_{2s-1} \\ \hline C_{y,1}^T & 0 & 0 & 0 \\ C_{y,2}^T & y_{N+1} & C_{y,2}^T & y_1 \\ \vdots & \vdots & \vdots & \vdots \\ C_{y,2s}^T & y_{N+2s-1} & C_{y,2s}^T & y_{2s-1} \end{array}\right], \tag{6.2.4}$$

where

$$J = \Sigma \oplus \Sigma$$

and

$$\Sigma = I_{k+1} \oplus -I_{k+1}.$$

Proof :
In order to prove the result we consider the displacement matrix $\nabla H^T H$:

$$\left[\begin{array}{c|c} U_{2s,N}U_{2s,N}^T - Z_m U_{2s,N}U_{2s,N}^T Z_m^T & U_{2s,N}Y_{2s,N}^T - Z_m U_{2s,N}Y_{2s,N}^T Z_l^T \\ \hline Y_{2s,N}U_{2s,N}^T - Z_l Y_{2s,N}U_{2s,N}^T Z_m^T & Y_{2s,N}Y_{2s,N}^T - Z_l Y_{2s,N}Y_{2s,N}^T Z_l^T \end{array}\right] \tag{6.2.5}$$

which ought to be equal to

$$\left[\begin{array}{c|c} G_u \Sigma G_u^T & G_u \Sigma G_y^T \\ \hline G_y \Sigma G_u^T & G_y \Sigma G_y^T \end{array}\right]. \tag{6.2.6}$$

It follows from (6.2.2,6.2.3) that $R_1 = [C_{u,1} C_{y,1}]$ and hence

$$\left[\begin{array}{c} C_{u,1}^T \\ \vdots \\ \hline C_{u,2s}^T \\ C_{y,1}^T \\ \cdots \\ C_{y,2s}^T \end{array}\right] \left[\begin{array}{cc} C_{u,1} & C_{y,1} \end{array}\right] = H^T \left[\begin{array}{c} q_1^T \\ q_2^T \\ \vdots \\ q_N^T \end{array}\right] R_1$$

which are the first block rows of the sub-blocks of (6.2.5). This thus verifies the first block rows and block columns of the equality between (6.2.5) and (6.2.6). The rest easily follows from the block Hankel structure of H. $\Diamond$

Note that if the first block columns of H in (6.2.2) have full rank then R_1 is square invertible and $k = m + l$. If moreover the whole matrix H has full column rank, then the generalized Schur algorithm will not encounter any singularities. But since the low rank case is of particular interest here, singularities in the generalized Schur algorithm will be encountered and hence lead to a lower complexity of the algorithm.

The above theorem also shows that the displacement rank of $H^T H$ is at most $2(k+1) \leq 2(m+l+1)$, with the same number of positive and negative generators. Hence the generalized Schur algorithm to compute the R factor requires about $(8Nrk)$ flops, where r is the rank of H. To compute the generator G of $H^T H$, the QR factorization (6.2.2) requires $(6N(m+l)^2)$ flops and the product (6.2.3) requires less than $(4Nk(m+l)s)$ flops. We recall that $k \leq (m+l)$ and $r \leq 2(m+l)s$ but that equality is obtained when no rank deficiency is detected. The most time consuming steps are then clearly the generalized Schur algorithm and the product (6.2.3).

6.3 The generalized Schur algorithm for rank–deficient matrices

The generalized Schur algorithm allows to handle rank deficient matrices $H^T H$. In this case we can drop some rows of the generator during the algorithm. For this we need a tolerance, say $\delta \triangleq \eta \|H^T H\|$ where η is the requested relative accuracy. We refer to the description of GSA Ch. 3, § 3.2.1, where only a hyperbolic rotation in factored form is performed at each iteration.

Reducing the positive and negative generator in proper form at the i–th iteration, we denote by $g_{p,i}$ and $g_{n,i}$ the i–th entries of the positive and negative pivot generators, respectively. We test if $g_{p,i}^2 - g_{n,i}^2 \leq \delta$. Let $[a_{11}\ \mathbf{a}_{12}]$ be the leading row of the current Schur complement. We then check as well if the norm of $\mathbf{a}_{12}$ is small. If so, the currently computed row of the Cholesky factor is negligible and we delete the two corresponding rows of the generator. It is possible that $\mathbf{a}_{12}$ is much larger than δ although $a_{11} \leq \delta$. In this case the deletion of a row of the Cholesky factor will yield residual errors $\|H^T H - R^T R\|$ of the same size. This is analyzed in this section. From the first example we can conclude that the described procedure works accurately when it is applied to a matrix H with a *sufficiently large* gap between *significant* singular values and *negligible* ones. On the other hand, a loss of accuracy in the computed factor R is observed when the distribution of the *small* singular values of H shows a uniform and slow decrease. The relative accuracy η is chosen equal to 10^{-13} in both examples.

Example 6.3.1: *Consider the matrix* $H = [U^T | Y^T]$, *with* $Y = U$, *where the first row and the last column of* U *are*

$$[\ 40 \quad 39 \quad 38 \quad \ldots \quad 3 \quad 2 \quad 1 \quad 2 \quad 2 \quad 3\ ],$$

Table 6.1. Numerical results for Example 6.3.1

R_M # flops	R_S # flops	backward error R_M	backward error R_S	numerical rank
31660	9489	1.51×10^{-16}	5.20×10^{-15}	5

$$[\ 3 \quad 2 \quad 2 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7\ ]^T,$$

respectively. The rank of the matrix H *is* 5 *and* $\|H^TH\|_1 = 6.31 \times 10^5$.

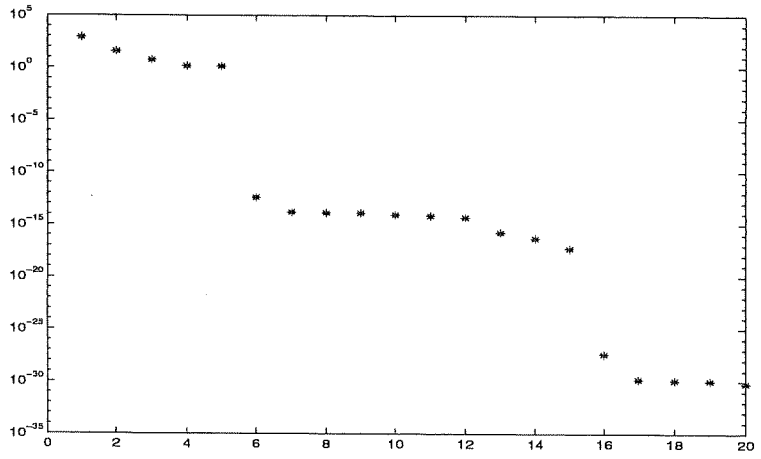


Figure 6.1. Distribution of the singular values, in logarithmic scale, of the matrix considered in Example 6.3.1

In table 6.1 the results of the computation of the R factor of the matrix H by means of the standard QR and the generalized Schur algorithm are shown. We denote by R_M, R_S, *backward error* R_*, *and numerical rank, the R factor of the QR factorization of H computed by the* `matlab` *function* `triu(qr(H))` *and by the generalized Schur algorithm, the backward error of* H^TH *defined as*

$$\frac{\|H^TH - R_*^TR_*\|_1}{\|H^TH\|_1},$$

and the rank of H detected by the generalized Schur algorithm, respectively. In this case, the R factor is accurately computed by the generalized Schur algorithm, because of the big difference between the significant singular values and the negligible ones of H (see Fig. 6.1).

Example 6.3.2: *This is application #4 : Mechanical flutter, considered in the next section. In Fig. 6.2 we can see that the distribution of the small singular values of the involved matrix H slightly decreases. We point out that the correlation matrix H^TH computed by* `matlab` *is not numerically s.p.d. because of the nearly rank deficiency of H. Furthermore $\|H^TH\|_1 = 3.99 \times 10^4$. So, in this case the fast Cholesky factorization, exploiting the block–Hankel structure of H and described in [116], can not be used. In table 6.2 we can see that, although the generalized Schur algorithm is very fast w.r.t. the standard QR algorithm, the achieved accuracy is not satisfactory.*

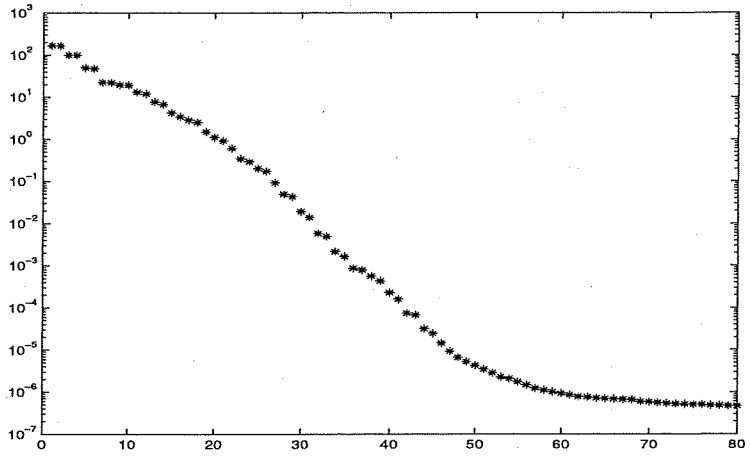


Figure 6.2. Distribution of the singular values, in logarithmic scale, of the matrix considered in Example 6.3.2

Table 6.2. Numerical results for Example 6.3.2

R_M # flops	R_S # flops	backward error R_M	backward error R_S	numerical rank
12.49×10^6	40.61×10^4	1.27×10^{-14}	2.92×10^{-2}	18

6.4 Numerical results

In this section results computing the R matrix by means of the generalized Schur algorithm are summarized. The data sets considered are publicly available on the DAISY web site

`http://www.esat.kuleuven.ac.be/sista/daisy`

At each iteration of the generalized Schur algorithm, two Householder matrices and one modified hyperbolic rotation are computed in order to reduce the generator in proper form. All the numerical results have been obtained on a Sun workstation Ultra 5 using `matlab 5.3`.

Table 6.3 gives a summary description of the applications considered in our comparison, indicating the number of inputs m, the number of outputs l, the number of block rows s, the total number of data samples used t and the number of rows of H.

Table 6.3. Summary description of applications.

Appl. #	Application	m	l	s	N
1	Glass tubes	2	2	20	1361
2	Labo dryer	1	1	15	970
3	Glass oven	3	6	10	1227
4	Mechanical flutter	1	1	20	960
5	Flexible robot arm	1	1	20	984
6	Evaporator	3	3	10	6285
7	CD player arm	2	2	15	2018
8	Ball and beam	1	1	20	960
9	Wall temperature	2	1	20	1640

Table 6.4. Comparative results for the computation of the R factor.

Appl. #	Application	R_M # flops	R_S # flops	backward error R_S	Rel.residual
1	Glass tubes	6.76×10^7	2.61×10^6	2.20×10^{-15}	8.73×10^{-14}
2	Labo dryer	7.01×10^6	3.36×10^5	8.30×10^{-15}	9.48×10^{-13}
3	Glass oven	7.63×10^7	6.38×10^6	3.73×10^{-15}	7.91×10^{-12}
4	Mechanical flutter	1.25×10^7	4.89×10^3	2.92×10^{-02}	$0.39 \times 10^{+00}$
5	Flexible robot arm	1.25×10^7	4.76×10^5	4.13×10^{-15}	3.38×10^{-05}
6	Evaporator	1.82×10^8	1.11×10^7	6.26×10^{-15}	5.13×10^{-14}
7	CD player arm	5.77×10^7	2.59×10^6	5.01×10^{-15}	2.08×10^{-08}
8	Ball and beam	1.22×10^7	4.67×10^5	7.59×10^{-15}	6.79×10^{-13}
9	Wall temperature	4.67×10^7	1.64×10^6	2.45×10^{-14}	3.76×10^{-12}

In Table 6.4 some results for the computation of the R factor of the QR factorization of H are presented. Rel. residual denotes

$$\frac{\| |R_M| - |R_S| \|_1}{\| |R_M| \|_1},$$

where R_M and R_S have been defined in §6.3.

The results in Table 6.4 are comparable with those described in [116], where the R factor is obtained considering the Cholesky factorization of the correlation matrix

$H^T H$, and exploiting the block-Hankel structure of H. The analysis of application #4 : Mechanical flutter, is described in Example 6.3.2 of the previous section.

6.5 Conclusions

In this chapter the generalized Schur algorithm to compute the R factor of the QR factorization of block–Hankel matrices, arising in some subspace identification problems, is described.

It is shown that the generalized Schur algorithm is significantly faster than the classsical QR factorization. A rank–revealing implementation of the generalized Schur algorithm in case of rank–deficient matrices is also discussed.

Algorithmic details and numerical results have been presented.

Chapter 7

COMPUTING THE SMALLEST EIGENVALUE OF A SYMMETRIC POSITIVE DEFINITE TOEPLITZ MATRIX

7.1 Introduction

Recent progress in signal processing and estimation has generated considerable interest in the problem of computing the minimal eigenvalue of Toeplitz matrices (see, i.e., Ch. 1, § 1.5). The fundamental modelling and solution that led to this are due to Pisarenko [109], while more recently numerous authors have discussed the computational aspects of the problem [58], [59], [31], [32], [33], [86].

Algorithms already exist for computing the smallest eigenvalue of a symmetric positive definite Toeplitz (SPDT) matrix (see, e.g., [32], [33], [58], [59], [86], [135]). The fundamental algorithm of [33] is based on using bisection to find a point between the smallest eigenvalue of the $n \times n$ SPDT matrix and that of its $(n-1) \times (n-1)$ principal submatrix, and then to improve that point using Newton's method for the determination of a zero of the secular equation. We will refer to the Newton portion of that algorithm as the CVN method. Replacement of the linearization in Newton's method by more involved rational local models of the secular equation leads to increased performance in the recent paper [86]. Most of the cited algorithms use Levinson-Durbin solutions of sequences of Yule-Walker Systems and converge from the right to the eigenvalue.

Similarly, the method presented in § 3 combines a modified version of the Levinson–Durbin algorithm. This is used, however, to compute the step of Newton's method as applied to the characteristic polynomial of the Toeplitz matrix. The method can start from 0, and converges from the left to the minimum eigenvalue. Though it is generally not faster than the method from [86], it is conceptually much simpler. In §4 we note that this method can be used to find an initial value for the CVN method. In §5 we describe how to compute an approximation of the associated eigenvector. In §6 we report some numerical results.

7.2 Characteristic Polynomial Associated to a Symmetric Positive Definite Toeplitz Matrix

In the sequel we use

$$T_i = \begin{bmatrix} t_0 & t_1 & \ddots & t_{i-1} \\ t_1 & t_0 & \ddots & \ddots \\ \ddots & \ddots & \ddots & t_1 \\ t_{i-1} & \ddots & t_1 & t_0 \end{bmatrix}, \quad \text{and} \quad E_i = \begin{bmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{bmatrix}_{i \times i}, \tag{7.2.1}$$

as the notation for an $i \times i$ SPDT matrix T_i and exchange ("flip") matrix E_i, respectively, for $i = 1, \ldots, n$. We also use $t^{(i)} = [t_1, t_2, \ldots, t_{i-1}, t_i]^T$ to denote the vector of entries in T_{i+1} (omitting the diagonal entry), and I_i to denote the $i \times i$ identity matrix. We let $p_i(\lambda) = \det(T_i - \lambda I_i)$ denote the characteristic polynomial for T_i, and let $\lambda_i(T_i) \leq \lambda_{i-1}(T_i) \leq \cdots \leq \lambda_1(T_i)$ denote the eigenvalues of T_i.

Proposition 7.2.1: *The following relations hold*

$$\begin{cases} p_i(\lambda) > 0, \\ p_i'(\lambda) < 0, \quad 1 \le i \le n, \ \lambda < \lambda_n(T_n), \\ p_i''(\lambda) > 0, \end{cases} \tag{7.2.2}$$

$$p_{i+1}(\lambda) = p_i(\lambda)(t_0 - \lambda - {t^{(i)}}^T E_i (T_i - \lambda I_i)^{-1} E_i t^{(i)}). \tag{7.2.3}$$

Proof. We observe that

$$p_i(\lambda) = \prod_{j=1}^{i} (\lambda_j(T_i) - \lambda), \ i = 1, \ldots, n.$$

Then, by the interlacing property of the SPDT matrices T_i, $i = 1, \ldots, n$, (7.2.2) follows since

$$\lambda_n(T_n) \le \lambda_i(T_i), 1 \le \ i \le n.$$

(7.2.3) follows since

$$T_{i+1} - \lambda I_{i+1} = \begin{bmatrix} T_i - \lambda I_i & E_i t^{(i)} \\ {t^{(i)}}^T E_i & t_0 - \lambda \end{bmatrix},$$

so that

$$\det(T_{i+1} - \lambda I_{i+1}) = \det(T_i - \lambda I_i) \left(t_0 - \lambda - {t^{(i)}}^T E_i \left(T_i - \lambda I_i\right)^{-1} E_i t^{(i)} \right)$$

. ◊

7.3 Construction of the Algorithm

The smallest eigenvalue of the SPDT matrix T_n is the smallest zero of p_n. The conditions (7.2.2) for p_n imply that Newton's method applied to p_n will converge to $\lambda_n(T_n)$ starting at any point in the open interval $(-\infty, \lambda_n(T_n))$ [123]. Since $\lambda_n(T_n) > 0$, we can choose the starting point to be $\max\{0, t_0 - \sum_{i=1}^{n-1} |t_i|\}$. To use this method we need to compute p_n and p_n'. By (7.2.3) it follows that

$$\begin{aligned} p_{i+1}'(\lambda) \ = \ & p_i'(\lambda) \left(t_0 - \lambda - {t^{(i)}}^T E_i (T_i - \lambda I_i)^{-1} E_i t^{(i)} \right) \\ & - p_i(\lambda) \left(1 + {t^{(i)}}^T E_i (T_i - \lambda I_i)^{-2} E_i t^{(i)} \right) \end{aligned} \tag{7.3.4}$$

Let

$$\beta_1 = t_0 - \lambda$$

$$\beta_{i+1} = t_0 - \lambda - {t^{(i)}}^T E_i (T_i - \lambda I_i)^{-1} E_i t^{(i)} \quad \text{for } i = 1, \ldots, n-1,$$

$$y^{(i)} = -(T_i - \lambda I_i)^{-1} E_i t^{(i)}$$

then the recurrences (7.2.3) and (7.3.4) become

$$p_{i+1}(\lambda) = \beta_{i+1}p_i(\lambda) \tag{7.3.5}$$

$$p'_{i+1}(\lambda) = \beta_{i+1}p'_i(\lambda) - p_i(\lambda)(1 + {y^{(i)}}^T y^{(i)}). \tag{7.3.6}$$

7.3.1 Newton's Method

To implement Newton's method we need to construct a recurrence relation to compute (7.3.5) and (7.3.6). This can be done if we know the solution of the *modified* YW systems

$$(T_i - \lambda I_i)y^{(i)} = -E_i t^{(i)}. \tag{7.3.7}$$

The Levinson–Durbin algorithm described in [53, p196] allows us to solve a YW system

$$(T_i - \lambda I_i)z^{(i)} = -t^{(i)} \tag{7.3.8}$$

given the solution to

$$(T_{i-1} - \lambda I_{i-1})z^{(i-1)} = -t^{(i-1)},$$

the same system of dimension $i-1$. The solutions $y^{(i)}$ and $z^{(i)}$ of (7.3.7) and (7.3.8) are related by the following relation

$$y^{(i)} = E_i z^{(i)}. \tag{7.3.9}$$

Indeed,

$$z^{(i)} = -(T_i - \lambda I_i)^{-1} t^{(i)}$$

and

$$y^{(i)} = -(T_i - \lambda I_i)^{-1} E_i t^{(i)}.$$

Furthermore,

$$(T_i - \lambda I_i)^{-1} E_i = E_i (T_i - \lambda I_i)^{-1}$$

since $(T_i - \lambda I_i)^{-1}$ is persymmetric, and thus (7.3.9) follows.

For the sake of completeness we report how the vector $y^{(i)}$ is computed, given the solution $y^{(i-1)}$ to the system

$$(T_{i-1} - \lambda I_{i-1})y^{(i-1)} = -E_{i-1} t^{(i-1)}.$$

We show how to compute the solution of the system (7.3.7), for $i = 2, \ldots, n-1$.

Partition $y^{(i)} = [\alpha_i, {z^{(i)}}^T]^T$. We can write (7.3.7) as follows:

$$\left[\begin{array}{c|c} t_0 - \lambda & {t^{(i-1)}}^T \\ \hline t^{(i-1)} & T_{i-1} - \lambda I_{i-1} \end{array}\right] \left[\begin{array}{c} \alpha_i \\ \hline z^{(i)} \end{array}\right] = -E_i t^{(i)} = -\left[\begin{array}{c} t_i \\ \hline E_{i-1} t^{(i-1)} \end{array}\right] \tag{7.3.10}$$

Then the individual partitions of $y^{(i)}$ are, with $\beta_i = t_0 - \lambda + {t^{(i-1)}}^T E_{i-1} y^{(i-1)}$:

$$\alpha_i = -\frac{t_i + {t^{(i-1)}}^T y^{(i-1)}}{t_0 - \lambda + {t^{(i-1)}}^T E_{i-1} y^{(i-1)}}, \quad z^{(i)} = \alpha_i E_{i-1} y^{(i-1)} + y^{(i-1)}, \tag{7.3.11}$$

We follow the same steps as in [53, p195]:

$$\begin{aligned}
\beta_i &= t_0 - \lambda + \begin{bmatrix} t^{(i-2)^T} & t_{i-1} \end{bmatrix} E_{i-1} \begin{bmatrix} \alpha_{i-1} \\ \alpha_{i-1} E_{i-2} y^{(i-2)} + y^{(i-2)} \end{bmatrix} \\
&= t_0 - \lambda + \begin{bmatrix} t^{(i-2)^T} & t_{i-1} \end{bmatrix} \begin{bmatrix} \alpha_{i-1} y^{(i-2)} + E_{i-2} y^{(i-2)} \\ \alpha_{i-1} \end{bmatrix} \\
&= t_0 - \lambda + t^{(i-2)^T} E_{i-2} y^{(i-2)} + \alpha_{i-1} [t^{(i-2)^T} y^{(i-2)} + t_{i-1}] \\
&= \beta_{i-1} + \alpha_{i-1} [-\beta_{i-1} \alpha_{i-1}]
\end{aligned}$$

to arrive at the following recurrence:

$$\begin{aligned}
&\beta_1 = t_0 - \lambda, \\
&\beta_i = (1 - \alpha_i^2)\beta_{i-1}, \qquad i = 2, \ldots, n.
\end{aligned} \tag{7.3.12}$$

At each pass through Newton's method we need to compute initially $p_n(\lambda)$, $p_n'(\lambda)$ and subsequently the ratio $p_n(\lambda)/p_n'(\lambda)$. To avoid numerical cancellation it is preferable to construct a recurrence relation for the rational function $\phi_i(\lambda) = p_i(\lambda)/p_i'(\lambda)$ directly. By (7.3.5) and (7.3.6),

$$\begin{aligned}
\frac{p_i(\lambda)}{p_i'(\lambda)} &= \frac{\beta_i p_{i-1}(\lambda)}{\beta_i p_{i-1}'(\lambda) - p_{i-1}(\lambda)\left(1 + \|y^{(i-1)}\|_2^2\right)} \\
&= \frac{\beta_i}{\beta_i \frac{p_{i-1}'(\lambda)}{p_{i-1}(\lambda)} - \left(1 + \|y^{(i-1)}\|_2^2\right)}
\end{aligned}$$

We thus have the following recurrence relation for the ϕ_i:

$$\begin{aligned}
&\phi_1(\lambda) = -\beta_1, \\
&\phi_{i+1}(\lambda) = \beta_{i+1}/[\beta_{i+1}/\phi_i(\lambda) - 1 - \|y^{(i)}\|_2^2], \qquad i = 1, \ldots, n-1.
\end{aligned}$$

7.3.2 The Algorithm

We summarize the overall algorithm in a `matlab`–like[1] style as follows:

Algorithm 7.3.1 (Newton's Method)

$k = -1;$

[1] Specifically `flipud`$(y(1:i-1))$ is the `matlab` implementation of $E_{i-1}y^{(i-1)}$.

$\lambda^{[0]} = \max\{0, t_0 - \|t(1:n-1)\|_1\}$;
$\beta_n = t_0 + \|t(1:n-1)\|_1$;
`toll` = *a given zero tolerance;*
`while` $(\beta_n >$ `toll`$)$
$k = k + 1$;
$\beta_1 = t_0 - \lambda^{[k]}$; $\phi_0(\lambda^{[k]}) = -\beta_1$;
`for` $i = 1 : n-1$,
$\alpha_i = -[t(i) + y(1:i-1)^T * t(1:i-1)]/\beta_i$;
$z(1:i-1) = y(1:i-1) + \alpha_i *$ `flipud`$(y(1:i-1))$;
$y(1) = \alpha_i$; $y(2:i) = z(1:i-1)$;
$\beta_{i+1} = \beta_i * (1 - \alpha_i^2)$;
$\phi_{i+1}(\lambda^{[k]}) = \beta_{i+1}/(\beta_{i+1}/\phi_i(\lambda^{[k]}) - 1 - \|y(1:i)\|_2^2)$;
`end`
$\lambda^{[k+1]} = \lambda^{[k]} - \phi_n(\lambda^{[k]})$;
`end`

$\diamondsuit$

Each iteration of Newton's method requires $3n^2 + O(n)$ flops[2], and our experiments seem to indicate that the number of Newton iterations appears to grow very slowly with n, if at all. The adopted stopping criterion $\beta_n \leq$ `toll` guarantees that $|\lambda_n(T_n) - \lambda^{[k]}| \leq$ `toll`. Indeed, by (7.3.5),

$$\beta_n = \frac{p_n(\lambda^{[k]})}{p_{n-1}(\lambda^{[k]})} = (\lambda_n(T_n) - \lambda^{[k]}) \prod_{j=1}^{n-1} \frac{\lambda_j(T_n) - \lambda^{[k]}}{\lambda_j(T_{n-1}) - \lambda^{[k]}}.$$

Since $\lambda_n(T_n) - \lambda^{[k]} > 0$ by construction of $\lambda^{[k]}$ and since from the interlacing property $\lambda_j(T_n) \geq \lambda_j(T_{n-1})$, $j = 1, \ldots, n-1$, all the factors of the right hand side product are not less than unity, one infers that $\beta_n \geq |\lambda_n(T_n) - \lambda^{[k]}|$.

7.4 Starting Point for the CVN Method

The numerical method proposed in [33] combines the bisection and Newton's method applied to the rational function p_n/p_{n-1} (as opposed to the method of §3 in which Newton is applied implicitly to p_n itself). The crucial point of this method is to find an initial point for Newton's method in the interval $[\lambda_n(T_n), \lambda_{n-1}(T_{n-1}))$ to generate a monotonic sequence that converges to $\lambda_n(T_n)$ from the right. Here we show how a point in this interval can be computed by means of Newton's method proposed in §3.

Lemma 7.4.1: *Let* $\lambda^{[k]} \in [0, \lambda_n(T_n))$ *and*

$$\lambda_L = \lambda^{[k]} - p_n(\lambda^{[k]})/p_n'(\lambda^{[k]}),$$

[2]We use the definition of flop given in [53, p18].

$$\lambda_{\mathrm{H}} = \lambda^{[k]} - p_{n-1}(\lambda^{[k]})/p'_{n-1}(\lambda^{[k]}).$$

Then

$$\lambda_{\mathrm{L}} \leq \lambda_{\mathrm{H}}.$$

Proof. We prove that

$$p_{n-1}(\lambda^{[k]})/p'_{n-1}(\lambda^{[k]}) \leq p_n(\lambda^{[k]})/p'_n(\lambda^{[k]}),$$

that is,

$$\left(\sum_{j=1}^{n-1} \frac{1}{\lambda_j(T_{n-1}) - \lambda^{[k]}}\right)^{-1} \leq \left(\sum_{j=1}^{n} \frac{1}{\lambda_j(T_n) - \lambda^{[k]}}\right)^{-1}.$$

By the interlacing property,

$$\lambda_j(T_{n-1}) \leq \lambda_{j+1}(T_n), \quad j = 1, \ldots, n-1.$$

Then

$$(\lambda_j(T_{n-1}) - \lambda^{[k]})^{-1} \geq (\lambda_{j+1}(T_n) - \lambda^{[k]})^{-1}, \quad j = 1, \ldots, n-1,$$

and, since

$$\sum_{j=1}^{n-1} \frac{1}{\lambda_j(T_{n-1}) - \lambda^{[k]}} \geq \sum_{j=1}^{n} \frac{1}{\lambda_j(T_n) - \lambda^{[k]}},$$

then

$$\lambda_{\mathrm{L}} \leq \lambda_{\mathrm{H}}.$$

◊

The λ_{H} considered in the above lemma is less than or equal to $\lambda_{n-1}(T_{n-1})$. To determine if $\lambda_{\mathrm{H}} > \lambda_n(T_n)$ we must consider the Sylvester law of Inertia with respect to $T_n - \lambda_{\mathrm{H}} I_n$. In [32] it is proved that the numbers of negative, zero and positive $\beta_i, i = 1, \ldots, n$ (7.3.12) are the numbers of the eigenvalues smaller, equal, and greater than λ_{H}, respectively. Since $\lambda_{\mathrm{H}} < \lambda_{n-1}(T_{n-1})$, it must be that $\beta_i > 0, i = 1, \ldots, n-1$, but β_n is free. If $\beta_n = 0$, then $\lambda_{\mathrm{H}} = \lambda_n(T_n)$ and we have found the desired eigenvalue. If $\beta_n > 0$, then $\lambda_{\mathrm{H}} < \lambda_n(T_n)$, and we can continue the iteration using the usual Newton iterate $\lambda^{[k+1]} = \lambda_{\mathrm{L}}$ or accelerate the iteration towards $\lambda_{n-1}(T_{n-1})$ by setting $\lambda^{[k+1]} = \lambda_{\mathrm{H}}$ instead. If $\beta_n < 0$, then $\lambda_{\mathrm{H}} \in (\lambda_n(T_n), \lambda_{n-1}(T_{n-1}))$, so λ_{H} is a suitable initial point for the CVN method. Furthermore, we have already solved the linear system

$$(T_{n-1} - \lambda_{\mathrm{H}} I_{n-1}) y^{(n-1)} = -E_{n-1} t^{(n-1)},$$

so the first iteration of the CVN method can be carried out almost for free:

$$\lambda^{[1]} = \lambda_{\mathrm{H}} + \frac{t_0 - \lambda_{\mathrm{H}} + {y^{(n-1)}}^T t^{(n-1)}}{1 + \|y^{(n-1)}\|_2^2}.$$

7.5 Computing the Eigenvector associated with $\lambda_n(T_n)$

Let $\tilde{\lambda}$ be the computed approximation of $\lambda_n(T_n)$ by means of one of the proposed methods. An approximation of the corresponding eigenvector can be obtained with inverse iteration by solving the system

$$(T_n - \tilde{\lambda} I_n) y^{(n)} = -E_n \tilde{t}^{(n)},$$

where $\tilde{t}^{(n)} = [t_1, t_2, \ldots, t_{n-1}, \mu]^T$ for some value of μ. Assuming $\lambda_n(T_n) < \lambda_{n-1}(T_{n-1})$, the first and last components of the desired eigenvector associated with $\lambda_n(T_n)$ are nonzero [33], so this eigenvector will not be orthogonal to $\tilde{t}^{(n)}$ for most values of μ, and for simplicity we have chosen $\mu = 0$ in our experiments. Since we have already solved the system

$$(T_{n-1} - \tilde{\lambda} I_{n-1}) y^{(n-1)} = -E_n t^{(n-1)}$$

while computing the eigenvalue $\tilde{\lambda}$, it follows from (7.3.11) that

$$y^{(n)} = \begin{bmatrix} \alpha_n \\ z^{(n)} \end{bmatrix} = \begin{bmatrix} [\mu + t^{(n-1)^T} y^{(n-1)}]/\beta_n \\ y^{(n-1)} + \alpha_n E_{n-1} y^{(n-1)} \end{bmatrix}$$

can be obtained with a computational cost of $4n$ flops. Denoting by

$$\tilde{x} = y^{(n)} / \|y^{(n)}\|_2$$

the normalized approximation of the eigenvector and by

$$r(\tilde{\lambda}, \tilde{x}) = T_n \tilde{x} - \tilde{\lambda} \tilde{x}$$

the residual, we notice that one step of the inverse iteration is sufficient to obtain an approximation of the eigenvector such that the residual norm is as small as the accuracy of the computed eigenvalue [60].

7.6 Numerical Results

In this section we present some numerical results. We test the presented algorithm with the matrices considered in [33]. More precisely, for each $n = 11, 21, \ldots, 91$ and $n = 16, 32, 64, \ldots, 512$, we generated 25 random SPDT matrices. These matrices have the form $T = m \sum_{k=1}^{n} w_k T_{2\pi\theta_k}$, where n is the dimension, m is chosen so that T is normalized, $T_\theta = (t_{ij})_{=}(\cos((i-j)\cdot\theta))$, and w_k and θ_k are uniformly distributed random numbers taken from $[0, 1]$ generated by the `matlab` function `rand`.

In Table 7.1 we report the results obtained by computing the smallest eigenvalue of SPDT matrices with the proposed Newton's method of § 3, denoted by N, with the method described in § 4, denoted by N + NCV, and with the algorithm of Cybenko and Van Loan, denoted by CV(1), choosing their Method 1 to compute the initial interval (see [33] for details). For the second and the third method

Table 7.1. Average behavior of the algorithms using 25 random examples for each dimension.

n	N		N+NCV			CV(1)		
	N steps	flops	N steps	NCV steps	flops	bis. steps	CV steps	flops
11	6.48	2.70×10^3	1.44	6.92	3.61×10^3	4.64	6.80	3.83×10^3
21	7.04	1.01×10^4	2.04	7.20	1.35×10^4	7.20	6.88	1.65×10^4
31	7.12	2.17×10^4	2.04	7.08	2.82×10^4	9.44	7.4	3.77×10^4
41	7.76	4.09×10^4	2.48	6.84	4.96×10^4	10.52	6.64	6.81×10^4
51	7.60	6.15×10^4	2.80	7.72	8.58×10^4	11.72	7.00	1.06×10^5
61	7.04	8.11×10^4	2.36	7.36	1.12×10^5	12.28	6.64	1.67×10^5
71	7.76	1.20×10^5	2.72	7.08	1.53×10^5	11.35	6.88	2.36×10^5
81	7.32	1.47×10^5	2.36	6.72	1.84×10^5	11.72	7.24	2.78×10^5
91	7.52	1.90×10^5	2.62	6.64	2.37×10^5	12.12	6.40	3.30×10^5
16	6.76	5.78×10^3	1.84	7.16	7.85×10^3	6.72	7.48	9.33×10^3
32	7.24	2.35×10^4	2.08	6.84	2.93×10^4	10.52	6.32	3.93×10^4
64	7.08	8.96×10^4	2.20	6.64	1.13×10^5	12.00	6.64	1.64×10^5
128	6.96	3.47×10^5	2.44	6.64	4.54×10^5	14.80	6.48	7.27×10^5
256	7.84	1.55×10^6	3.20	7.92	2.20×10^6	17.48	6.80	3.41×10^6
512	8.09	6.38×10^6	3.77	7.00	8.51×10^6	20.31	6.09	1.42×10^7

we report in column 4 and column 7 the number of Newton and bisection steps to compute an initial point in the interval $(\lambda_n(T_n), \lambda_{n-1}(T_{n-1})$, respectively. For every method we report the average number of steps and the average number of `matlab` flops needed to compute the smallest eigenvalue with $\texttt{toll} = 10^{-14}$. The maximum Euclidian norm of the residual $r(\tilde{\lambda}, \tilde{x})$ encountered was $\approx 4.39 \times 10^{-13}$, and the average Euclidean norm was $\approx 1.04 \times 10^{-14}$. All the computations were performed in `matlab` on a Sun Workstation with machine precision $\epsilon \approx 2.2204 \times 10^{-16}$, and using `matlab`'s own internal flop counter. We can conclude from the table that all the methods exhibit $O(n^2)$ cost, and that Method N appears to be the most efficient.

7.7 Conclusions

In this chapter, a fast algorithm for computing the smallest eigenpair of symmetric positive definite Toeplitz matrices is described. Exploiting a recurrence relation for the principal minors of a symmetric positive definite Toeplitz matrix of order n, an $O(n^2)$ algorithm has been derived. Although its conceptual semplicity, the performances of the considered algorithm turn out to be comparable with those ones of the other proposed algorithms for solving such a problem.

Chapter 8

FAST AND STABLE ALGORITHMS FOR DIAGONAL PLUS SEMI–SEPARABLE MATRICES

8.1 Introduction

This chapter concerns fast and stable algorithms for solving basic linear algebra problems involving diagonal plus semi–separable matrices. In particular, fast and stable algorithms for solving linear systems involving DSS matrices, for reducing symmetric DSS matrices into symmetric tridiagonal ones and for reducing DSS matrices into bidiagonal ones by means of orthogonal transformations are considered.

DSS matrices appear when one discretizes the Green's function for the two point boundary value problem appropriately, and also from the discretization of eigenvalue problems for integral operators whose kernels can be written as the sum of a semi–separable part and a degenerate one [54], [63], [117]. Brownian matrices (see Ch.1, §1.6) and the inverse of tridiagonal matrices have this form too.

Fast algorithms for solving linear systems involving DSS matrices have been considered in [20], [40], [41]. The algorithm described in § 8.2, based on that one described in [20], is fast and numerically stable with linear complexity. However it is strictly sequential. In § 8.3 we describe a two–way algorithm for solving DSS linear systems with the same computational complexity of the algorithm proposed in [20]. Furthermore the proposed algorithm is better suited for a parallel implementation on a system of two processors with a speed–up equal to 1.84. Furthermore the basic idea of the algorithm can be easily generalized to compute the solution of linear systems of equations where the coefficient matrix is banded diagonal plus a semi–separable matrix of low rank (see [20] for details).

The standard direct procedure for computing the eigendecomposition of a symmetric DSS matrix of order N reduces first the matrix into tridiagonal form. This algorithm doesn't take into account the structure of the DSS matrix and requires $O(N^3)$ flops ([53, Chap. 8]). Although Lanczos' method can take advantage of the structure of the symmetric DSS matrix and can be used to find the eigendecomposition of such a matrix, the convergence rate is highly problem–dependent and can be very slow in many cases [99]. The fast algorithm for tridiagonalizing diagonal plus semi–separable matrices described in [21] and § 8.4 costs $O(N^2)$ flops.

In § 8.5 we present a two–way chasing algorithm for solving the same problem that halves the computational complexity of the latter algorithm. The two-chasing technique has been introduced in [136] and further investigated in [126] to halve the computational complexity of the algorithm for reducing arrowhead matrices into tridiagonal form proposed in [4].

In § 8.6 a fast algorithm is presented to reduce an unsymmetric diagonal plus a semi–separable matrix to a bidiagonal one. Since the latter algorithm does not accumulate the computed Givens rotations, its application is particularly recommended in the following cases:

1. Only the computation of the singular values is required.

2. One wants to solve $\min \|b - Bx\|_2^2 + \lambda \|x\|_2^2$, for many values of λ, where B is a rectangular matrix and b and x are vectors of appropriate dimensions [42].

§ 8.7 contains the conclusions.

8.2 Ch–Gu algorithm

A *diagonal plus semi–separable (DSS) matrix* is a matrix of the form

$$A = \begin{bmatrix} d_1 & u_2v_2 & u_2v_3 & \cdots & u_2v_{N-1} & u_2v_N \\ p_2q_2 & d_2 & u_3v_3 & u_3v_4 & \cdots & u_3v_N \\ p_3q_2 & p_3q_3 & \ddots & \cdots & \cdots & \cdots \\ \vdots & p_4q_3 & \vdots & \ddots & \cdots & \cdots \\ p_{N-1}q_2 & \vdots & \vdots & \vdots & d_{N-1} & u_Nv_N \\ p_Nq_2 & p_Nq_3 & \vdots & \vdots & p_Nq_N & d_N \end{bmatrix}. \tag{8.2.1}$$

Let $D = \mathrm{diag}(d_1, \ldots, d_N)$, $\mathbf{u} = [u_2, u_3, \ldots, u_N, 0]^T$, $\mathbf{v} = [0, v_2, v_3, \ldots, v_N]^T$, $\mathbf{p} = [0, p_2, p_3, \ldots, p_N]^T$, $\mathbf{q} = [q_2, q_3, \ldots, q_N, 0]^T$. A can be written as the sum of three matrices:

$$A = D + triu(\mathbf{uv}^T, 1) + tril(\mathbf{pq}^T, -1).$$

In this section we describe the basic (Ch–Gu) algorithm [20] for solving a linear system of equations,

$$Ax = b, \tag{8.2.2}$$

where A is a DSS matrix, $x = [x_1, \ldots, x_N]^T$,and $b = [b_1, \ldots, b_N]^T$. Let $A^{(0)} = A, x^{(0)} = x, b^{(0)} = b$, where the superscript between brackets denotes the iteration number, and $\tilde{G}_1$ be the *left* Givens rotation chosen in order to annihilate the entries in the first row,

$$\tilde{G}_1 \begin{bmatrix} u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{u}_3 \end{bmatrix}$$

and $G^{(1)} = \mathrm{diag}(\tilde{G}_1, I_{N-2})$, where I_k, $k \geq 1$ denotes the identity matrix of order k. Then

$$\tilde{A}^{(0)} = G^{(1)}A^{(0)} = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_{1,2} & & & \\ \tilde{a}_{2,1} & \tilde{d}_2 & \tilde{u}_3v_3 & \ldots & \tilde{u}_3v_N \\ p_3q_2 & p_3q_3 & \ddots & \ldots & \vdots \\ \vdots & \vdots & \vdots & d_{N-1} & u_Nv_N \\ p_Nq_2 & p_Nq_3 & \ldots & p_Nq_N & d_N \end{bmatrix} \text{ and } \tilde{b}^{(0)} = G^{(1)}b^{(0)}.$$

Let $\hat{H}_1$ be the *right* Givens rotation chosen in order to annihilate $\tilde{a}_{1,2}$ in $\tilde{A}^{(0)}$,

$$\hat{H}_1 \begin{bmatrix} \tilde{d}_1 \\ \tilde{a}_{1,2} \end{bmatrix} = \begin{bmatrix} \hat{d}_1 \\ 0 \end{bmatrix}$$

and $H^{(1)} = \operatorname{diag}(\hat{H}_1, I_{N-2})$. Then

$$\hat{A}^{(0)} = \tilde{A}^{(0)} H^{(1)^T} = \begin{bmatrix} \hat{d}_1 & & & & \\ \hat{\alpha}_{2,1} & \hat{d}_2 & \tilde{u}_3 v_3 & \dots & \tilde{u}_3 v_N \\ p_3 \hat{q}_2 & p_3 \hat{q}_3 & \ddots & \dots & \vdots \\ \vdots & \vdots & \vdots & d_{N-1} & u_N v_N \\ p_N \hat{q}_2 & p_N \hat{q}_3 & \dots & p_N q_N & d_N \end{bmatrix}$$

and

$$\hat{x}^{(0)} = H^{(1)} x^{(0)} = \begin{bmatrix} \hat{x}_1^{(0)} \\ \hat{x}_2^{(0)} \\ x_3^{(0)} \\ \vdots \\ x_N^{(0)} \end{bmatrix}.$$

Now $\hat{x}_1^{(0)}$ can be directly computed,

$$\hat{x}_1^{(0)} = \frac{\tilde{b}_1^{(0)}}{\hat{d}_1}.$$

Let

$$\begin{aligned} A^{(1)} &= \hat{A}^{(0)}(2:N, 2:N), \\ x^{(1)} &= \hat{x}^{(0)}(2:N), \\ b^{(1)} &= \tilde{b}^{(0)}(2:N) - \hat{x}_1^{(0)} \hat{A}^{(0)}(2:N, 1). \end{aligned}$$

The problem is reduced to solving the $(N-1)\times(N-1)$ linear system $A^{(1)}x^{(1)} = b^{(1)}$, where $A^{(1)}$ is a diagonal plus a semi-separable matrix of order $N-1$. Applying the described technique recursively $N-2$ times, the problem is reduced to solving a 2×2 linear system

$$A^{(N-2)} x^{(N-2)} = b^{(N-2)}.$$

Define

$$x^{(N-2)} = [x_1^{(N-2)}, x_2^{(N-2)}]^T.$$

Let

$$\begin{aligned} y &= [\hat{x}_1^{(0)}, \hat{x}_1^{(1)}, \dots, \hat{x}_1^{(N-3)}, x_1^{(N-2)}, x_2^{(N-2)}]^T, \\ \tilde{H}^{(i)} &= \operatorname{diag}(I_{i-1}, H^{(i)}), \quad i = 1, \dots, N-2, \end{aligned}$$

the solution of (8.2.2) is given by

$$x = \tilde{H}^{(1)} \dots \tilde{H}^{(N-3)} \tilde{H}^{(N-2)} y.$$

The corresponding `matlab`–like function `semisep` can be found in Appendix C.1.

We use the definition of flop given in [53, p. 18], *"A flop is a floating point operation"*. The square root is considered a flop, too.

Hence the total number of flops of the Ch–Gu algorithm is $59N$.

8.3 Two–way algorithm

The algorithm we are going to describe is a two-way variant of that one described in [20]. Consider the linear system of equations (8.2.2). Let $\tilde{N} = \lfloor N/2 \rfloor - 1$. Let $A^{(0)} = A, x^{(0)} = x, b^{(0)} = b$, where the superscript between brackets denotes the iteration number, and $\tilde{G}_1$, $\tilde{G}_N$ be *left* Givens rotations chosen in order to annihilate the entries in the first row and the last row of (8.2.1), respectively,

$$\tilde{G}_1 \begin{bmatrix} u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{u}_3 \end{bmatrix} \quad \tilde{G}_N \begin{bmatrix} p_{N-1} \\ p_N \end{bmatrix} = \begin{bmatrix} \tilde{p}_{N-1} \\ 0 \end{bmatrix}$$

and $G^{(1)} = \mathrm{diag}(\tilde{G}_1, I_{N-4}, \tilde{G}_N)$. Then

$$\tilde{A}^{(0)} = G^{(1)} A^{(0)} = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_{1,2} & & & & \\ \tilde{a}_{2,1} & \tilde{d}_2 & \tilde{u}_3 v_3 & \cdots & \tilde{u}_3 v_{N-1} & \tilde{u}_3 v_N \\ p_3 q_2 & p_3 q_3 & \ddots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & u_{N-1} v_{N-1} & u_{N-1} v_N \\ \tilde{p}_{N-1} q_2 & \tilde{p}_{N-1} q_3 & \vdots & \tilde{p}_{N-1} q_{N-1} & \tilde{d}_{N-1} & \tilde{a}_{N-1,N} \\ & & & & \tilde{a}_{N,N-1} & \tilde{d}_N \end{bmatrix}$$

and $\tilde{b}^{(0)} = G^{(1)} b^{(0)}$. Let $\hat{H}_1$ and $\hat{H}_N$ be *right* Givens rotations chosen in order to annihilate $\tilde{a}_{1,2}$ and $\tilde{a}_{N,N-1}$ in $\tilde{A}^{(0)}$, respectively,

$$\hat{H}_1 \begin{bmatrix} \tilde{d}_1 \\ \tilde{a}_{1,2} \end{bmatrix} = \begin{bmatrix} \hat{d}_1 \\ 0 \end{bmatrix}, \quad \hat{H}_N \begin{bmatrix} \tilde{a}_{N,N-1} \\ \tilde{d}_N \end{bmatrix} = \begin{bmatrix} 0 \\ \hat{d}_N \end{bmatrix}$$

and $H^{(1)} = \mathrm{diag}(\hat{H}_1, I_{N-4}, \hat{H}_N)$. Then

$$\hat{A}^{(0)} = \tilde{A}^{(0)} {H^{(1)}}^T = \begin{bmatrix} \hat{d}_1 & & & & & \\ \hat{a}_{2,1} & \hat{d}_2 & \tilde{u}_3 v_3 & \cdots & \tilde{u}_3 \hat{v}_{N-1} & \tilde{u}_3 \hat{v}_N \\ p_3 \hat{q}_2 & p_3 \hat{q}_3 & \ddots & \cdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & u_{N-1} \hat{v}_{N-1} & u_{N-1} \hat{v}_N \\ \tilde{p}_{N-1} \hat{q}_2 & \tilde{p}_{N-1} \hat{q}_3 & \vdots & \tilde{p}_{N-1} q_{N-1} & \hat{d}_{N-1} & \hat{a}_{N-1,N} \\ & & & & & \hat{d}_N \end{bmatrix}$$

and

$$\hat{x}^{(0)} = H^{(1)}x^{(0)} = \begin{bmatrix} \hat{x}_1^{(0)} \\ \hat{x}_2^{(0)} \\ x_3^{(0)} \\ \vdots \\ \hat{x}_{N-1}^{(0)} \\ \hat{x}_N^{(0)} \end{bmatrix}.$$

Now $\hat{x}_1^{(0)}$ and $\hat{x}_N^{(0)}$ can be directly computed,

$$\hat{x}_1^{(0)} = \frac{\tilde{b}_1^{(0)}}{\hat{d}_1}, \quad \hat{x}_N^{(0)} = \frac{\tilde{b}_N^{(0)}}{\hat{d}_N}.$$

Let

$$\begin{aligned} A^{(1)} &= \hat{A}^{(0)}(2:N-1, 2:N-1), \\ x^{(1)} &= \hat{x}^{(0)}(2:N-1), \\ b^{(1)} &= \tilde{b}^{(0)}(2:N-1) - \hat{x}_1^{(0)}\hat{A}^{(0)}(2:N-1,1) - \hat{x}_N^{(0)}\hat{A}^{(0)}(2:N-1,N). \end{aligned}$$

The new problem to solve is the linear system $A^{(1)}x^{(1)} = b^{(1)}$, where $A^{(1)}$ is a diagonal plus a semi-separable matrix of order $N-2$. Applying the described technique recursively $\tilde{N}$ times, the problem is reduced to solving either a 2×2 (if N is even) or 3×3 (if N is odd) linear system

$$A^{(\tilde{N})}x^{(\tilde{N})} = b^{(\tilde{N})}. \tag{8.3.3}$$

More precisely, the coefficient matrix of the reduced linear system to be solved is

$$\begin{bmatrix} \hat{d}_{\tilde{N}+1} & \tilde{u}_{\tilde{N}+2}v_{\tilde{N}+2} & \tilde{u}_{\tilde{N}+2}\widehat{v}_{N-\tilde{N}} \\ p_{\tilde{N}+2}\hat{q}_{\tilde{N}+2} & \hat{d}_{\tilde{N}+2} & u_{N-\tilde{N}}\widehat{v}_{N-\tilde{N}} \\ \tilde{p}_{N-\tilde{N}}\hat{q}_{\tilde{N}+2} & \tilde{p}_{N-\tilde{N}}q_{\tilde{N}+2} & \hat{d}_{N-\tilde{N}} \end{bmatrix}$$

if N is odd,

$$\begin{bmatrix} \hat{d}_{\tilde{N}+1} & \tilde{u}_{N-\tilde{N}}\widehat{v}_{N-\tilde{N}} \\ \tilde{p}_{N-\tilde{N}}\hat{q}_{N-\tilde{N}} & \hat{d}_{N-\tilde{N}} \end{bmatrix}$$

if N is even. Define $x^{(\tilde{N})} = [x_1^{(\tilde{N})}, \ldots, x_{N-2\tilde{N}}^{(\tilde{N})}]^T$.

Let

$$\begin{aligned} y &= \left[\hat{x}_1^{(0)}, \hat{x}_1^{(1)}, \ldots, \hat{x}_1^{(\tilde{N}-1)}, x_1^{(\tilde{N})}, \ldots, x_{N-2\tilde{N}}^{(\tilde{N})}, \hat{x}_{N-2\tilde{N}+2}^{(\tilde{N}-1)}, \ldots, \hat{x}_{N-2}^{(1)}, \hat{x}_N^{(0)}\right]^T, \\ \tilde{H}^{(i)} &= \operatorname{diag}(I_{i-1}, H^{(i)}, I_{i-1}), \quad i = 1, \ldots, \tilde{N}, \end{aligned}$$

the solution of (8.2.2) is given by

$$x = \tilde{H}^{(1)^T} \cdots \tilde{H}^{(\tilde{N}-1)^T} \tilde{H}^{(\tilde{N})^T} y.$$

The algorithm is summarized in the `matlab`–like function `semisep2` that can be found in Appendix C.2.

The total number of flops is $64N$ if the algorithm is implemented on a single processor. The total number of flops is halved if the algorithm is implemented on two processors. The number of flops required by the one–way algorithm presented in [20] for solving (8.2.2) is $59N$. Hence the speed-up[1] of the new algorithm is equal to 1.84. However, an analysis of a parallel implementation of the algorithm should take into account the interprocessor communications. In tightly coupled 2-processor systems, the data exchange overhead is comparable with a few, say K_f, floating point operations. Since interprocessor comunication is needed in each of the first $\tilde{N} \approx N/2$ steps of the algorithm, each processor requires an execution time equivalent to the execution of $(32 + K_f/2)N$ flops. In this case, the speed-up is given by the ratio $59/(32 + K_f/2)$.

8.4 One-way chasing algorithm

In this section we consider the method described in [21]. It requires $N - 2$ steps to reduce a diagonal plus symmetric semi–separable matrix (8.2.1), with $\mathbf{p} = \mathbf{u}$ and $\mathbf{q} = \mathbf{p}$, into a similar tridiagonal one. In the following we represent with $\tilde{*}$ the elements of the original matrix that are modified by Givens transformations.

- step 1. Let $\tilde{G}_1$ be the Givens rotation such that

$$\tilde{G}_1 \begin{bmatrix} u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{u}_3 \end{bmatrix}$$

  and $G_1 = \mathrm{diag}(\tilde{G}_1, I_{N-2})$, where I_{N-2} is the identity matrix of order $N - 2$. Then

$$\tilde{A}_1 = G_1 A_0 G_1^T = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & 0 & \cdots & 0 & 0 \\ \tilde{a}_1 & \tilde{d}_2 & \tilde{u}_3 v_3 & \tilde{u}_3 v_4 & \cdots & \tilde{u}_3 v_N \\ 0 & \tilde{u}_3 v_3 & \ddots & \cdots & \cdots & \cdots \\ \vdots & \tilde{u}_3 v_4 & \vdots & \ddots & \cdots & \cdots \\ 0 & \vdots & \vdots & \vdots & d_{N-1} & u_N v_N \\ 0 & \tilde{u}_3 v_N & \vdots & \vdots & u_N v_N & d_N \end{bmatrix}.$$

[1] Slightly modifying the definition in [53], we say that a parallel algorithm for a particular problem achieves speed–up S if $S = F_{seq}/F_{par}$, where F_{par} is the number of flops required for the execution of the parallel algorithm on each processor and F_{seq} is the number of flops required by one processor when the best uniprocessor procedure is used.

- step $i.1$, $i \leq N-2$.

$$A_{i-1} = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & & & & & & \\ \tilde{a}_1 & \ddots & \ddots & & & & & \\ & \ddots & \tilde{d}_{i-1} & \tilde{a}_{i-1} & & & & \\ & & \tilde{a}_{i-1} & \tilde{d}_i & \tilde{u}_{i+1}v_{i+1} & \tilde{u}_{i+1}v_{i+2} & \cdots & \tilde{u}_{i+1}v_N \\ & & & \tilde{u}_{i+1}v_{i+1} & d_{i+1} & u_{i+2}v_{i+2} & \cdots & u_{i+2}v_N \\ & & & \tilde{u}_{i+1}v_{i+2} & u_{i+2}v_{i+2} & \ddots & \cdots & \cdots \\ & & & \vdots & \vdots & \vdots & \ddots & u_Nv_N \\ & & & \tilde{u}_{i+1}v_N & u_{i+2}v_N & \vdots & u_Nv_N & d_N \end{bmatrix}.$$

Let $\tilde{G}_i$ be the Givens rotation such that

$$\tilde{G}_i \begin{bmatrix} \tilde{u}_{i+1} \\ u_{i+2} \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{u}_{i+2} \end{bmatrix}$$

and $G_i = \mathrm{diag}(I_{i-1}, \tilde{G}_i, I_{N-i-1})$. Then

$$\begin{aligned} \tilde{A}_i &= G_i A_{i-1} G_i^T \\ &= \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & & & & & & \\ \tilde{a}_1 & \ddots & \ddots & & & & & \\ & \ddots & \tilde{d}_{i-1} & \tilde{a}_{i-1} & e & & & \\ & & \tilde{a}_{i-1} & \tilde{d}_i & \tilde{a}_i & & & \\ & & e & \tilde{a}_i & d_{i+1} & \tilde{u}_{i+2}v_{i+2} & \cdots & \tilde{u}_{i+2}v_N \\ & & & & \tilde{u}_{i+2}v_{i+2} & \ddots & \cdots & \cdots \\ & & & & \vdots & \vdots & \ddots & u_Nv_N \\ & & & & \tilde{u}_{i+2}v_N & \vdots & u_Nv_N & d_N \end{bmatrix}. \end{aligned}$$

- step $i.2$: elimination of the bulge e. This can be accomplished by standard chasing algorithms [114]. For instance, suppose $i = 4$,

$$\tilde{A}_4 = \begin{bmatrix} \times & \times & & & & & \\ \times & \times & \times & & & & \\ & \times & \times & \times & \otimes & & \\ & & \times & \times & \times & & \\ & & \otimes & \times & \times & \cdots & \times \\ & & & & \vdots & \ddots & \vdots \\ & & & & \times & \cdots & \times \end{bmatrix}.$$

where the symbol $\otimes$ represents the entry to be annihilated at the current step. The bulge can be eliminated using a sequence of 3 suitable Givens rotations H_j, $j = 3, \ldots, 1$ (see Fig. 8.1)

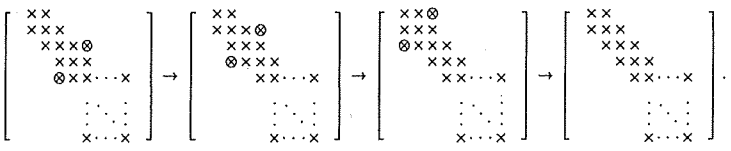


Figure 8.1. Elimination of the bulge. The bulge is eliminated using a sequence of suitable Givens rotations H_j. At each step, the bulge, denoted by $\otimes$, is moved from the entries $(j+2, j)$ and $(j, j+2)$ to the entries $(j+1, j-1)$ and $(j-1, j+1)$.

The i–th step requires i Givens transformations. We observe that the entries of the vector $\mathbf{v}$ are not modified. At the i–th step i Givens similarity transformations are performed. Each of these requires 26 flops, hence the total number of flops is $13N^2$.

The algorithm is summarized in the `matlab`–like function `trid` and can be found in Appendix C.3.

8.5 Two-way chasing algorithm

At first we consider the case N even. The case N odd is handled at the end of the section. Let $\tilde{N} = N/2$. The two-way chasing algorithm we describe consists of $\tilde{N} - 1$ steps:

- step 1.

  Let $\tilde{G}_1$ and $\tilde{G}_N$ be Givens rotations such that

$$\tilde{G}_1 \begin{bmatrix} u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{u}_3 \end{bmatrix}, \quad \tilde{G}_N \begin{bmatrix} v_{N-1} \\ v_N \end{bmatrix} = \begin{bmatrix} \tilde{v}_{N-1} \\ 0 \end{bmatrix},$$

respectively, and $G_1 = \mathrm{diag}(\tilde{G}_1, I_{N-4}, \tilde{G}_N)$. Then

$$G_1 A_0 G_1^T = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & & & & \\ \tilde{a}_1 & \tilde{d}_2 & \tilde{u}_3\tilde{v}_3 & \cdots & \tilde{u}_3\tilde{v}_{N-1} & \\ & \tilde{u}_3\tilde{v}_3 & \ddots & \cdots & \cdots & \\ & \vdots & \vdots & \ddots & u_{N-1}\tilde{v}_{N-1} & \\ & \tilde{u}_3\tilde{v}_{N-1} & \vdots & u_{N-1}\tilde{v}_{N-1} & \tilde{d}_{N-1} & \tilde{a}_{N-1} \\ & & & & \tilde{a}_{N-1} & \tilde{d}_N \end{bmatrix}.$$

- step $i.1$, $i \leq N-2$. Let $M = N-i+1$, $L = N-i$, $k = i+1$ and $l = i+2$.

$$A_{i-1} = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & & & & & & & \\ \tilde{a}_1 & \ddots & \ddots & & & & & & \\ & \ddots & \tilde{d}_{i-1} & \tilde{a}_{i-1} & & & & & \\ & & \tilde{a}_{i-1} & \tilde{d}_i & \tilde{u}_k v_k & \cdots & \tilde{u}_k\tilde{v}_M & & \\ & & & \tilde{u}_k v_k & d_k & \cdots & \cdots & & \\ & & & \vdots & \vdots & \ddots & u_M\tilde{v}_M & & \\ & & & \tilde{u}_k\tilde{v}_M & \vdots & u_M\tilde{v}_M & \tilde{d}_M & \tilde{a}_M & \\ & & & & & & \tilde{a}_M & \ddots & \ddots \\ & & & & & & & \ddots & \ddots & \tilde{a}_{N-1} \\ & & & & & & & & \tilde{a}_{N-1} & \tilde{d}_N \end{bmatrix}.$$

Let $\tilde{G}_i$ and $\tilde{G}_{N-i+1}$ be Givens rotations such that

$$\tilde{G}_i \begin{bmatrix} u_{i+1} \\ \tilde{u}_{i+2} \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{u}_{i+2} \end{bmatrix}, \quad \tilde{G}_{N-i+1} \begin{bmatrix} v_{N-i} \\ \tilde{v}_{N-i+1} \end{bmatrix} = \begin{bmatrix} \tilde{v}_{N-i} \\ 0 \end{bmatrix},$$

respectively, and $G_i = \mathrm{diag}(I_{i-1}, \tilde{G}_i, I_{N-2i-2}, \tilde{G}_{N-i+1}, I_{i-1})$. Then

$$\tilde{A}_i = G_i A_{i-1} G_i^T$$

$$= \begin{bmatrix}
\tilde{d}_1 & \tilde{a}_1 & & & & & & & & & \\
\tilde{a}_1 & \ddots & \ddots & & & & & & & & \\
 & \ddots & \tilde{d}_{i-1}\tilde{a}_{i-1} & e & & & & & & & \\
 & & \tilde{a}_{i-1} & \tilde{d}_i & \tilde{a}_i & & & & & & \\
 & & e & \tilde{a}_i & \tilde{d}_k & \tilde{u}_l v_l & \cdots & \tilde{u}_l\tilde{v}_L & & & \\
 & & & & \tilde{u}_l v_l & d_l & \cdots & \vdots & & & \\
 & & & & \vdots & \vdots & \ddots & u_L\tilde{v}_L & & & \\
 & & & & \tilde{u}_l\tilde{v}_L & \cdots & \tilde{u}_L\tilde{v}_L & \tilde{d}_L & \tilde{a}_L & f & \\
 & & & & & & & \tilde{a}_L & \tilde{d}_M & \tilde{a}_M & \\
 & & & & & & & f & \tilde{a}_M & \ddots & \ddots \\
 & & & & & & & & \ddots & \ddots & \tilde{a}_{N-1} \\
 & & & & & & & & & \tilde{a}_{N-1} & \tilde{d}_N
\end{bmatrix}.$$

- step $i.2$: elimination of the bulges e and f. This can be accomplished as previously.

For instance, suppose $i = 4$,

$$\tilde{A}_4 = \begin{bmatrix}
\times & \times & & & & & & & & \\
\times & \times & \times & & & & & & & \\
 & \times & \times & \times & \otimes & & & & & \\
 & & \times & \times & \times & & & & & \\
 & & \otimes & \times & \times & \cdots & \times & & & \\
 & & & & \vdots & \ddots & \vdots & & & \\
 & & & & \times & \cdots & \times & \times & \otimes & \\
 & & & & & & \times & \times & \times & \\
 & & & & & & \otimes & \times & \times & \times \\
 & & & & & & & & \times & \times \\
 & & & & & & & & & \times
\end{bmatrix},$$

The bulges can be eliminated using two sequences of 3 suitable Givens rotations H_j and K_{N-j+1}, $j = 3, \ldots, 1$ (see Fig. 8.2).The matrices H_j move the bulge from the entries $(j+2, j)$ and $(j, j+2)$ to the entries $(j+1, j-1)$ and $(j-1, j+1)$. The matrices K_{N-j+1} move the bulge from the entries $(N-j-1, N-j+1)$ and $(N-j+1, N-j-1)$ to the entries $(N-j-2, N-j)$ and $(N-j, N-j-2)$,

The i–th step requires i Givens similarity transformations. The algorithm is summarized in the `matlab`–like function `trid2` in Appendix C.4.

At the i–th step i Givens similarity transformations are performed, requiring $52i$ flops, hence the total number of flops is $26\tilde{N}^2 = 6.5N^2$.

In case N odd, we proceed in the following way. Let $\tilde{N} = \lfloor N/2 \rfloor$. Applying a slight modification of procedure `trid2`, the resulting similar matrix is

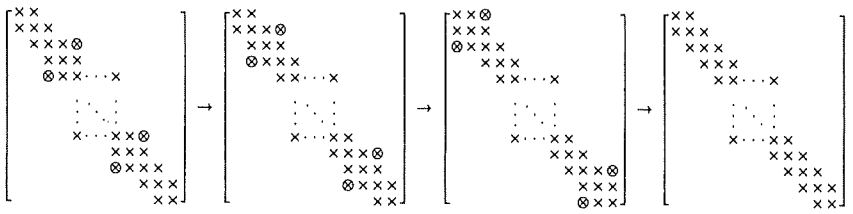


Figure 8.2. Elimination of the bulge. At each step, the bulges are eliminated using two sequences of suitable Givens rotations H_j and K_{N-j+1}. The matrices H_j move the bulge from the entries $(j+2, j)$ and $(j, j+2)$ to the entries $(j+1, j-1)$ and $(j-1, j+1)$. The matrices K_{N-j+1} move the bulge from the entries $(N-j-1, N-j+1)$ and $(N-j+1, N-j-1)$ to the entries $(N-j-2, N-j)$ and $(N-j, N-j-2)$,

$$\tilde{A}_{\tilde{N}-2} = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & & & & & & & & \\ \tilde{a}_1 & \ddots & \ddots & & & & & & & \\ & \ddots & \tilde{d}_{\tilde{N}-1} & \tilde{a}_{\tilde{N}-1} & & & & & & \\ & & \tilde{a}_{\tilde{N}-1} & \tilde{d}_{\tilde{N}} & \tilde{u}_{\tilde{N}+1}v_{\tilde{N}+1} & \tilde{u}_{\tilde{N}+1}\tilde{v}_{\tilde{N}+2} & & & & \\ & & & \tilde{u}_{\tilde{N}+1}v_{\tilde{N}+1} & d_{\tilde{N}+1} & u_{\tilde{N}+2}\tilde{v}_{\tilde{N}+2} & & & & \\ & & & \tilde{u}_{\tilde{N}+1}\tilde{v}_{\tilde{N}+2} & u_{\tilde{N}+2}\tilde{v}_{\tilde{N}+2} & \tilde{d}_{\tilde{N}+2} & \tilde{a}_{\tilde{N}+2} & & & \\ & & & & & \tilde{a}_{\tilde{N}+2} & \tilde{d}_{\tilde{N}+3} & \ddots & & \\ & & & & & & \ddots & \ddots & \tilde{a}_{N-1} & \\ & & & & & & & \tilde{a}_{N-1} & \tilde{d}_N & \end{bmatrix}.$$

Let $\tilde{G}_{\tilde{N}-1}$ be the Givens rotation such that

$$\tilde{G}_{\tilde{N}-1} \begin{bmatrix} u_{\tilde{N}+1} \\ u_{\tilde{N}+2} \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{u}_{\tilde{N}+2} \end{bmatrix}$$

and $G_{\tilde{N}-1} = \mathrm{diag}(I_{i-1}, \tilde{G}_{\tilde{N}-1}, I_{\tilde{N}-i-1})$. Then $\tilde{A}_{\tilde{N}-1} = G_{\tilde{N}-1} A_{\tilde{N}-2} G_{\tilde{N}-1}^T$,

$$\tilde{A}_{\tilde{N}-1} = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & & & & & & & \\ \tilde{a}_1 & \ddots & \ddots & & & & & & \\ & \ddots & \tilde{d}_{\tilde{N}-1} & \tilde{a}_{\tilde{N}-1} & e & & & & \\ & & \tilde{a}_{\tilde{N}-1} & \tilde{d}_{\tilde{N}} & \tilde{a}_{\tilde{N}} & & & & \\ & & e & \tilde{a}_{\tilde{N}} & \tilde{d}_{\tilde{N}+1} & \tilde{a}_{\tilde{N}+1} & & & \\ & & & & \tilde{a}_{\tilde{N}+1} & \tilde{d}_{\tilde{N}+2} & \tilde{a}_{\tilde{N}+2} & & \\ & & & & & \tilde{a}_{\tilde{N}+2} & \tilde{d}_{\tilde{N}+3} & \ddots & \\ & & & & & & \ddots & \ddots & \tilde{a}_{N-1} \\ & & & & & & & \tilde{a}_{N-1} & \tilde{d}_N \end{bmatrix}.$$

The bulge e can be eliminated as previously, using $\tilde{N} - 1$ Givens similarity transformations.

Remark 8.5.1: *We observe that the reduction of the original matrix in the top left corner modifies only the vector* $\mathbf{u}$, *the reduction of the matrix (8.2.1) in the bottom right corner modifies only the vector* $\mathbf{v}$. *These reductions are completely independent and can be efficiently performed in a separate way by two parallel processors.*

Remark 8.5.2: *In [22] a one-way chasing algorithm is presented which reduces a symmetric banded diagonal matrix of bandwidth b plus a symmetric semi–separable matrix of rank* $r \geq 1$ *into a symmetric tridiagonal one. The algorithm consists of two steps. In the first step the initial matrix is reduced to a symmetric banded matrix of bandwidth* $b + r$, *and in the second step the technique described in [114] is used to reduce the banded matrix to tridiagonal form. The number of computations of the first step can be halved, considering an appropriate modification of the described two–way chasing algorithm.*

8.6 Bidiagonalizing diagonal plus unsymmetric semi–separable matrices

In this section we consider an algorithm to reduce the diagonal plus an unsymmetric semi–separable matrix (8.2.1) into bidiagonal form by means of orthogonal transformations. The algorithm is divided in three parts:

1. transformation of (8.2.1) into an unsymmetric tridiagonal matrix;
2. transformation of the unsymmetric tridiagonal matrix into an upper tridiagonal one;
3. transformation of the upper tridiagonal matrix into an upper bidiagonal one.

8.6.1 Transformation of a diagonal plus an unsymmetric semi-separable matrix into an unsymmetric tridiagonal one

This part of the algorithm is quite similar to that considered before to reduce a diagonal plus a symmetric semi-separable matrix into a symmetric tridiagonal one.

Let $\tilde{G}_1$ and $\tilde{H}_1$ be Givens rotations such that

$$\tilde{G}_1 \begin{bmatrix} u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{u}_3 \end{bmatrix}, \quad \tilde{H}_1 \begin{bmatrix} q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{q}_3 \end{bmatrix}$$

and $G_1 = \mathrm{diag}(\tilde{G}_1, I_{N-2})$, $H_1 = \mathrm{diag}(\tilde{H}_1, I_{N-2})$. Then

$$G_1 A_0 H_1^T = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & 0 & \cdots & 0 & 0 \\ \tilde{b}_1 & \tilde{d}_2 & \tilde{u}_3 v_3 & \tilde{u}_3 v_4 & \cdots & \tilde{u}_3 v_N \\ 0 & \tilde{q}_3 p_3 & \ddots & \cdots & \cdots & \cdots \\ \vdots & \tilde{q}_3 p_4 & \vdots & \ddots & \cdots & \cdots \\ 0 & \vdots & \vdots & \vdots & d_{N-1} & u_N v_N \\ 0 & \tilde{q}_3 p_N & \vdots & \vdots & q_N p_N & d_N \end{bmatrix}.$$

step $i.1$, $i \leq N-2$.

$$A_{i-1} = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & & & & & & \\ \tilde{b}_1 & \ddots & \ddots & & & & & \\ & \ddots & \tilde{d}_{i-1} & \tilde{a}_{i-1} & & & & \\ & & \tilde{a}_{i-1} & \tilde{d}_i & \tilde{u}_{i+1} v_{i+1} & \tilde{u}_{i+1} v_{i+2} & \cdots & \tilde{u}_{i+1} v_N \\ & & & \tilde{q}_{i+1} p_{i+1} & \tilde{d}_{i+1} & u_{i+2} v_{i+2} & \cdots & u_{i+2} v_N \\ & & & \tilde{q}_{i+1} p_{i+2} & q_{i+2} p_{i+2} & \ddots & \cdots & \cdots \\ & & & \vdots & \vdots & \vdots & \ddots & u_N v_N \\ & & & \tilde{q}_{i+1} p_N & q_{i+2} p_N & \vdots & q_N p_N & d_N \end{bmatrix}.$$

Let $\tilde{G}_i$ and $\tilde{H}_i$ be Givens rotations such that

$$\tilde{G}_i \begin{bmatrix} \tilde{u}_{i+1} \\ u_{i+2} \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{u}_{i+2} \end{bmatrix}, \quad \tilde{H}_i \begin{bmatrix} \tilde{q}_{i+1} \\ q_{i+2} \end{bmatrix} = \begin{bmatrix} 0 \\ \tilde{q}_{i+2} \end{bmatrix}$$

and $G_i = \mathrm{diag}(I_{i-1}, \tilde{G}_i, I_{N-i-1})$, $H_i = \mathrm{diag}(I_{i-1}, \tilde{H}_i, I_{N-i-1})$. Then

$$\tilde{A}_i = G_i A_{i-1} H_i^T = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & & & & & & \\ \tilde{b}_1 & \ddots & \ddots & & & & & \\ & \ddots & \tilde{d}_{i-1} & \tilde{a}_{i-1} & e & & & \\ & & \tilde{b}_{i-1} & \tilde{d}_i & \tilde{a}_i & & & \\ & & f & \tilde{a}_i & d_{i+1} & \tilde{u}_{i+2}v_{i+2} & \cdots & \tilde{u}_{i+2}v_N \\ & & & & \tilde{q}_{i+2}p_{i+2} & \ddots & \cdots & \cdots \\ & & & & \vdots & \vdots & \ddots & u_N v_N \\ & & & & \tilde{q}_{i+2}p_N & \vdots & q_N p_N & d_N \end{bmatrix}.$$

step $i.2$: elimination of the bulges e and f. The bulges e and f can be eliminated considering $2i-2$ Givens matrices, $\hat{H}_j$ and $\hat{K}_j$, $j = i-1, \ldots, 1$. Let $\hat{A}_i = \tilde{A}_i$. Then, for $j = i-1, \ldots, 1$, $\hat{A}_j = \hat{H}_j \hat{A}_{j+1} \hat{K}_j^T$ where the matrices $\hat{H}_j$ and $\hat{K}_j$ are chosen in order to move the bulges e and f from the entry $(j, j+2)$ and $(j+2, j)$ of $\hat{A}_j$ to the entry $(j-1, j+1)$ and $(j+1, j-1)$ of $\hat{A}_{j-1}$. Finally, we pose $A_i = \hat{A}_1$.

The i–th step requires $2i$ Givens transformations. We observe that the entries of the vector $\mathbf{v}$ and $\mathbf{p}$ are not modified.

At the i–th step the matrix A_{i-1} is multiplied by i Givens matrices on the left and i Givens matrices on the right. The i–th step requires $48i$ flops, hence the total number of flops is $24N^2$.

The `matlab`–like function `tridns`, corresponding to the latter algorithm, can be found in Appendix C.5.

Considering the two–way chasing described in §3, it is straightforward to derive a two–way chasing algorithm reducing (8.2.1) to an unsymmetric tridiagonal matrix, halving the number of computations.

8.6.2 Transformation of the unsymmetric tridagonal matrix into an upper tridiagonal one

This step can be accomplished using $N-1$ Givens rotations.

- Iteration i, $1 \leq i \leq N-1$.

  Let

$$A_{i-1} = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & \tilde{b}_1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \tilde{d}_{i-1} & \tilde{a}_{i-1} & \tilde{b}_{i-1} & & \\ & & & d_i & a_i & & \\ & & & b_i & d_{i+1} & \ddots & \\ & & & & \ddots & \ddots & a_{N-1} \\ & & & & & b_{N-1} & d_N \end{bmatrix},$$

$\tilde{G}_i$ be the Givens rotation such that

$$\tilde{G}_i \begin{bmatrix} d_i \\ b_i \end{bmatrix} = \begin{bmatrix} \tilde{d}_i \\ 0 \end{bmatrix},$$

and $G_i = \mathrm{diag}(I_{i-1}, \tilde{G}_i, I_{N-i-1})$. Then

$$A_i = G_i A_{i-1} = \begin{bmatrix} \tilde{d}_1 & \tilde{a}_1 & \tilde{b}_1 & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & \tilde{d}_i & \tilde{a}_i & \tilde{b}_i & & & \\ & & & d_{i+1} & a_{i+1} & & & \\ & & & b_{i+1} & d_{i+2} & \ddots & & \\ & & & & \ddots & \ddots & a_{N-1} & \\ & & & & & b_{N-1} & d_N \end{bmatrix}.$$

The i–th step, $i = 1, \ldots, N-1$, requires 14 flops. Hence the total number of flops is $14N$. The previous algorithm is summarized in the `matlab`–like function `triup` in Appendix C.6.

8.6.3 Transformation of the upper tridiagonal matrix into an upper bidiagonal one

The bidiagonalization of the upper tridiagonal matrix is accomplished in $N-2$ iterations.

- iteration i, step 1, $1 \leq i \leq N-2$.

  Let

$$A_{i-1} = \begin{bmatrix} d_1 & a_1 & & & & & & \\ & \ddots & \ddots & & & & & \\ & & d_{i-1} & a_{i-1} & & & & \\ & & & d_i & a_i & b_i & & \\ & & & & d_{i+1} & \ddots & \ddots & \\ & & & & & \ddots & a_{N-2} & b_{N-1} \\ & & & & & & d_{N-1} & a_{N-1} \\ & & & & & & & d_N \end{bmatrix},$$

  $\tilde{G}_i$ be the Givens rotation such that

$$\tilde{G}_i \begin{bmatrix} a_i \\ b_i \end{bmatrix} = \begin{bmatrix} \tilde{a}_i \\ 0 \end{bmatrix},$$

and $G_i = \mathrm{diag}(I_{i-1}, \tilde{G}_i, I_{N-i-1})$. Then

$$\tilde{A}_i = A_{i-1}G_i^T = \begin{bmatrix} d_1 & a_1 & & & & & & & \\ & \ddots & \ddots & & & & & & \\ & & d_{i-1} & a_{i-1} & & & & & \\ & & & d_i & \tilde{a}_i & & & & \\ & & & & d_{i+1} & a_{i+1} & b_{i+1} & & \\ & & & & e & d_{i+2} & \ddots & \ddots & \\ & & & & & & \ddots & a_{N-1} & \\ & & & & & & & d_N & \end{bmatrix}.$$

iteration i, step 2. Chase the bulge e in the matrix $\tilde{A}_i$. This can be accomplished using the technique considered in [126]. For instance, if $N = 5$, the procedure is summarized in Fig. 8.3.

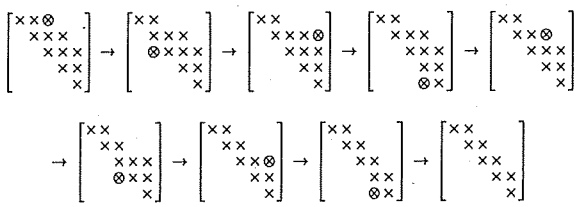


Figure 8.3. Bidiagonalization of an upper tridiagonal matrix of order 5 by means of Givens rotations.

$20i + O(1)$ flops are required at the i–th iteration. Hence the total number of flops is $10N^2$. The `matlab`–like function `tridbid`, described in Appendix C.7, reduces an upper tridiagonal matrix of even order N into an upper bidiagonal one.

8.7 Conclusions

In § 8.2 and § 8.3, two algorithms for the solution of linear systems of equations where the coefficient matrix is diagonal plus a semi–separable matrix have been presented. Both algorithms are stable with linear complexity. Furthermore the one presented in § 8.3 is suitable for an implementation on a system of two processors, with speed-up equal to 1.84.

In § 8.4, the one-way chasing algorithm to reduce a diagonal plus a symmetric semi–separable matrix to a tridiagonal one, described in [22], has been presented.

In § 8.5, a two-way chasing algorithm to reduce a diagonal plus a symmetric semi–separable matrix to a tridiagonal one has been presented. The required number of flops for the two–way chasing algorithm is half of that described in [22].

In § 8.6, a fast algorithm to reduce a diagonal plus an unsymmetric semiseparable matrix to a bidiagonal one has been presented. This algorithm is divided into three steps requiring a computational cost of order N^2.

All the algorithms considered in this chapter can be easily generalized to a banded plus semi-separable matrix whose off-diagonal part has rank bigger than 1 (see [20], [22] for details).

Chapter 9

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In the first section of this chapter the conclusions are summarized. A second section describes open problems and outlines suggestions for further research.

9.1 Conclusions

In this thesis we have treated fast and reliable algorithms for solving different problems arising in Signal Processing and System Identification. We now present the conclusions and original contributions.

- Algorithms for solving Structured Total Least Squares problems have been considered, based on the generalized Schur algorithm. In particular, exploiting the structure of the involved generators we have managed to design algorithms that reduce the computational complexity of the generalized Schur algorithm by at least one order of magnitude. Despite the fact that these algorithms require two hyperbolic rotations at each iteration of these algorithms, we have proved that they are backward stable. Compared with the algorithms known in literature for solving STLS problems, the proposed algorithms are much faster and also reliable, even for small sized problems.
- Recent approaches in subspace identification problems require the computation of the R factor of the QR factorization of a block–Hankel matrix H, which, in general has a huge number of rows. We have considered a fast algorithm to compute the R factor based on the generalized Schur algorithm. This algorithm is more than ten times faster than the classical QR algorithm. Moreover it allows to handle the rank–deficient case.
- By deriving a recurrence relation for the principal minors of a symmetric definite positive Toeplitz matrix, a fast algorithm for computing its smallest eigenvalue, and corresponding eigenvector, has been derived. It relies solely upon a modified version of the Levinson–Durbin algorithm, combined with a version of Newton's method. Although its conceptual semplicity, the performances of this algorithm turn out to be comparable with those ones of the other proposed algorithms for solving such a problem.
- Fast and stable algorithms for solving basic linear algebra problems involving diagonal plus semi–separable matrices have been proposed. In particular, fast and stable algorithms for solving linear systems involving DSS matrices, for reducing symmetric DSS matrices into symmetric tridiagonal ones and for reducing DSS matrices into bidiagonal ones by means of orthogonal transformations are considered. The considered algorithms are at least two times faster than stable algorithms known in the literature for solving such problems.

9.2 Further research and open problems

- In the last decade, many *superfast* algorithms have been proposed for solving linear systems involving structured matrices $S \in \mathbb{R}^{(m \times n)}$ (see i.e. [66], [71],

[125], [73]). The computational complexity of these algorithms is $O((m+n)\log^2(m+n))$ which is lower than those of *fast* algorithms ($O(mn)$ computational complexity) when the size of the problem is large enough. It would be interesting to design superfast algorithms for Structured Total Least Squares problems. The idea is to transform the structured linear systems, kernel of the STLS algorithms described in Ch. 4 and Ch. 5, into interpolation problems. Solving the latter problems, indeed, is equivalent to computing polynomial vectors that satisfy interpolation conditions in the roots of unity [125], [73]. Because this particular choice of the interpolation points, superfast algorithms can be built based on the Fast Fourier Transform (FFT).

- The STLS algorithms presented in this thesis are restricted to Hankel/Toeplitz matrices and their block version. However, many problems in different fields are formulated in terms of matrices with other kinds of stucture (e.g., Cauchy, Vandermonde and Löwner matrices). Part of the future work will be devoted to the study of fast and reliable algorithms for solving Structured Total Least Squares and Least Squares problems involving these structures.

- In Ch. 7 a fast algorithm for computing the smallest eigenvalue, and corresponding eigenvector, of symmetric positive definite Toeplitz matrices, has been derived. It relies upon a modified version of the Levinson–Durbin algorithm. An algorithm based on the generalized Schur algorithm can also be derived. The aim of further research is to study such an implementation and its convergence properties.

- Finally, it should be noted that the STLS algorithms considered in this dissertation are restricted to STLS problems $Ax \approx b$, with A full rank and one right–hand side vector b. Further research will focus to generalizations of these problems towards other norms (e.g. L_1 norm or L_∞ norm), more than one right–hand side ($Ax \approx B$, in which B is a matrix), rank–deficient problems which need appropriate use of regularization to improve robustness, and nonlinearly STLS problems.

Appendix A

ELEMENTARY TRANSFORMATIONS

In this appendix we review elementary (orthogonal and hyperbolic) transformations that can be used to annihilate selected entries in a vector and thereby reduce a generator in proper form.

A.1 ELEMENTARY HOUSEHOLDER TRANSFORMATIONS

Suppose we wish to transform an $n-$dimensional row vector $x = [\ x_1 \quad x_2 \quad \dots \quad x_n\ ]$ to the form $[\ \alpha \quad 0 \quad 0 \quad 0\ ]$.

One way to achieve this transformation is to apply a so-called *Householder reflection* Θ: it performs the transformation

$$[\ x_1 \quad x_2 \quad \dots \quad x_n\ ]\,\Theta = \alpha\, e_1^T\ , \tag{A.1.1}$$

for some α to be determined, with Θ orthogonal and symmetric, (*i.e.*, $\Theta\Theta^T = I = \Theta^T\Theta$ and $\Theta = \Theta^T$). In particular, from (A.1.1) it follows that α must be equal to $\|x\|_2$. Indeed,

$$x\Theta\Theta^T x^T = \|x\|_2^2 = |\alpha|^2.$$

Then

$$\alpha = \pm\|x\|_2. \tag{A.1.2}$$

Define

$$\Theta \triangleq I - 2\,\frac{g^T g}{g g^T} \quad \text{where} \quad g = x - \alpha\, e_1^T\ , \tag{A.1.3}$$

and $\alpha = \pm\|x\|_2$. Then is straightforward to verify that

$$[\ x_1 \quad x_2 \quad \dots \quad x_n\ ]\,\Theta = \mp\|x\|_2 e_1^T\ ,$$

We remark that, in order to avoid cancellation in the computation of g, the sign in (A.1.2) is chosen the same as the sign of the leading entry of the row vector x, *viz.*, the sign of x_1.

A.2 ELEMENTARY CIRCULAR OR GIVENS ROTATIONS

Suppose we wish to transform a $2-$dimensional row vector $x = [\ a \quad b\ ]$ to the form $[\ \alpha \quad 0\ ]$. This can be accomplished by applying to the vector x the elementary 2×2 orthogonal rotation Θ (also known as Givens or circular rotation)

$$\Theta = \frac{1}{\sqrt{1+\rho^2}} \begin{bmatrix} 1 & -\rho \\ \rho & 1 \end{bmatrix} \quad \text{where} \quad \rho = \frac{b}{a}\ ,\ a \neq 0.$$

It turns out that

$$[\ a \quad b\ ]\,\Theta = \pm\sqrt{1+\rho^2}.$$

A.3 HYPERBOLIC TRANSFORMATIONS

In many cases, it is necessary to use hyperbolic transformations rather than orthogonal transformations. We therefore exhibit here the necessary modifications.

Elementary Hyperbolic Rotations

Let

$$J = \begin{bmatrix} 1 & \\ & -1 \end{bmatrix}.$$

A 2×2 matrix Θ is an elementary hyperbolic transformation if

$$\Theta J \Theta^T = J.$$

Let x be the row vector $x = \begin{bmatrix} a & b \end{bmatrix}$. We are interested in considering a hyperbolic rotation Θ, such that, the left-multiplication of x by Θ annihilates one of the entries of x.

More precisely, Θ performs either transformations:

$$\begin{bmatrix} a & b \end{bmatrix} \Theta = \begin{bmatrix} \alpha & 0 \end{bmatrix} \quad \text{if } |a| > |b| , \tag{A.3.1}$$

$$\begin{bmatrix} a & b \end{bmatrix} \Theta = \begin{bmatrix} 0 & \alpha \end{bmatrix} \quad \text{if } |a| < |b| . \tag{A.3.2}$$

Furthermore, it turns out

$$\begin{cases} \alpha^2 = a^2 - b^2 & \text{if } \quad |a| > |b|, \\ \alpha^2 = b^2 - a^2 & \text{if } \quad |b| > |a|. \end{cases}$$

An expression for a $J-$orthogonal hyperbolic rotation Θ that achieves (A.3.1) or (A.3.2) is given by

$$\Theta = \frac{1}{\sqrt{1-\rho^2}} \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} \quad \text{where } \rho = -\frac{b}{a} , \ a \neq 0 \ , \text{ if } |a| > |b| \tag{A.3.3}$$

$$\Theta = \frac{1}{\sqrt{1-\rho^2}} \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} \quad \text{where } \rho = -\frac{a}{b} , \ b \neq 0 \ , \text{ if } |a| < |b| \tag{A.3.4}$$

It can be verified by direct calculation that

$$\begin{bmatrix} a & b \end{bmatrix} \Theta = \begin{bmatrix} \pm\sqrt{a^2 - b^2} & 0 \end{bmatrix} \quad \text{when } |a| > |b| ,$$

$$\begin{bmatrix} a & b \end{bmatrix} \Theta = \begin{bmatrix} 0 & \pm\sqrt{b^2 - a^2} \end{bmatrix} \quad \text{when } |b| > |a| .$$

The numerical stability of hyperbolic rotations depends on the implementation. If it is implemented as in (A.3.1) and (A.3.2) it can give rise to large errors in the computation [16], [66]. There exist alternative implementations of the hyperbolic rotation Θ that exhibit better numerical properties. Here we briefly mention three stable modifications.

Mixed Downdating

Assume we apply a hyperbolic rotation Θ to a row vector $\begin{bmatrix} x & y \end{bmatrix}$, say

$$\begin{bmatrix} x_1 & y_1 \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \frac{1}{\sqrt{1-\rho^2}} \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} . \tag{A.3.5}$$

Then, more explicitly,

$$x_1 = \frac{1}{\sqrt{1-\rho^2}} [x + \rho y] , \tag{A.3.6}$$

$$y_1 = \frac{1}{\sqrt{1-\rho^2}} [\rho x + y] . \tag{A.3.7}$$

Solving for x in terms of x_1 from the first equation and substituting into the second equation we obtain

$$y_1 = \rho x_1 + \sqrt{1-\rho^2}\, y . \tag{A.3.8}$$

An implementation that is based on (A.3.6) and (A.3.8) is said to be in mixed downdating form. It has better numerical stability properties than a direct implementation of Θ as in (A.3.5) (see [16]). In the above mixed form, we first evaluate x_1 and then use it to compute y_1. We can obtain a similar procedure that first evaluates y_1 and then uses it to compute x_1. For this purpose, we solve for y in terms of y_1 from (A.3.7) and substitute into (A.3.6) to obtain

$$x_1 = \rho y_1 + \sqrt{1-\rho^2}\, x . \tag{A.3.9}$$

Eqs. (A.3.7) and (A.3.9) represent the second mixed form.

An hyperbolic rotation in mixed downdating form is also called an hyperbolic rotation in *factored* form because (A.3.6) and (A.3.8) can be expressed in the following in matrix form

$$\begin{bmatrix} x_1 & y_1 \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \rho & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{1-\rho^2}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \rho \\ 0 & \sqrt{1-\rho^2} \end{bmatrix} .$$

The OD Method

The OD (Orthogonal-Diagonal) procedure is based on the eigendecomposition of the hyperbolic rotation Θ. Let $\rho = -b/a$, where $|a| > |b|$. Then it is straightforward to verify that any hyperbolic rotation of this form admits the following eigendecomposition:

$$\Theta = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{\frac{a+b}{a-b}} & 0 \\ 0 & \sqrt{\frac{a-b}{a+b}} \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \frac{1}{\sqrt{2}} \triangleq QDQ^T, \tag{A.3.10}$$

where the matrix

$$Q = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

is orthogonal ($QQ^T = I$), and T denotes transposition.

Due to the special form of the factors (Q, D), an hyperbolic rotation, with $|\rho| < 1$ can then be applied to a row vector $[\ x \quad y\ ]$ to yield $[\ x_1 \quad y_1\ ]$ as follows (note that the first and last steps involve simple additions and subtractions):

$$[\ x' \quad y'\ ] \leftarrow [\ x \quad y\ ] \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

$$[\ x'' \quad y''\ ] \leftarrow [\ x' \quad y'\ ] \begin{bmatrix} \frac{1}{2}\sqrt{\frac{a+b}{a-b}} & 0 \\ 0 & \frac{1}{2}\sqrt{\frac{a-b}{a+b}} \end{bmatrix}$$

$$[\ x_1 \quad y_1\ ] \leftarrow [\ x'' \quad y''\ ] \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

This procedure is numerically stable, as shown in [23].

A.3.1 The H Procedure

Another method introduced in [23] is the following. Let $\rho = -\beta/\alpha$ be the reflection coefficient of a hyperbolic rotation Θ,

$$\Theta = \frac{1}{\sqrt{1-\rho^2}} \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix},$$

with $|\rho| < 1$. Let $[\ x_1 \quad y_1\ ]$ and $[\ x \quad y\ ]$ be such that

$$[\ x_1 \quad y_1\ ] = [\ x \quad y\ ]\,\Theta, \quad \text{with} \quad |x| > |y|.$$

Algorithm A.3.1 (The H Procedure) *Given a hyperbolic rotation Θ with reflection coefficient $\rho = \beta/\alpha$, $|\rho| < 1$, and a prearray $[\ x \quad y\ ]$ with $|x| > |y|$, the postarray $[\ x_1 \quad y_1\ ]$ can be computed as follows:*

If $\frac{\beta}{\alpha}\frac{y}{x} < 1/2$
  then $\xi = 1 - \frac{\beta}{\alpha}\frac{y}{x}$
else
  $d_1 = \frac{|\alpha|-|\beta|}{|\alpha|}$, $d_2 = \frac{|x|-|y|}{|x|}$
  $\xi = d_1 + d_2 - d_1 d_2$
endif
$x_1 = \frac{|\alpha| x \xi}{\sqrt{(\alpha-\beta)(\alpha+\beta)}}$
$y_1 = x_1 - \sqrt{\frac{\alpha+\beta}{\alpha-\beta}}\,(x-y).$

$\Diamond$

In [23] it is proved that the H procedure is forward stable.

Appendix B

MATLAB–LIKE FUNCTIONS OF STLS PROBLEMS

In this appendix the matlab–like programs of the algorithms described in Ch. 4 and Ch. 5 are listed.

B.1 Matlab-like program for the Toeplitz/Hankel STLS problem

The algorithm described in Ch 4, § 4.3.1 is summarized in the function `FastLD`. `generator` is a Matlab function that, given the matrix $\hat{K}$, computes the generators $g_i, i = 1 \ldots, 4$, with respect to the block shift function Z. `givens` is a Matlab function that computes the coefficients of the Givens rotation. The variables t_1, t_2, t_3, t_4 are temporary variables like *temp* in which the partial product (4.3.19) is stored.

`function`$[L, D]$=`FastLD`(K, m, n)

$mn = m + n$;
$(g_1, g_2, g_3, g_4) =$ `generator`(K);
% Phase 1
$L(1 : mn, 1) = g_1$;
$g_1(2 : mn) = g_1(1 : mn - 1)$; $g_1(m + 1) = 0$;
$D(1) = -1$;
$t_3 = g_3(m + 1)$;
$temp = 1$;
% Phase 2
`for` $i = 2 : m - n$,
  $t_4 = g_4(m + 1)$;
  $(c, s) =$ `givens`$(g_1(i), g_4(i))$;
  $g_4([i : n + i, m + 1 : mn]) = -sg_1([i : n + i, m + 1 : mn]) + cg_4([i : n + i, m + 1 : mn])$;
  $g_1(m + 1) = s(t_4 - t_3)/c$;
  $t_3 = c \times t_3 - s \times g_1(m + 1)$;
  `if` $i < m - n$,
    $temp = temp \times c$;
    $g_4(n + i + 1) = temp \times g_4(n + i + 1)$;
  `end`
  % stabilized hyperbolic rotation
  $s = g_2(i)/g_1(i)$;
  $c_1 = \sqrt{g_1(i)^2 - g_2(i)^2}$;
  $c = c_1/g_1(i)$;
  $g_1([i + 1 : n + i, m : mn]) = (g_1([i + 1 : n + i, m : mn]) - sg_2([i + 1 : n + i, m : mn]))/c$;
  $g_2([i + 1 : n + i, m : mn]) = -sg_1([i + 1 : n + i, m : mn]) + cg_2([i + 1 : n + i, m : mn])$;
  $g_1(i) = c_1$;
  $L(i : mn, i) = g_1(i : mn)$;
  $g_1(i + 1 : mn) = g_1(i : mn - 1)$; $g_1(m + 1) = 0$;
  $D(i) = -1$;
`end`
% Phase 3
`for` $i = m - n + 1 : m$,
  $t_4 = g_4(m + 1)$;
  $(c, s) =$ `givens`$(g_1(i), g_4(i))$;

$g_4(i+1:mn) = -sg_1(i+1:mn) + cg_4(i+1:mn)$;
$g_1(m+1) = g_1(m+1) + s(t_4 - t_3)/c$;
$t_3 = c \times t_3 - s \times g_1(m+1)$;
% stabilized hyperbolic rotation
$s = g_2(i)/g_1(i)$;
$c_1 = \sqrt{g_1(i)^2 - g_2(i)^2}$;
$c = c_1/g_1(i)$;
$g_1([i+1:mn]) = (g_1([i+1:mn]) - sg_2([i+1:mn]))/c$;
$g_2([i+1:mn]) = -sg_1([i+1:mn]) + cg_2([i+1:mn])$;
$g_1(i) = c_1$;
$L(i:mn, i) = g_1(i:mn)$;
$g_1(i+1:mn) = g_1(i:mn-1)$; $g_1(m+1) = 0$;
$D(i) = -1$;
end
$g_3 = g_4$; $g_3(m+1) = t_3$;
% Phase 4
for $i = m+1:mn$,
$(c, s) =$ givens$(g_1(i), g_4(i))$;

$$\begin{pmatrix} g_1^T(i:mn) \\ g_4^T(i:mn) \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} g_1^T(i:mn) \\ g_4^T(i:mn) \end{pmatrix};$$

$(c, s) =$ givens$(g_2(i), g_3(i))$;

$$\begin{pmatrix} g_2^T(i:mn) \\ g_3^T(i:mn) \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} g_2^T(i:mn) \\ g_3^T(i:mn) \end{pmatrix};$$

% stabilized hyperbolic rotation
$s = g_1(i)/g_2(i)$;
$c_1 = \sqrt{g_2(i)^2 - g_1(i)^2}$;
$c = c_1/g_2(i)$;
$g_2([i+1:mn]) = (g_2([i+1:mn]) - sg_1([i+1:mn]))/c$;
$g_1([i+1:mn]) = -sg_2([i+1:mn]) + cg_1([i+1:mn])$;
$g_2(i) = c_1$;
$L(i:mn, i) = g_2(i:mn)$; $g_2(i+1:mn) = g_2(i:mn-1)$;
$D(i) = 1$;
end

B.2 Matlab-like program for the basic deconvolution problem

Here the algorithm for computing the Cholesky factor L of the matrix M in Ch.5, § 5.4.1, is summarized in the function `FTriang`. `generator` is a Matlab function that, given the matrix M, computes the generator $g_i, i = 1\ldots,5$ of M^TM. The functions `givens` computes the coefficients of the involved Givens rotation. The variables t_1, t_2, t_3, t_4 are temporary variables like *temp* in which the partial product (5.4.13) is stored.

function$[L]$=FTriang(M, m, n)

$(g_1, g_2, g_3, g_4, g_5) =$ generator(M);
$temp = 1$; $mn1 = m + 2n - 1$; $mn2 = m + 2n - 2$;

% Initialization
$(c, s) = \texttt{givens}(g_1(1), g_2(1))$;
$\begin{pmatrix} g_1^T([1:n, m+n:mn1]) \\ g_2^T([1:n, m+n:mn1]) \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} g_1^T([1:n, m+n:mn1]) \\ g_2^T([1:n, m+n:mn1]) \end{pmatrix}$
$L(1:mn1, 1) = g_1$; $g_1(2:mn1) = g_1(1:mn2)$; $g_1(m+n) = 0$;
% Phase 1
for $i = 2:m$,
  $(c, s) = \texttt{givens}(g_1(i), g_2(i))$;
  $\begin{pmatrix} g_1^T([i:n+i-1, m+n:mn1]) \\ g_2^T([i:n+i-1, m+n:mn1]) \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} g_1^T([i:n+i-1, m+n:mn1]) \\ g_2^T([i:n+i-1, m+n:mn1]) \end{pmatrix}$
  $t_3 = g_3(n+m)$; $t_4 = g_4(n+m)$;
  $(c, s) = \texttt{givens}(g_1(i), g_3(i))$;
  $g_3^T([i+1:n+i-1, m+n:mn1]) = -sg_1^T([i+1:n+i-1, m+n:mn1]) + cg_3^T([i+1:n+i-1, m+n:mn1])$;
  $g_1(m+n) = g_1(m+n) + s(t_3 - t_4)/c$;
  $g_4(m+n) = -sg_1(m+n) + ct_4$;
  $temp = temp \times c$;
  if $i < m$,
    $g_3(n+i) = temp \times g_3(n+i)$;
  end
  $L(i:mn1, i) = g_1(i:mn1)$;
  $g_1(i+1:mn1) = g_1(i:mn2)$; $g_1(m+n) = 0$;
end
% Phase 2
for $i = m+1:m+n-1$,
  $(c, s) = \texttt{givens}(g_1(i), g_2(i))$;
  $\begin{pmatrix} g_1^T(i:mn1) \\ g_2^T(i:mn1) \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} g_1^T(i:mn1) \\ g_2^T(i:mn1) \end{pmatrix}$
  $t_3 = g_3(n+m)$; $t_4 = g_4(n+m)$;
  $(c, s) = \texttt{givens}(g_1(i), g_3(i))$;
  $g_3^T(i+1:mn1) = -sg_1^T(i+1:mn1) + cg_3^T(i+1:mn1)$;
  $g_1(m+n) = g_1(m+n) + s(t_3 - t_4)/c$;
  $g_4(m+n) = -sg_1(m+n) + ct_4$;
  $s = g_5(i)/g_1(i)$;
  $c_1 = \sqrt{g_1(i)^2 - g_5(i)^2}$;
  $c = c_1/g_1(i)$;
  % $\rho = -s/c$
  $g_1^T(i+1:m+n) = \left(g_1^T(i+1:m+n) - sg_5^T(i+1:m+n)\right)/c$;
  $g_5^T(i+1:m+n) = -sg_1^T(i+1:m+n) + cg_5^T(i+1:m+n)$;
  $g_1(i) = c_1$;
  $L(i:mn1, i) = g_1(i:mn1)$; $g_1(i+1:mn1) = g_1(i:mn2)$; $g_1(m+n) = 0$;
end
$g_4(n+m) = t_4$; $g_4(m+n+1:mn1) = g_3(m+n+1:mn1)$;
% Phase 3
for $i = m+n:mn1$,
  $(c, s) = \texttt{givens}(g_1(i), g_2(i))$;

$$\begin{pmatrix} g_1^T(i:mn1) \\ g_2^T(i:mn1) \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} g_1^T(i:mn1) \\ g_2^T(i:mn1) \end{pmatrix}$$
$(c,s) = \texttt{givens}(g_1(i), g_3(i));$
$$\begin{pmatrix} g_1^T(i:mn1) \\ g_3^T(i:mn1) \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} g_1^T(i:mn1) \\ g_3^T(i:mn1) \end{pmatrix}$$
$(c,s) = \texttt{givens}(g_5(i), g_4(i));$
$$\begin{pmatrix} g_5^T(i:mn1) \\ g_4^T(i:mn1) \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} g_5^T(i:mn1) \\ g_4^T(i:mn1) \end{pmatrix}$$
$s = g_5(i)/g_1(i);$
$c_1 = \sqrt{g_1(i)^2 - g_5(i)^2};$
$c = c_1/g_1(i);$
% $\rho = -s/c$
$g_1^T(i+1:m+n) = \left(g_1^T(i+1:m+n) - sg_5^T(i+1:m+n)\right)/c;$
$g_5^T(i+1:m+n) = -sg_1^T(i+1:m+n) + cg_5^T(i+1:m+n);$
$g_1(i) = c_1;$
$L(i:mn1,i) = g_1(i:mn1);\ g_1(i+1:mn1) = g_1(i:mn2);$
end

◊

B.3 Matlab-like program for the modified basic deconvolution problem

Here the algorithm for computing the Cholesky factor L of the matrix M in Ch. 5, § 5.4.2 is summarized in the function `FTriang2`. `generator2` is a Matlab function that, given the matrix M, computes the generator $g_i, i = 1\ldots,5$ of M^TM. The functions `givens` computes the coefficients of the involved Givens rotation. The variables t_1, t_2, t_3, t_4 are temporary variables like *temp* in which the partial product (5.4.13) is stored.

`function`$[R]$=`Ftriang2`(M,m,n)
$(x_1,x_2,x_3,x_4,x_5) = \texttt{generator2}(M);$
% Initialization
$temp = 1;$
$R(1,1:m+n) = x_1^T;\ x_1^T(2:m+n) = x_1^T(1:m+n-1);\ x_1(m+1) = 0;$
% Phase 1
`for` $i = 2 : m-n+1,$
$t_2 = x_2(m+1);\ t_3 = x_3(m+1);$
$(c,s) = \texttt{givens}(x_1(i), x_2(i));$
$x_2^T([i+1:n+i-1, m+1:m+n]) = -sx_1^T([i+1:n+i-1, m+1:m+n])$
$+cx_2^T([i+1:n+i-1, m+1:m+n])$
$x_1(m+1) = s*(t_2 - t_3)/c;$
$x_3(m+1) = c*t_3 - s*x_1(m+1);$
$temp = temp*c;$
`if` $i < m-n+1,$
$x_2(n+i) = temp*x_2(n+i);$

```
    end
    s = x4(i)/x1(i);
    c1 = sqrt(x1(i)^2 - x4(i)^2);
    c = c1/x1(i);
    x1^T([i+1 : n+i-1, m+1 : m+n]) = (x1^T([i+1 : n+i-1, m+1 : m+n])
                                      -s x4^T([i+1 : n+i-1, m+1 : m+n])) /c;
    x4^T([i+1 : n+i-1, m+1 : m+n]) = -s x1^T([i+1 : n+i-1, m+1 : m+n])
                                     +c x4^T([i+1 : n+i-1, m+1 : m+n]);
    x1(i) = c1;
    R(i, i : m+n) = x1(i : m+n)^T;
    x1(i+1 : m+n) = x1(i : m+n-1); x1(m+1) = 0;
end
% Phase 2
for i = m-n+2 : m,
    t2 = x2(m+1); t3 = x3(m+1);
    (c, s) = givens(x1(i), x2(i));
    x2^T(i+1 : m+n) = -s x1^T(i+1 : m+n) + c x2^T(i+1 : m+n);
    x1(m+1) = x1(m+1) + s*(t2 - t3)/c;
    x3(m+1) = c*t3 - s*x1(m+1);
    (c, s) = givens(x4(i), x5(i));
    ( x4^T(i : m+n) ) = (  c   s ) ( x4^T(i : m+n) )
    ( x5^T(i : m+n) )   ( -s   c ) ( x5^T(i : m+n) )
    s = x4(i)/x1(i);
    c1 = sqrt(x1(i)^2 - x4(i)^2);
    c = c1/x1(i);
    x1^T(i+1 : m+n) = (x1^T(i+1 : m+n) + -s x4^T(i+1 : m+n)) /c;
    x4^T(i+1 : m+n) = -s x1^T(i+1 : m+n) + c x4^T(i+1 : m+n);
    x1(i) = c1;
    R(i, i : m+n) = x1(i : m+n)^T;
    x1(i+1 : m+n) = x1(i : m+n-1); x1(m+1) = 0;
end
x3(m+2 : m+n) = x2(m+2 : m+n);
% Phase 3
for i = m+1 : m+n,
    (c, s) = givens(x1(i), x2(i));
    ( x1^T(i : m+n) ) = (  c   s ) ( x1^T(i : m+n) )
    ( x2^T(i : m+n) )   ( -s   c ) ( x2^T(i : m+n) )
    (c, s) = givens(x4(i), x3(i));
    ( x4^T(i : m+n) ) = (  c   s ) ( x4^T(i : m+n) )
    ( x3^T(i : m+n) )   ( -s   c ) ( x3^T(i : m+n) )
    (c, s) = givens(x4(i), x5(i));
    ( x4^T(i : m+n) ) = (  c   s ) ( x4^T(i : m+n) )
    ( x5^T(i : m+n) )   ( -s   c ) ( x5^T(i : m+n) )
    s = x4(i)/x1(i);
    c1 = sqrt(x1(i)^2 - x4(i)^2);
    c = c1/x1(i);
    x1^T(i+1 : m+n) = (x1^T(i+1 : m+n) + -s x4^T(i+1 : m+n)) /c;
```

$x_4^T(i+1:m+n) = -sx_1^T(i+1:m+n) + +cx_4^T(i+1:m+n);$
$x_1(i) = c_1;$
$R(i, i:m+n) = x_1(i:m+n)^T;\ x_1(i+1:m+n) = x_1(i:m+n-1);$
end

Appendix C

MATLAB–LIKE FUNCTIONS FOR DIAGONAL PLUS SEMI–SEPARABLE MATRICES

C.1 Matlab–like function `semisep`

The `matlab`–like function `semisep` computes the solution of the linear system (8.2.2) in a one–way fashion, as described in § 8.2. The input of the function are the vectors d, u, v, p, q, b. The output is the solution x of the linear system (8.2.2).

The function `givens` computes the coefficients of the Givens rotation and r such that

$$\begin{bmatrix} r \\ 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}.$$

`function`$[x]$ =`semisep` (d, u, v, p, q, b);
$\tau = 0$;
`for` $i = 1 : N - 2$,
  $[u_{i+2}, c, s]$ = `givens`(u_{i+2}, u_{i+1});
  $\alpha = u_{i+1} v_{i+1}$;
  $\beta = p_{i+1} q_{i+1}$;
  $\begin{bmatrix} d_i & \alpha \\ \beta & d_{i+1} \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} d_i & \alpha \\ \beta & d_{i+1} \end{bmatrix}$; $\begin{bmatrix} b_i \\ b_{i+1} \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} b_i \\ b_{i+1} \end{bmatrix}$;
  $[d_i, c, s]$ = `givens`(d_i, α);
  $\begin{bmatrix} \beta \\ d_{i+1} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} \beta \\ d_{i+1} \end{bmatrix}$
  $\begin{bmatrix} q_{i+1} \\ q_{i+2} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} q_{i+1} \\ q_{i+2} \end{bmatrix}$
  $H_{1,i} = c$; $H_{2,i} = s$;
  $x_i = b_i / d_i$;
  $b_{i+1} = b_{i+1} - x_i p_{i+1}$;
  $\tau = \tau + x_i q_{i+1}$;
  $b_{i+2} = b_{i+2} - \tau p_{i+2}$;
`end`
$\beta = p_N q_N$;
$[d_{N-1}, c, s]$ = `givens`(d_{N-1}, β);
$\begin{bmatrix} \alpha \\ d_N \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} \alpha \\ d_N \end{bmatrix}$; $\begin{bmatrix} b_{N-1} \\ b_N \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} b_{N-1} \\ b_N \end{bmatrix}$;
$x_N = b_N / d_N$; $x_{N-1} = (b_{N-1} - \alpha x_N) / d_{N-1}$;
`for` $i = N - 2 : -1 : 1$,
  $\begin{bmatrix} x_i \\ x_{i+1} \end{bmatrix} = \begin{bmatrix} H_{1,i} & -H_{2,i} \\ H_{2,i} & H_{1,i} \end{bmatrix} \begin{bmatrix} x_i \\ x_{i+1} \end{bmatrix}$
`end`

C.2 Matlab–like function `semisep2`

The matlab–like function `semisep2` computes the solution of the linear system (8.2.2) in a two–way fashion, as described in § 8.3. The input of the function are the

vectors d, u, v, p, q, b. The output is the solution x of the linear system (8.2.2). The function is written into two columns to emphasize the parallelism of the algorithm. In particular, we denote by `top` and `bottom` the part of the algorithm that can be processed independently by two processors. `top` modifies the first half part of the involved vectors, `bottom` the last half.

`send` and `receive` are used to emphasize the data's exchange among the processors in a possible parallel implementation of the algorithm. The solution of the reduced system is written in both columns because, in a possible parallel implementation, this system must be solved by both processors. We remark that the solution of the reduced system is the only non–parallel part of the algorithm. However its computational cost is negligible because it is independent of N. The function `solvrsys` computes the solution of the reduced system (8.3.3).

`function`$[x]$ =`semisep2` (d, u, v, p, q, b);
$\tilde{N} = \lfloor N/2 \rfloor - 1$;

% top	% bottom
$\tau = 0$;	$\tau = 0$;
$\mu = 0$;	$\mu = 0$;
`for` $i = 1 : \tilde{N}$,	
$j = i + 1$;	$M = N - i$;
$[u_{i+2}, c, s] =$ `givens`(u_{i+2}, u_j);	$[p_M, c, s] =$ `givens`(u_M, u_{M+1});
$\alpha = u_j v_j$;	$\gamma = u_{M+1} v_{M+1}$;
$\beta = p_j q_j$;	$\delta = p_{M+1} q_{M+1}$;
$\begin{bmatrix} d_i & \alpha \\ \beta & d_j \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} d_i & \alpha \\ \beta & d_j \end{bmatrix}$;	$\begin{bmatrix} d_M & \gamma \\ \delta & d_{M+1} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} d_M & \gamma \\ \delta & d_{M+1} \end{bmatrix}$;
$\begin{bmatrix} b_i \\ b_j \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} b_i \\ b_j \end{bmatrix}$;	$\begin{bmatrix} b_M \\ b_{M+1} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} b_M \\ b_{M+1} \end{bmatrix}$;
$[d_i, c, s] =$ `givens`(d_i, α);	$[d_{M+1}, c, s] =$ `givens`(d_{M+1}, δ);
$\begin{bmatrix} \beta \\ d_j \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} \beta \\ d_j \end{bmatrix}$	$\begin{bmatrix} \gamma \\ d_M \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} \gamma \\ d_M \end{bmatrix}$
$\begin{bmatrix} q_j \\ q_{i+2} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} q_j \\ q_{i+2} \end{bmatrix}$;	$\begin{bmatrix} v_{M+1} \\ v_M \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} v_{M+1} \\ v_M \end{bmatrix}$;
$H_{1,i} = c$; $H_{2,i} = s$;	$H_{1,M+1} = c$; $H_{2,M+1} = s$;
$x_i = b_i / d_i$;	$x_{M+1} = b_{M+1} / d_{M+1}$;
$\tau_1 = x_i q_j$;	$\mu_1 = x_{M+1} v_{M+1}$;
% `send`(τ_1);	% `send`(μ_1);
$\tau = \tau + \tau_1$;	$\mu = \mu + \mu_1$;
% `receive`(μ_1);	% `receive`(τ_1);
$\mu = \mu + \mu_1$;	$\tau = \tau + \tau_1$;
$b_j = b_j - \beta x_i - \mu_1 u_{i+2}$;	$b_M = b_M - \gamma x_{M+1} - \tau_1 p_M$;
`if` $i < \tilde{N}$,	`if` $i < \tilde{N}$,
$b_{i+2} = b_{i+2} - \tau p_{i+2} - \mu u_{i+3}$;	$b_{M-1} = b_{M-1} - \mu u_M - \tau p_{M-1}$;
`end`	`end`

end

% send($d_{\tilde{N}+1}, u_{\tilde{N}+2}, q_{\tilde{N}+2}, b_{\tilde{N}+1}$);
% receive($d_{N-\tilde{N}}, p_{N-\tilde{N}}, v_{N-\tilde{N}}, b_{N-\tilde{N}}$);

% send($d_{N-\tilde{N}}, p_{N-\tilde{N}}, v_{N-\tilde{N}}, b_{N-\tilde{N}}$);
% receive($d_{\tilde{N}+1}, u_{\tilde{N}+2}, q_{\tilde{N}+2}, b_{\tilde{N}+1}$);

% *solution of the reduced system*

$[x_{[\tilde{N}+1:N-\tilde{N}]}]$ solvrsys= $(A^{(\tilde{N})}, b^{(\tilde{N})})$;

for $i = \tilde{N} : -1 : 1$,
  $j = i + 1$;
  $\begin{bmatrix} x_i \\ x_j \end{bmatrix} = \begin{bmatrix} H_{1,i} & -H_{2,i} \\ H_{2,i} & H_{1,i} \end{bmatrix} \begin{bmatrix} x_i \\ x_j \end{bmatrix}$
end

$[x_{[\tilde{N}+1:N-\tilde{N}]}]$ solvrsys= $(A^{(\tilde{N})}, b^{(\tilde{N})})$;

for $i = \tilde{N} + 3 : N - 1$,
  $j = i + 1$;
  $\begin{bmatrix} x_i \\ x_j \end{bmatrix} = \begin{bmatrix} H_{1,i} & -H_{2,i} \\ H_{2,i} & H_{1,i} \end{bmatrix} \begin{bmatrix} x_i \\ x_j \end{bmatrix}$
end

C.3 Matlab–like function `trid`

The matlab–like function `trid` reduces a symmetric DSS matrix into a similar symmetric tridiagonal one, as described in § 8.4. The input of the function are the vectors $\mathbf{d}, \mathbf{u}, \mathbf{v}$ and the output are the main diagonal $\mathbf{d}$ and the subdiagonal $\mathbf{a}$, of the similar tridiagonal matrix. `givens` is a function that computes the coefficients of the Givens rotation and r such that

$$\begin{bmatrix} r \\ 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}.$$

function$[d, a]$ =trid (d, u, v);
for $i = 1 : N - 2$,
  $[u_{i+2}, c, s]$ = givens(u_{i+2}, u_{i+1});
  $a_i = u_{i+1} v_{i+1}$;
  $\begin{bmatrix} d_i & a_i \\ a_i & d_{i+1} \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} d_i & a_i \\ a_i & d_{i+1} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$
  if $i \neq 1$,
    $\begin{bmatrix} e \\ a_{i-1} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} 0 \\ a_{i-1} \end{bmatrix}$
  end
  for $j = i : -1 : 2$,
    $[a_j, c, s]$ = givens(a_j, e);
    $\begin{bmatrix} d_{j-1} & a_{j-1} \\ a_{j-1} & d_j \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} d_{j-1} & a_{j-1} \\ a_{j-1} & d_j \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$
    if $i \neq 1$,
      $\begin{bmatrix} e \\ a_{j-2} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} 0 \\ a_{j-2} \end{bmatrix}$

```
        end
    end
end
```
$a_{N-1} = u_{N-1}v_{N-1}$;

C.4 Matlab–like function trid2

The matlab–like function trid2 reduces a symmetric DSS matrix into a similar symmetric tridiagonal one, as described in § 8.5. The input of the function are the vectors $\mathbf{d}, \mathbf{u}, \mathbf{v}$ and the output are the main diagonal $\mathbf{d}$ and the subdiagonal $\mathbf{a}$, of the similar tridiagonal matrix.

```
function[d, a] =trid2 (d, u, v);
for i = 1 : Ñ − 2,
```
$[u_{i+2}, c_1, s_1] = \texttt{givens}(u_{i+2}, u_{i+1})$;
$[v_{N-i}, c_2, s_2] = \texttt{givens}(v_{N-i}, v_{N-i+1})$;
$a_i = u_{i+1}v_{i+1}$;
$a_{N-i} = u_{N-i+1}v_{N-i+1}$;

$$\begin{bmatrix} d_i & a_i \\ a_i & d_{i+1} \end{bmatrix} = \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} d_i & a_i \\ a_i & d_{i+1} \end{bmatrix} \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}$$

$$\begin{bmatrix} d_{N-i} & a_{N-i} \\ a_{N-i} & d_{N-i+1} \end{bmatrix} = \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} \begin{bmatrix} d_{N-i} & a_{N-i} \\ a_{N-i} & d_{N-i+1} \end{bmatrix} \begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix}$$

```
    if i ≠ 1,
```

$$\begin{bmatrix} e \\ a_{i-1} \end{bmatrix} = \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix} \begin{bmatrix} 0 \\ a_{i-1} \end{bmatrix}$$

$$\begin{bmatrix} a_{N-i+1} \\ f \end{bmatrix} = \begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix} \begin{bmatrix} a_{N-i+1} \\ 0 \end{bmatrix}$$

```
    end
    for j = i : −1 : 2,
```
$[a_j, c_1, s_1] = \texttt{givens}(a_j, e)$;
$[a_{N-j}, c_2, s_2] = \texttt{givens}(a_{N-j}, f)$;

$$\begin{bmatrix} d_{j-1} & a_{j-1} \\ a_{j-1} & d_j \end{bmatrix} = \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} d_{j-1} & a_{j-1} \\ a_{j-1} & d_j \end{bmatrix} \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}$$

$$\begin{bmatrix} d_{N-j+1} & a_{N-j+1} \\ a_{N-j+1} & d_{N-j+2} \end{bmatrix} = \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} \begin{bmatrix} d_{N-j+1} & a_{N-j+1} \\ a_{N-j+1} & d_{N-j+2} \end{bmatrix} \begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix}$$

```
        if i ≠ 1,
```

$$\begin{bmatrix} e \\ a_{j-2} \end{bmatrix} = \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix} \begin{bmatrix} 0 \\ a_{j-2} \end{bmatrix}$$

$$\begin{bmatrix} a_{N-j+1} \\ f \end{bmatrix} = \begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix} \begin{bmatrix} a_{N-j+1} \\ 0 \end{bmatrix}$$

```
        end
    end
end
```

$a_{\tilde{N}} = u_{\tilde{N}+1} v_{\tilde{N}+1}$

C.5 Matlab–like function `tridns`

The matlab–like function `tridns` reduces a DSS matrix into a tridiagonal one by means of orthogonal transformations, as described in § 8.6.1. The input of the function are the vectors $\mathbf{d}, \mathbf{u}, \mathbf{v}, \mathbf{p}, \mathbf{q}$ and the output are the main diagonal $\mathbf{d}$ and the lower and upper diagonal $\mathbf{b}$ and $\mathbf{a}$ of the unsymmetric tridiagonal matrix.

`function`$[d, a, b]$ =`tridns` (d, u, v, p, q);
`for` $i = 1 : N - 2$,
$[u_{i+2}, c_1, s_1]$ = `givens`(u_{i+2}, u_{i+1});
$[q_{i+2}, c_2, s_2]$ = `givens`(q_{i+2}, q_{i+1});
$a_i = u_{i+1} v_{i+1}$;
$b_i = p_{i+1} q_{i+1}$;

$$\begin{bmatrix} d_i & a_i \\ b_i & d_{i+1} \end{bmatrix} = \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} d_i & a_i \\ b_i & d_{i+1} \end{bmatrix} \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix}$$

`if` $i \neq 1$,

$$\begin{bmatrix} a_{i-1} \\ e \end{bmatrix} = \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} a_{i-1} \\ 0 \end{bmatrix}, \quad \begin{bmatrix} b_{i-1} \\ f \end{bmatrix} = \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} \begin{bmatrix} b_{i-1} \\ 0 \end{bmatrix}$$

`end`
`for` $j = i : -1 : 2$,
$[a_j, c_1, s_1]$ = `givens`(a_j, e);
$[b_j, c_1, s_2]$ = `givens`(b_j, f);

$$\begin{bmatrix} d_{j-1} & a_{j-1} \\ b_{j-1} & d_j \end{bmatrix} = \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} d_{j-1} & a_{j-1} \\ b_{j-1} & d_j \end{bmatrix} \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix}$$

`if` $i \neq 1$,

$$\begin{bmatrix} e \\ e_{j-2} \end{bmatrix} = \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} \begin{bmatrix} 0 \\ e_{j-2} \end{bmatrix}, \quad \begin{bmatrix} b_{j-2} \\ f \end{bmatrix} = \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} b_{j-2} \\ 0 \end{bmatrix}.$$

`end`
`end`
`end`
$a_{N-1} = u_N v_N$;
$b_{N-1} = u_N v_N$;

C.6 Matlab–like function `triup`

The matlab–like function `triup` reduces a tridiagonal matrix into an upper tridiagonal one by means of orthogonal transformations, as described in § 8.6.2. The input of the function are the main diagonal $\mathbf{d}$, the lower diagonal $\mathbf{b}$ and the upper diagonal $\mathbf{a}$, the output are the main diagonal $\mathbf{d}$ and the upper diagonal $\mathbf{a}$.

```
function[d, a, b] =triup (d, a, b);
for i = 1 : N − 1,
    [d_i, c, s] = givens(d_i, b_i);
    [a_i; d_{i+1}] = [c  s; −s  c] [a_i; d_{i+1}]
    if i ≠ N − 1,
        [b_i; a_{i+1}] = [c  s; −s  c] [0; a_{i+1}];
    end
end
```

C.7 Matlab–like function `tridbid`

The `matlab`–like function `tribid` reduces an upper tridiagonal one into an upper bidiagonal one by means of orthogonal transformations, as described in § 8.6.3. The following `matlab`–like function `tridbid` reduces an upper tridiagonal matrix of even order N into an upper bidiagonal one. The input of the function are: the main diagonal $\mathbf{d}$, the first upper diagonal $\mathbf{a}$ and the second upper diagonal $\mathbf{b}$. The output are the main diagonal $\mathbf{d}$ and the first upper diagonal $\mathbf{a}$ of the bidiagonal matrix. If N is odd, the same function `tridbid` can be used on the modified vectors $\tilde{\mathbf{d}} = [\mathbf{d}^T, 0]^T$, $\tilde{\mathbf{a}} = [\mathbf{a}^T, 0]^T$, $\tilde{\mathbf{b}} = [\mathbf{b}^T, 0]^T$.

```
function[d, a] =tridbid (d, a, b);
for i = 1 : N − 2,
    [a_i, c, s] = givens(a_i, b_i);
[d_{i+1}  a_{i+1}; e  d_{i+2}] = [d_{i+1}  a_{i+1}; 0  d_{i+2}] [c  −s; s  c]
    for j + 1 = i : 2 : n − 3,
        [d_j, c, s] = givens(d_j, e);
        [a_j  b_j; d_{j+1}  a_{j+1}] = [c  s; −s  c] [a_j  b_j; d_{j+1}  a_{j+1}]
        [e; b_{j+1}] = [c  s; −s  c] [0; b_{i+1}]
        [b_j, c, s] = givens(b_j, e);
        [a_{j+1}  b_{j+1}; d_{j+2}  a_{j+2}] = [a_{j+1}  b_{j+1}; d_{j+2}  a_{j+2}] [c  −s; s  c]
        [e; d_{j+3}] = [c  s; −s  c] [0; d_{j+3}]
    end
    if mod (i, 2) ≠ 0,
        [d_{N−2}, c, s] = givens(d_{N−2}, e);
        [a_{N−2}  b_{N−2}; d_{N−1}  a_{N−1}] = [c  s; −s  c] [a_{N−2}  b_{N−2}; d_{N−1}  a_{N−1}]
    else
```

$[d_{N-1}, c, s] = \texttt{givens}(d_{N-1}, e);$

$$\begin{bmatrix} a_{N-1} \\ d_N \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a_{N-1} \\ d_N \end{bmatrix}$$

```
    end
end
```

BIBLIOGRAPHY

[1] T. J. Abatzoglou and J. M. Mendel, *Constrained total least squares*, Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Dallas, 1987, pp. 1485–1488.

[2] T. J. Abatzoglou and R. H. Hackman, *Constrained total least squares based extraction of acoustic resonances of elastic targets*, Proceedings of IEEE International Conf. on Acoustics, Speech & Signal Processing, pp. 3385–3388, Toronto, Ontario, Canada, May 1991.

[3] T. J. Abatzoglou, J. M. Mendel, and G. A. Harada, *The constrained total least squares technique and its applications to harmonic superresolution,* IEEE Transactions on Signal Processing, 39, 1991, pp. 1070–1086.

[4] A. Abdallah and Y. Hu, *Parallel VLSI computing array implementation for signal subspace updating algorithm*, IEEE Trans. Acoust. Speech and Signal Processing, ASSP–37, 1989, pp. 742–748.

[5] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, 1995.

[6] E. Anderson, Z. Bai, and J. J. Dongarra, *Generalized QR factorization and its applications,* Technical Report CS-91-131 (LAPACK Working Note 31), Computer Science Dept., University of Tennessee, Knoxville, 1991.

[7] M. T. Bajen, R. Puchal, A. Gonzalez, J. M. Grinyo, A. Castelao, J. Mora, and J. Martin Comin, *MAG3 renogram deconvolution in kidney transplantation: Utility of the measurement of initial tracer uptake*, J. Nucl. Med., 38 1997, pp. 1295–1299.

[8] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga, *SLICOT - a subroutine library in systems and control theory.* In B.N. Datta, Ed., Applied and Computational Control, Signals, and Circuits, vol. 1, ch. 10, pp. 499-539, Birkhäuser, Boston, MA, 1999.

[9] A. Blanc–Lapierre and B. Picibono, *Fonctions Aléatoires,* Masson, Paris, 1981.

[10] M. Berry, S. Dumais and G.O. O'Brien, *Using linear algebra for intelligent information retrieval*, SIAM Review, 37(4), 1995, pp. 573–595.

[11] A. Björck , *Numerical Methods for Least Squares Problems,* SIAM, Philadelphia, 1996.

[12] K. Blom, *Information retrieval using the singular value decomposition and Krylov subspaces*, Department of Mathematics, Chalmers University of Technology, Göteborg, Sweden, 1999.

[13] A.W. Bojanczyk, R.P. Brent, F.R. De Hoog, and D.R. Sweet, *On the stability of the Bareiss and related Toeplitz factorization algorithms,* SIAM J. Matrix Anal. Appl., 16, 1995, pp. 40–57.

[14] A.W. Bojanczyk, R.P. Brent, F.R. De Hoog, *Stability analysis of a general Toeplitz systems solver,* Numerical Algorithms 10, 1995, pp. 225–244.

[15] A. W. Bojanczyk, R.P. Brent and F. R. De Hoog, *QR factorization of Toeplitz matrices*, Numer. Math., 49, 1986, pp. 81–94.

[16] A.W. Bojanczyk, R.P. Brent, P. Van Dooren and F.R. De Hoog, *A note on downdating the Cholesky factorization,* SIAM J. Sci. Stat. Comput., 1 1980, pp. 210–220.

[17] A.W. Bojanczyk, and A.O. Steinhardt, *Stabilized hyperbolic householder transformations,* IEEE Trans. Acoust., Speech and Signal Processing ASSP–37,1989, pp. 1286-1288.

[18] J. Bunch, *Stability of methods for solving Toeplitz systems of Equations,* SIAM J. Sci. Stat. Comput., 6, 1985, pp. 349–364.

[19] A. Capderou, D. Douguet, T. Similowski, A. Aurengo, and M. Zelter, *Non-invasive assessment of technetium-99m albumin transit time distribution in the pulmonary circulation by first-pass angiocardiography*, Eur. J. Nucl. Med., 24, 1997, pp. 745–753.

[20] S.Chandrasekaran and M. Gu, *Fast and stable algorithms for banded plus semi–separable systems of linear equations*, submitted for publication, available on `http://www.math.ucla.edu/ mgu/`.

[21] S. Chandrasekaran and M. Gu, *A Divide–and–Conquer algorithm for the eigendecomposition of symmetric block–diagonal plus semi–separable matrices,* submitted for publication.

[22] S. Chandrasekaran and M. Gu, *Fast and stable eigendecomposition of symmetric banded plus semi–separable matrices,* Linear Algebra Appl., 313, 2000, pp. 107–114.

[23] S. Chandrasekaran, A.H. Sayed, *Stabilizing the generalized Schur algorithm,* SIAM J. Matrix Anal. Appl., 17(4), 1996 pp. 950–983.

[24] S. Chandrasekaran, A.H. Sayed, *A fast stable solver for nonsymmetric Toeplitz and quasi-Toeplitz systems of linear equations.* SIAM J. Mat. Anal. and Appl., 19, 1998, pp. 107–139.

[25] H. Chen, S. Van Huffel and D. Van Ormondt, *Application of the Structured Total Least Norm technique in spectral estimation,* in: Proc. of the 8th European Signal Processing Conference (EUSIPCO'96), Trieste, Italy, Sep. 1996, pp. 706-709.

[26] Y.M. Cho, G.Xu and T. Kailath, *Fast recursive identification of state space models via exploitation of displacement structure*, Automatica, 30(1), 1994, pp. 45–59.

[27] J. Chun, *Fast array algorithms for structured matrices,* PhD thesis, Dept. of Elect. Eng., Stanford University, 1989.

[28] J. Chun, T. Kailath and H. Lev–ari, *Fast parallel algorithms for QR and triangular factorization*, SIAM J. Sci. and Stat. Comp., 8(6), 1987, pp. 899–913.

[29] G. Cybenko, *A general orthogonalization technique with applications to time series analysis and signal processing*, Math. Comp., 40, 1983, pp. 323–336.

[30] G. Cybenko, *Fast Toeplitz orthogonalization using inner products*, SIAM J. Sci. Stat. Computing, 8, 1987, pp. 734–740.

[31] G. Cybenko, *The numerical stability of the Levinson–Durbin algorithm for Toeplitz systems of equations,* SIAM J. Sci. and Stat. Comp., 1, 1980, pp. 303–310.

[32] G. Cybenko, *On the eigenstructure of Toeplitz matrices,* IEEE Trans. Acoust., Speech, Signal Proc., ASSP–31, 1984, pp. 910–920.

[33] G. Cybenko and C. F. Van Loan, *Computing the minimum eigenvalue of a symmetric positive definite Toeplitz matrix,* SIAM J. Sci. Stat. Comp., 7, 1986, pp. 123–131.

[34] R. de Beer and D. van Ormondt, *Analysis of NMR data using time-domain fitting procedures*, in: NMR Basic Principles and Progress, M. Rudin and J. Seelig, eds. Springer-Verlag, 1992.

[35] S. Deerwester and S. Dumais and G. Furnas and T. Landauer and R. Harshman, *Indexing by Latent Semantic Analysis*, Journal of the American Society for Information Science, 41(6), 1990, pp. 391-409.

[36] B. De Moor, *Structured total least squares and L_2 approximation problems,* Linear Algebra Appl., 188-189, 1993, pp. 163-205.

[37] B. De Moor, *Linear system identification, structured total least squares and the Riemannian SVD,* in: Recent advances in total least squares techniques and errors-in-variables modeling, S.Van Huffel, Ed., Philadelphia, SIAM, 1997, pp. 225-238.

[38] B. De Moor, *The Riemannian singular value decomposition,* in: SVD and signal processing III: algorithms, architectures and applications, M. Moonen and B. De Moor, Ed., Elsevier, Amsterdam, 1995, pp. 61-78.

[39] B. De Moor, P. Van Overschee and W. Favoreel, *Numerical Algorithms for Subspace State-Space System Identification – An Overview*, in Applied and Computational Control, Signals and Circuits, Vol. 1, Chapter 6, Birkhäuser, Boston, 1999.

[40] Y. Eidelman and I. Gohberg, *Inversion formulas and linear complexity algorithms for diagonal plus semi–separable matrices*, Computers and Mathematics with Applications, 33(4), 1997, pp. 69–79.

[41] Y. Eidelman and I. Gohberg, *A look-ahead block Schur algorithm for diagonal plus semi-separable matrices*, Computers and Mathematics with Applications, 35(10), 1998, pp. 25–34.

[42] L. Eldén, *Algorithms for the regularization of ill-conditioned least squares problems*, BIT, 17, 1979, pp. 134–145.

[43] R.R. Ernst and G. Bodenhausen and A. Wokaun, *Principles of Nuclear Magnetic Resonance in One and Two Dimensions*, Clarendon Press, Oxford, 1987.

[44] J. S. Fleming and B. A. Goddard, *A technique for the deconvolution of the renogram*, Phys. Med. Biol., 19, 1974, pp. 546–549.

[45] J. S. Fleming, *Measurement of Hippuran plasma clearance using a gamma camera*, Phys. Med. Biol., 22 1977, pp. 526–530.

[46] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, New York, 1987.

[47] D.G. Gadian, *Nuclear Magnetic Resonance and its applications to living systems*, Clarendon Press, Oxford, 1982.

[48] K.A. Gallivan, S. Thirumalai, P. Van Dooren and V. Vermaut, *High performance algorithms for Toeplitz and block Toeplitz matrices,* Linear Algebra Appl., 241-243 1996, pp. 343-388.

[49] K F. Gauss, *Teoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium. Reprinted translation: theory of the Motion of the Heavinly Bodies Moving about the Sun in Conic Sections*, Dover, New York, 1963.

[50] A. Gersho and R. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Boston, 1992.

[51] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London, 1981.

[52] G. H. Golub and C.F. Van Loan, *An analysis of the total least squares problem, SIAM J. Numer. Anal.*, 17, 1980, pp. 883–893.

[53] G. H. Golub and C. F. Van Loan, *Matrix Computations,* Third ed., The John Hopkins University Press, Baltimore, MD, 1996.

[54] L. Greengard and V. Rokhlin, *On the numerical solution of two-point boundary value problems,* Comm. Pure Appl. Math., XLIV, 1991, pp. 419–452.

[55] M.H. Hayes and M.A. Clements, *An Efficient Algorithm for Computing Pisarenko's Harmonic Decomposition using Levinson's Recursion,* IEEE Trans. Acoust. Speech Signal Process. ASSP–34, 1986, pp. 485–491.

[56] G. Heinig and K. Rost, *Algebraic Methods for Toeplitz-like Matrices and Operators*, Akademie-Verlag, Berlin, and Birkhäuser, Boston, 1984.

[57] R. Howman Giles, A. Moase, K. Gaskin, and R. Uren, *Hepatobiliary scintigraphy in a pediatric population: Determination of hepatic extraction fraction by deconvolution analysis*, J. Nucl. Med., 34, 1993, pp. 214–221.

[58] Y.H. Hu and S.Y. Kung, *Computation of the minimum eigenvalue of a Toeplitz matrix by the Levinson algorithm,* Proceedings SPIE 25th International Conference (Real Time Signal Processing), San Diego, August 1980, pp. 40–45.

[59] Y.H. Hu and S.Y. Kung, *A Toeplitz eigensystem solver,* IEEE Trans. Acoust. Speech Signal Process. ASSP–33 1986, pp. 1264–1271.

[60] I. C. F. Ipsen, *Computing an eigenvector with inverse iteration,* SIAM Rev. 39(2),1997, pp. 254-291.

[61] E. P. Jiang, *Information retrieval and filtering using the Riemannian SVD*, Dept. of Computer Science, University of Tennessee, Knoxville, TN, 1998.

[62] E. P. Jiang and M. W. Berry, *Information Filtering Using the Riemannian SVD (R-SVD)*, in: Proceedings of the Fifth International Symposium on: Solving Irregularly Structured Problems in Parallel, Lecture Notes in Computer Science 1457, 1998, pp. 386–395.

[63] June–Yub Lee and L. Greengard, *A fast adaptive numerical method for stiff two–point boundary value problems,* SIAM J. Scient. Comput., 18(2), 1997, pp. 403–429.

[64] T. Kailath, J. Chun, *Generalized displacement structure for block–Toeplitz, Toeplitz–block and Toeplitz–derived matrices.* SIAM J. Matrix Anal. Appl., 15, 1994, pp. 114–128.

[65] T. Kailath, S. Kung and M. Morf, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 1979, pp. 395–407.

[66] T. Kailath, *Displacement structure and array algorithms,* in Fast Reliable Algorithms for Matrices with Structure, T. Kailath and A. H. Sayed, Ed., SIAM, Philadelphia, 1999.

[67] T. Kailath, A.H. Sayed, *Displacement structure: Theory and applications,* SIAM Review, 37, 1995, pp. 297–386.

[68] S.M. Kay and S.L. Marple, Jr., *Spectrum analysis–A modern perspective,* Proc. IEEE, 69, 1981, pp. 1380-1419.

[69] M.A. Kohler and L.M. Supplee, *Progress Towards a New Government Standard 2400 bps Voice Coder*, in: Proc. Int. Conf. Acoust., Speech and Signal Proc., Detroit, Michigan, 1995.

[70] W.B. Kleijn and K.K. Paliwal, *Speech Coding and Synthesis*, Elsevier, 1995.

[71] P. Kravanja, and M. Van Barel, *A fast Hankel solver based on an inversion formula for Loewner matrices,* Lin. Algebra Appl. 282(1–3), 1998, pp. 275-295.

[72] S. Y. Kung and K. S. Arun, and D. V. Bhaskar Rao, *State-space and singular-value decomposition-based approximation methods for the harmonic retrieval problem*, J. Opt. Soc. Amer., 73 1983, pp. 1799–1811.

[73] P. Kravanja and M. Van Barel, *Algorithms for solving rational interpolation problems related to fast and superfast solvers for Toeplitz systems,* in: Advanced Signal Processing Algorithms, Architectures, and Implementations IX, part of the proceedings of SPIE's International Symposium on Optical Science, Engineering, and Instrumentation, 18–23 July 1999, Denver, CO, U.S.A., SPIE, 1999, pp. 359–370.

[74] S. Y. Kung and K. S. Arun, and D. V. Bhaskar Rao, *State-space and singular-value decomposition-based approximation methods for the harmonic retrieval problem*, J. Opt. Soc. Amer., 73 1983, pp. 1799–1811.

[75] H. Lev-Ari and T. Kailath, *Triangular factorization of structured Hermitian matrices*, Operator Theory: Advances and Applications, 18, 1986, pp. 301–324.

[76] P. Lemmerling, I. Dologlou, and S. Van Huffel, *Speech compression based on exact modeling and structured total least norm optimization*, in: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'98), Vol. 1, Seattle, WA, 1998, pp. 353–356.

[77] P. Lemmerling, I. Dologlou, and S. Van Huffel, *Variable rate speech compression based on exact modeling and waveform vector quantization,* Signal Processing Symposium (SPS 98), pp. 127–130, Katholieke Universiteit Leuven, Leuven, March 1998. IEEE Benelux Signal Processing Chapter.

[78] P. Lemmerling and S. Van Huffel, *Analysis of the structured total least squares problem for Hankel/Toeplitz matrices*, Technical Report Internal Report 98-100, ESAT-SISTA/COSIC, K.U.Leuven, 1998.

[79] P. Lemmerling, *Structured total least squares: analysis,algorithms and applications*, PhD thesis, Dept. of Elect. Eng., ESAT-SISTA/COSIC, Katholieke Universiteit Leuven, May 1999.

[80] P. Lemmerling, N. Mastronardi and S. Van Huffel, *Fast algorithm for solving the Hankel/Toeplitz structured total least squares problem,* Numerical Algorithms, 23, 2000, pp. 371–392.

[81] P. Lemmerling and B. De Moor and S. Van Huffel, *On the equivalence of constrained total least squares and structured total least squares*, IEEE Trans. on Signal Processing, 44(11), 1996, pp. 2908–2910.

[82] P. Lemmerling, S. Van Huffel, and B. De Moor, *Structured total least squares problems: Formulations, algorithms and applications*, in: Recent Advances in Total Least Squares Techniques and Errors-in-Variables Modeling, S. Van Huffel, Ed., SIAM, Philadelphia, 1997, pp. 215–223.

[83] H. Lev-Ari and T. Kailath, *Triangular factorization of structured Hermitian matrices*, Operator Theory: Advances and Applications, 18, 1986, pp. 301–324.

[84] T.Y. Li and Z. Zeng, *The Laguerre iteration in solving the symmetric tridiagonal eigenproblem revisited,* SIAM J. Sci. Comput., 15(5), 1994, pp. 1145–1173.

[85] L. Ljung, *System identification: theory for the user,* Prentice-Hall, Inc., 1987.

[86] W. Mackens, H. Voss, *The Minimum Eigenvalue of a Symmetric Positive-Definite Toeplitz Matrix and Rational Hermitian Interpolation,* SIAM J. Matrix Anal. Appl., 18, 1997, pp. 521–534.

[87] J. Makhoul, *On the eigenvectors of asymmetric Toeplitz matrices,* IEEE Trans. Acoust., Speech and Signal Processing, ASSP–29, 1981, pp. 868-872.

[88] N. Mastronardi, *Some algorithms for computing the minimum eigenvalue of a symmetric positive definite Toeplitz matrix,* UMSI 97/90, Supercomputing Institute, University of Minnesota, 1997.

[89] N. Mastronardi and D. Boley, *Computing the smallest eigenpair of a symmetric positive definite Toeplitz matrix,* SIAM J. Sci. Comput. 20(5), 1999, pp. 1921-1927.

[90] N. Mastronardi, S. Chandrasekaran and S. Van Huffel, *Fast and Stable algorithms for reducing diagonal plus semi–separable matrices to tridiagonal and bidiagonal form,* BIT, 41(1), 2001, pp. 149–157.

[91] N. Mastronardi, S. Chandrasekaran, and S. Van Huffel, *Fast and stable two-way algorithm for diagonal plus semi-separable systems of linear equations,* Numerical Linear Algebra with Applications, 8, 2001, pp. 7–12.

[92] N. Mastronardi, D. Kressner, V. Sima, P. Van Dooren, and S. Van Huffel, *A fast algorithm for subspace state-space system identification via exploitation of the displacement structure,* J. Comput. Appl. Math., July 2001, to appear.

[93] N. Mastronardi, P. Lemmerling and S. Van Huffel, *Fast structured total least squares algorithm for solving the basic deconvolution problem,* SIAM J. Matrix Anal. Appl., 22(2), 2000, pp. 533–553.

[94] N. Mastronardi, P. Lemmerling and S. Van Huffel, *The structured total least squares problem,* to appear in Contemporary Mathematics, AMS volume, Proceedings of the Conference *"Structured Matrices in Operator Theory, Numerical Analysis, Control, Signal and Image Processing,"*, Boulder, June 27–July 1, 1999.

[95] N. Mastronardi and G. Piazza, *Limite a posteriori del numero di condizione in norma spettrale per matrici tridiagonali,* Calcolo, 1–2(29), 1992, pp. 81–96.

[96] N. Mastronardi, P. Van Dooren, and S. Van Huffel, *Fast implementation of the QR factorization in Subspace Identification,* in Proc. of the UKACC International Conference CONTROL 2000, Cambridge (UK), September 2000.

[97] N. Mastronardi, P. Van Dooren, and S. Van Huffel, *On the stability of the generalized Schur algorithm,* Proceedings of the conference *"Numerical Analysis and Its applications",* Rousse, Bulgaria, June 2000. Lecture Notes in Computer Science, 1988, 2001, pp. 560–567.

[98] H. Park, L. Elden, *Stability analysis and fast algorithms for triangularization of Toeplitz matrices*, Numer. Math., Vol. 76(3), 1996, pp. 383–402.

[99] B. N. Parlett, *The Symmetric Eigenvalue Problem,* Prentice Hall, Englewood Cliffs, NJ, 1980.

[100] D.B. Paul, The Spectral Envelope Estimation Vocoder, IEEE Trans. on Acoust., Speech and Signal Proc., 29, 1981, pp. 786–794.

[101] J. J. Pedroso de Lima, *Nuclear medicine and mathematics*, Eur. J. Nucl. Med., 23, 1996, pp. 705–719.

[102] B. Picibono, *Fast Algorithms for Brownian Matrices,* IEEE Trans. Acoust. Speech and Signal Processing, ASSP–30, 1982, pp. 100–104.

[103] B. Picibono, *A geometrical interpretation of signal detection and estimation,* IEEE Trans. Inform. Theory, IT–26, 1980, pp. 493–497.

[104] W.W.F. Pijnappel and R. de Beer and D. van Ormondt, *State space modelling of medical NMR signals*, in: Proc. PRORISC, J.P. Veen and M.J. de Ket eds., STW, Utrecht, 1992, pp. 225-230.

[105] C.M. Rader and A.O. Steihardt, *Hyperbolic Householder transforms,* SIAM J. Matrix Anal. Appl., 9 1988, pp. 269–290.

[106] J. B. Rosen, H. Park, and J. Glick, *Total Least Norm Problem: Formulation and Algorithms*, Preprint 94–041, Army High Performance Computing Research Center, University of Minnesota, 1993, revised 1994.

[107] J.B. Rosen, H. Park, and J. Glick. *Total least norm formulation and solution for structured problems,* SIAM Journal on Matrix Anal. Appl., 17(1), 1996, pp. 110–128.

[108] V.F. Pisarenko, *On the estimation of the spectra by means of nonlinear functions,* Geophys. J. Roy. Astron. Soc., 1972, pp. 511–531.

[109] V.F. Pisarenko, *The retrieval of harmonics from a covariance function,* Geophys. J. Roy. Astron. Soc., 33, 1973, pp. 347–366.

[110] A. H. Sayed, *Displacement structure in Signal Processing and Mathematics,* PhD thesis, Dept. of Elect. Eng., Stanford University, August 1992.

[111] A. H. Sayed and T. Kailath, *A look-ahead block Schur algorithm for Toeplitz-like matrices*, SIAM J. Matrix Anal. Appl., 16, 1995, pp. 388–413.

[112] A. H. Sayed and T. Kailath, *Fast algorithms for generalized displacement structures and lossless systems*, Linear Algebra and Its Applications, 219, 1995, pp. 49–78.

[113] I. Schur, *Über potenzreihen die im Inneren des Einheitskreises beschränkt sind*, Journal für die Reine und Angewandte Mathematik, 147, 1917, pp. 205–232. (English translation in *Operator Theory: Advances and Applications*, vol. 18, pp. 31–88, edited by I. Gohberg, Birkhäuser, Boston, 1986).

[114] H. R. Schwarz, *Tridiagonalization of a symmetric banded matrix,* Numer. Math., 12, 1968, pp. 231–241.

[115] V. Sima, *Subspace-based algorithms for multivariable system identification*, Studies in Informatics and Control, 5(4), 1996, pp. 335–344.

[116] V. Sima, *Cholesky or QR factorization for data compression in subspace–based identification?*, in: Proceedings Second NICONET Workshop, Paris–Versailles, Dec. 3 1999, pp.75–80.

[117] H. P. Starr, Jr., *On the numerical solution of One–dimensional Integral and Differential Equations,* PhD Thesis, Dept. of Computer Science, Yale University, May 1992.

[118] G. W. Stewart, *Introduction to Matrix Computations,* Academic Press,New York, 1973.

[119] M. Stewart and P. Van Dooren, *Stability issues in the factorization of structured matrices*, SIAM J. Matrix Anal. Appl., 18, 1997, pp. 104–118.

[120] P. Stoica, R. L. Moses, B. Friedlander, and T. Söderström, *Maximum likelihood estimation of the parameters of multiple sinusoids from noisy measurements*, IEEE Trans. Acoust., Speech Signal Process., 37 1989, pp. 378–391.

[121] D. R. Sweet, *Numerical Methods for Toeplitz Matrices*, PhD thesis, University of Adelaide, 1982.

[122] D. R. Sweet, *Fast Toeplitz orthogonalization*, Numer. Math., 43, 1984, pp. 1–21.

[123] J.F. Traub, *Iterative methods for the solution of equations,* Englewood Cliff, N.J., Prentice Hall, 1962.

[124] W.F. Trench, *Numerical solution of the eigenvalue problem for hermitian Toeplitz matrices,* SIAM J. Matrix Anal.Appl., 10(2), 1989, pp. 135–146.

[125] M. Van Barel and P. Kravanja, *A stabilized superfast solver for indefinite Hankel systems,* Lin. Algebra Appl. 284(1–3), 1998, pp. 335-355.

[126] S. Van Huffel and H. Park, *Efficient reduction algorithms for bordered matrices,* Numerical Linear Algebra with Applications, 2(2), 1995, pp. 95–113.

[127] S. Van Huffel and H. Park and J. B. Rosen, *Formulation and Solution of Structured Total Least Norm Problems for Parameter Estimation,* IEEE Trans. on Signal Processing, 44(10), 1996, pp. 2464–2474.

[128] S. Van Huffel, H. Chen, C. Decanniere and P. Van Hecke, *Algorithm for time-domain nmr data fitting based on total least squares,* J. Magn. Reson., A110, 1994, pp. 228–237.

[129] S. Van Huffel and J. Vandewalle, *The Total Least Squares Problem: computational aspects and analysis*, SIAM, Philadelphia, 1991.

[130] S. Van Huffel, J. Vandewalle, M. Ch. De Roo, and J. L. Willems, *Reliable and efficient deconvolution technique based on total linear least squares for calculating the renal retention function*, Med. & Biol. Eng. & Comput., 25, 1987, pp. 26–33.

[131] S. Van Huffel, H. Park, and J. Ben Rosen, *Formulation and solution of structured total least norm problems for parameter estimation*, IEEE Trans. Signal Process., SP-44, 1996, pp. 2464–2474.

[132] P. Van Overschee and B. De Moor, *Subspace Identification for Linear Systems: Theory, Implementation, Applications*, Kluwer Academic Publishers, Dordrecht, 1996.

[133] P. Van Overschee and B. De Moor, `N4SID`: *Two subspace algorithms for the identification of combined deterministic-stochastic systems*, Automatica, 30(1), 1994, pp. 75–93.

[134] M.Verhaegen, *Subspace model identification. Part 3: Analysis of the ordinary output–error state–space model identification algorithm,* Int. J. Control, 58(3) 1993, pp. 555–586.

[135] J. H. Wilkinson, *The Algebraic Eigenvalue Problem,* Claredon Press, Oxford, 1965.

[136] H. Zha, *A Two–Way chasing scheme for reducing a symmetric arrowhead matrix to tridiagonal form,* J. Numerical Linear Algebra, 1, 1993, pp. 494–499.