KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
AFDELING ESAT
K. Mercierlaan 94 — 3001 Heverlee (Belgium)

JACOBI-TYPE UPDATING ALGORITHMS
for signal processing, systems identification and control

Jury :
Prof. J. Berlamont, vice-dekaan, voorzitter
Prof. J. Vandewalle, promotor
Prof. A. Bultheel
Prof. F. Catthoor
Prof. B. De Moor
Prof. J. McWhirter (RSRE - U.K.)
Prof. A. Oosterlinck
Dr. P. Van Dooren (Philips Research)

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

door

Marc MOONEN

U.D.C. 519.71

November 1990

Preface

During the course of this research, several people have provided varying degrees of support and encouragement, and I would like to hereby record my thanks for their assistance.

First of all, I wish to express my sincere gratitude and appreciation to my advisor Professor Joos Vandewalle for his enthusiastic and invaluable support and guidance.

I would like to thank Paul Van Dooren, my second and semi-official advisor, for many enlightening discussions, for carefully proofreading or even re-writing early drafts and trying to translate everything into decent English, and above all for a long-time friendship.

Thanks is also due to Bart De Moor, whose enthusiasm and creativity keeps us all going, for providing many ideas and opportunities, and for his constructive comments as a member of my thesis reading committee.

I wish to thank Professor Adhemar Bultheel, also member of the reading committee, for carefully reviewing the manuscript. I would also like to thank the other members of my examination committee, Professor J. Berlamont, Professor F. Catthoor, Professor A. Oosterlinck and especially Professor J.G. McWhirter, for taking the trouble.

I would be remiss not to thank my many colleagues at ESAT/SISTA. For providing me with the right atmosphere, I thank André Barbé, Dirk Callaerts, Jiande Chen, Johan David, Pieter De Bie, Jianbin Hao, Walter Mommaerts, Christiaan Moons, Geert Schelfhout, Johan Suykens, Jan Swevers, Shaohua Tan, Dirk Van Compernolle, Lieven Vandenberghe, Sabine Van Huffel, Jan Van Impe, Steven Van Leemput, Peter Van Overschee, and last but not least our secretary Ingrid Tokka. I am especially grateful to my

friend and one-time office mate Jan Swevers, also for making the experimental robot data available to me.

I thank Professor Gene Golub of Stanford University, for giving me the opportunity to spend a wonderful summer at Stanford, and for introducing me to American hospitality.

Finally, I would like to thank my wife Katia, for her love and patience throughout, and my parents and parents-in-law for their encouragement.

The financial support provided by the Belgian NFWO (National Fund for scientific Research) is gratefully acknowledged.

Abstract

This thesis focuses on efficient algorithms for signal processing, systems identification and control.

First of all, a **novel SVD updating algorithm** *is developed, with a significantly reduced computational complexity. It is shown how this can be used for recursive SVD-based systems identification.*

Second, a **parallel implementation** *of this algorithm is described, namely on a systolic array. A new type of systolic array is developed that implements the combined SVD updating algorithm in such a way that the data are processed at a rate which is independent of the problem size.*

Finally, it is shown how the approach to developing the SVD updating algorithm/array can also be used for developing **similar updating algorithms/arrays** *within the broad field of signal processing and systems and control. Apart from the extensions for PSVD and QSVD updating, new algorithms and systolic arrays are also developed for Kalman filtering, for recursive least squares computations with inverse factorization, and for solving Riccati and Lyapunov equations.*

Contents

Preface i

Abstract iii

Contents v

Notation vii

Nederlandse samenvatting ix

Introduction 1

1 Background material and motivation 9
- *1.1 Vectors & matrices, state space identification* . . . 10
- *1.2 QR decomposition* . . . 16
- *1.3 QR updating* . . . 22
- *1.4 SVD theorems and applications* . . . 24
- *1.5 Kogbetliantz's SVD algorithm* . . . 32
- *1.6 State space identification revisited* . . . 41
- *1.7 Conclusions* . . . 46

2 SVD updating - algorithms and analysis 49
- *2.1 A fast SVD updating algorithm* . . . 50
- *2.2 Re-orthogonalization* . . . 54
- *2.3 Subspace tracking* . . . 57
- *2.4 Error analysis* . . . 71
- *2.5 Conclusions* . . . 80

3 Systolic arrays and square root free implementation 81
- *3.1 QRD and SVD arrays* . . . 83
- *3.2 A systolic array for SVD updating* . . . 90
- *3.3 Including re-orthogonalizations* . . . 101
- *3.4 Square root free algorithms* . . . 108
- *3.5 Conclusions* . . . 118

4 Updating generalized SVD's 121
- *4.1 QSVD and QSVD-based identification* . . . 122
- *4.2 QSVD updating* . . . 131
- *4.3 PSVD updating* . . . 151
- *4.4 Conclusions* . . . 162

5 Jacobi-type algorithms for signal processing, systems and control 163
- *5.1 Riccati and Lyapunov equations* . . . 164
- *5.2 RLS with stabilized inverse factorization* . . . 175
- *5.3 RLS with orthogonal-inverse updating* . . . 189
- *5.4 Kalman filtering* . . . 202
- *5.5 Conclusions* . . . 214

Conclusions and open problems 215

Appendices 219

References 225

Notation

$\Re^{m\times n}$ *denotes the vector space of* $m \times n$ *matrices.*

$\Re^m$ *denotes* $\Re^{m\times 1}$

Matrices are denoted by capitals A, X, *etc., sometimes with an iteration index* A_1, A_2, *etc. or a parenthesized time label in subscript* $A_{[k]}$, $A_{[k+1]}$, *etc.*

a_{ij} *denotes the* (i,j) *component of* A.

$\{A\}_{ij}$ *denotes the* 2×2 *submatrix of* A *on the intersection of rows* i *and* j *and columns* i *and* j.

$\{A\}_{r:s,v:w}$ *denotes the submatrix of* A *on the intersection of rows* $r, r+1, \ldots, s$ *and columns* $v, v+1, \ldots, w$.

A^T, A^{-1} *and* A^{-T} *respectively denote the transpose of* A, *the inverse of* A *and its transpose.*

$\mathcal{R}\{A\} = span_{column}\{A\}$ *denotes the vector space generated by the columns of* A *(range).*

$\mathcal{R}^\perp\{A\} = span^\perp_{column}\{A\}$ *denotes the orthogonal complement of the vector space generated by the columns of* A.

$\mathcal{N}\{A\} = span^\perp_{row}\{A\}$ *denotes the orthogonal complement of the vector space generated by the rows of* A *(null space).*

$\mathcal{N}^\perp\{A\} = span_{row}\{A\}$ *denotes the vector space generated by the rows of* A.

$r = rank\{A\}$ *denotes the algebraic rank of* A.

The $n \times n$ *identity matrix is denoted by* $I_{n\times n}$ *or sometimes also* I_n *for short.*

Vectors are denoted by lowercase letters a, x, *etc., with parenthesized time labels in subscript* $x_{[k]}$, *etc.*

$\|X\|_F$ *and* $\|x\|_2$ *denote the Frobenius and 2-norm as follows*

$$\|X\|_F = \sqrt{\sum_i \sum_j x_{ij}^2}$$

$$\|x\|_2 = \sqrt{\sum_i x_i^2}$$

The QR *factorization of a matrix* A *is denoted by*

$$A = Q \cdot \begin{bmatrix} R \\ 0 \end{bmatrix}$$

see page 17.

The singular value decomposition a matrix A *is denoted by*

$$A = U \cdot \Sigma \cdot V^T$$

see page 24.

$G\{i,j,\theta\}$ *denotes a Givens transformation, as defined on page 20.*

Nederlandse samenvatting

Lineaire algebra en matrixrekenen spelen in toenemende mate een belangrijke rol in de moderne systeemtheorie en signaalverwerking. Matrixtechnieken worden tegenwoordig gebruikt voor spectraalanalyse, beeldverwerking, systeemidentificatie, regeltechniek, enz. Meestal zijn deze geavanceerde technieken echter bijzonder rekenintensief, wat bijvoorbeeld bij reële tijd-verwerking een niet te onderschatten probleem vormt. In dit proefschrift worden snelle algoritmen ontwikkeld voor zulke veeleisende reële tijd-toepassingen.

Voor vele ingenieurstoepassingen laat de lineaire algebra niet alleen toe het probleem op een elegante manier te formuleren, maar ook wordt direct een geschikte numeriek betrouwbare oplossingsmethode geserveerd. Naarmate de mathematische problemen complexer worden, worden inderdaad ook numerieke aspecten belangrijker, zoals foutenvoortplanting, gevoeligheid, enz. Het belang van de numerieke lineaire algebra wordt bevestigd door het veelvuldig gebruik van orthogonale technieken zoals QR-factorisaties en (vooral) singuliere waardenontbinding [Golub & Van Loan 1983, 1988]. Voor frequentieanalyse bijvoorbeeld zijn vele klassieke methoden gebaseerd op eigenwaarden/eigenvector-ontbinding [Pisarenko 1973]. In de praktijk echter, wanneer men rekening moet houden met meetruis, enz., is het voor het berekenen van die eigenwaarden/eigenvectoren aangewezen gebruik te maken van een singuliere waardenontbinding [Tufts & Kumaresan 1982, Tufts et al. 1982, Kung et al 1983]. Orthogonale matrixfactorisaties, QR-factorisatie en vooral singuliere waardenontbinding, spelen daarom een centrale rol in dit proefschrift. Voor praktische reële tijd-toepassingen werden deze technieken echter tot dusver nauwelijks gebruikt, hetgeen voornamelijk te maken heeft met de hoge berekeningscomplexiteit. Adaptieve singuliere waardenontbinding of SWO-updating voor reële tijd-toepassingen is daarom één van de centrale thema's in dit proefschrift.

De strategie voor het opstellen van een adaptief SWO-algoritme, bestaat erin verschillende algoritmen te combineren (zie verder). In het algemeen kan een brede waaier aan gecombineerde algoritmen samengesteld worden, door telkens gebruik te maken van een beperkte verzameling elementaire matrixalgoritmen, meestal Jacobi-algoritmen. Deze gecombineerde algoritmen vormen een tweede thema in dit proefschrift. Zulke algoritmen worden afgeleid voor Kalman-filteren, voor recursieve kleinste kwadraten-berekeningen, en voor het oplossen van Riccati- en Lyapunov-vergelijkingen.

Tenslotte, voor toepassingen met hoge doorvoersnelheden, zoals bij radar en beeldverwerking, moet men kunnen beschikken over parallelle algoritmen en architecturen. Systolische roosters bieden dan bijvoorbeeld de mogelijkheid om door het gebruik van massaal parallellisme, spectaculair hogere performanties te bereiken, consistent met de vereisten voor reële tijd-verwerking. Hoewel praktische parallelle algoritmen gekend zijn voor de meeste elementaire matrixoperaties, vormt het implementeren van gecombineerde algoritmen veelal een probleem. Systolische roosters vormen daarom een derde thema in dit proefschrift. Voor elk van de hogergenoemde gecombineerde algoritmen wordt een parallelle implementatie ontwikkeld.

Algemeen overzicht

De drie centrale thema's in dit proefschrift zijn dus i) gecombineerde Jacobi-algoritmen voor adaptieve singuliere waardenontbinding en ii) voor een aantal belangrijke gelijkaardige updatingproblemen, en tenslotte iii) de parallelle implementatie van deze gecombineerde algoritmen. We geven een kort overzicht voor elk van deze thema's.

Adaptieve singuliere waardenontbinding

Voor vele signaalverwerkingstoepassingen is het nodig bepaalde matrixontbindingen continu bij te stellen, naarmate nieuwe observaties toegevoegd worden. We denken aan radartoepassingen, adaptieve systeemidentificatie, spectraalanalyse, enz. Efficiënte updatingtechnieken zijn bijvoorbeeld gekend voor QR-factorisatie [Gill et al. 1974], en hiervoor werd ook een systolische implementatie ontwikkeld [Gentleman & Kung 1981]. Het gelijkaardige SWO-updatingprobleem, dat minstens even belangrijk is maar tegelijkertijd ook veel moeilijker, werd pas zeer recentelijk beschreven. Voor een overzicht van gebruikelijke oplossingsmethoden verwijzen we naar [Comon & Golub 1989].

Bij de meeste van deze methoden wordt het SWO-updatingprobleen opgelost via het overeenkomstig 'rang-1'-gewijzigd symmetrisch eigenwaardenprobleem [Bunch et al. 1978, Bunch & Nielsen 1978, Golub 1973]. In elke tijdstap wordt een exacte decompositie berekend, vertrekkend van een exacte decompositie van de vorige tijdstap. De berekeningscomplexiteit is dan $O\{n^3\}$, waarbij n de dimensie van het probleem voorstelt. Voor vele praktische toepassingen vormt deze berekeningscomplexiteit dikwijls een onoverkomelijk probleem. Deelruimtemethoden (E: Subspace methods) werden ontwikkeld om hieraan te verhelpen, bijvoorbeeld door de kleinste (ruis-) singuliere waarden in elke tijdstap uit te middelen [Karasalo 1986, Schreiber 1986, DeGroat & Roberts 1988, 1990]. De berekeningscomplexiteit is dan $O\{nd^2\}$, waarbij d de dimensie is van de 'signaalruimte', of nog het aantal dominante singuliere waarden. Anderzijds wordt echter door het uitmiddelen in elke stap telkens een supplementaire fout ingevoerd.

Bij wijze van alternatief hiervoor, wordt in dit proefschrift een **Jacobi-methode** *ontwikkeld voor SWO-updating, een combinatie van twee gekende algoritmen, enerzijds een Jacobi-methode voor SWO [Kogbetliantz 1954, 1955, Luk 1986a], anderzijds het QR-updatingalgoritme [Gill et al. 1974]. Het QR-updatingalgoritme zorgt er dan voor dat telkens een triangulaire factor van de observatiematrix beschikbaar is. De SWO-procedure reduceert de triangulaire factor dan verder tot een diagonaalmatrix, met de singuliere waarden op de hoofddiagonaal. Het SWO algoritme is noodzakelijk iteratief, maar het blijkt nu dat nauwkeurige benaderingen kunnen berekend worden ook wanneer in elke tijdstap slechts een beperkt aantal iteraties uitgevoerd wordt, hetgeen in een aanzienlijke besparing op de hoeveelheid rekenwerk resulteert. Deze updatingprocedure is niet alleen goedkoop in aantal bewerkingen –$O\{n^2\}$, zoals gewone QR-updating– maar ook nauwkeurig (voor traag tijdsvariante systemen) en bovendien uitermate geschikt voor parallelle implementatie. Tenslotte is deze methode ook uitbreidbaar voor veralgemeende SWO-problemen, zoals product- en quotiënt-SWO. Zulke problemen bleven tot dusver onbehandeld in de literatuur.*

De theoretische resultaten in dit proefschrift worden geïllustreerd aan de hand van een praktische toepassing uit de systeemtheorie, met name (adaptieve) **systeemidentificatie.** *Allereerst wordt een op SWO gebaseerde methode beschreven voor het identificeren van toestandsruimtemodellen, vertrekkend van in- en uitgangsmetingen. Voor het geval waarbij de in-uitgangsgegevens verstoord zijn met gekleurde ruis, wordt ook een procedure ontwikkeld die gebaseerd is op quotiënt-SWO. Deze beide methoden worden tenslotte omgevormd tot adaptieve identificatiemethoden voor tijdsvariante systemen, door gebruik te maken van de ontwikkelde updatingtechnieken voor SWO en quotiënt-SWO. Enkele geconstrueerde voorbeelden en een levensecht voorbeeld met een flexibele robotarm illustreren de theorie.*

Gecombineerde algoritmen

Het SWO-updatingprobleem is op zich bijzonder belangrijk, maar de voorgestelde oplossingsstrategie blijkt ook voor andere updatingproblemen uitermate bruikbaar, met ruime toepassingsmogelijkheden in de signaalverwerking en systeemtheorie. Het sleutelconcept is dan het **Jacobi-algoritme.** *Voor de meeste matrixdecomposities zijn Jacobi-algoritmen beschikbaar, of kunnen bestaande procedures vertaald worden als Jacobi-algoritmen, die dan telkens gebaseerd zijn op het herhaaldelijk oplossen van kleine '2-bij-2' problemen. Naast het SWO-algoritme [Luk 1986a], werden bijvoorbeeld ook Jacobi-algoritmen ontwikkeld voor het symmetrich eigenwaardenprobleem [Brent & Luk 1985], voor de Schur- [Stewart 1985] of veralgemeende Schurdecompositie [Charlier & Van Dooren 1989a], voor de quotiënt-SWO [Luk 1985, Paige 1986] en de product-SWO [Heath et al. 1986], voor de QR-factorisatie [Luk 1986b], de LDU- en Cholesky-decompositie [Moonen et al. 1990e], voor matrixinversie (hoofdstuk 5), enz. Het grote voordeel bij deze methoden is de eenvoud, wat resulteert in een elegante implementatie op parallelle architecturen.*

Vele praktische algoritmen voor regeltechniek, filteren, enz., zijn essentieel samengesteld als een combinatie van elementaire matrixbewerkingen, en voor de meeste van die bewerkingen zijn dus Jacobi-algoritmen beschikbaar. De SWO-updatingprocedure is hiervan één voorbeeld. Meer algemeen kunnen elementaire Jacobi-algoritmen samengebracht worden in gecombineerde Jacobi-algoritmen. De regelmatige structuur van zulke gecombineerde algoritmen maakt hen uitermate geschikt voor parallelle implementatie. In dit proefschrift worden gecombineerde Jacobi-algoritmen ontwikkeld voor Kalman-filteren, voor recursieve kleinste kwadraten-berekeningen, en voor het oplossen van Riccati- en Lyapunov-vergelijkingen.

Systolische roosters

Systolische rooster zijn netwerken van processoren. Naar analogie met de wijze waarop een hart bloed door het lichaam pompt (vandaar de naam), wordt hier een gegevensstroom door het netwerk gepompt, waarbij elke processor elementaire bewerkingen uitvoert op ontvangen gegevens, alvorens die door te sturen naar de volgende processor [Kung & Leiserson 1980]. Het massaal parallellisme biedt hier de mogelijkheid tot het bereiken van extreem hoge doorvoersnelheden voor reële tijd-toepassingen. Systolische roosters werden tot heden voornamelijk gebruikt voor redelijk eenvoudige operaties (convoluties, enz.). Het ontwikkelen van systolische roosters voor complexe gecombineerde algoritmen is betrekkelijk nieuw. Meestal worden verschillende elementaire matrixbewerkingen dan afzonderlijk geparallelliseerd. De communicatie tussen de verschillende bouwblokken resulteert daarbij in wachttijden en een inefficiënt processorgebruik.

Elementaire Jacobi-algoritmen kunnen probleemloos op systolische roosters geïmplementeerd worden. Hetzelfde geldt dan ook voor de gecombineerde Jacobi-algoritmen. Soms (meestal) moeten echter nog een aantal bijkomende operaties geïmplementeerd worden, zoals matrix-vectorvermenigvuldigingen in de SWO-updatingprocedure, terugsubstituties, enz. In dit proefschrift wordt een methode ontwikkeld voor het gecombineerd implementeren van deze operaties met de Jacobi-procedures. De verschillende operaties corresponderen dan met verschillende 'berekeningsgolven', en telkens wanneer twee golven elkaar kruisen worden enkele bewerkingen toegevoegd voor het bijstellen van intermediaire resultaten. Het aantal bijkomende bewerkingen is echter uitermate beperkt, zodat de efficiëntie niet in het gedrang komt. Het beschikbare parallellisme wordt volledig benut, wat dan resulteert in de hoogst mogelijke doorvoersnelheid. De op die manier ontwikkelde roosters zijn meestel veel efficiënter dan gekende implementaties.

In wat volgt worden de belangrijkste resultaten van elk hoofdstuk samengevat.

Hoofdstuk 1: Achtergrond en motivatie

Het eerste hoofdstuk is inleidend en bevat het nodige 'achtergrondmateriaal'. In het kort wordt de wiskundige theorie voor zowel QR**-factorisatie** *als* **singuliere waardenontbinding** *in herinnering gebracht.*

Door een QR-factorisatie wordt een $m \times n$ -matrix A ($m \geq n$) ontbonden in een product van een orthogonale matrix Q en een triangulaire factor R

$$\underbrace{A}_{m\times n} = \underbrace{Q}_{m\times m} \cdot \begin{bmatrix} R \\ 0 \end{bmatrix}.$$

Een singuliere waardenontbinding is een ontbinding in een product van orthogonale matrices U en V en een diagonaalmatrix Σ

$$\underbrace{A}_{m\times n} = \underbrace{U}_{m\times m} \cdot \underbrace{\Sigma}_{m\times n} \cdot \underbrace{V^T}_{n\times n}.$$

Het belang van deze matrixontbindingen voor ingenieurstoepassingen wordt geïllustreerd aan de hand van enkele theorema's en praktische voorbeelden, en in het bijzonder in het licht van de SWO-gebaseerde identificatieprocedure.

Systeemidentificatie *beoogt het opstellen van mathematische modellen, vertrekkend van alleen maar in- en uitgangsmetingen van een proces. Een toestandsruimtemodel van de vorm*

$$\begin{aligned} x_{[k+1]} &= \underbrace{A}_{n\times n} \cdot x_{[k]} + \underbrace{B}_{n\times m} \cdot u_{[k]} \\ y_{[k]} &= \underbrace{C}_{l\times n} \cdot x_{[k]} + \underbrace{D}_{l\times m} \cdot u_{[k]} \end{aligned}$$

(A, B, C en D zijn de systeemmatrices, $u_{[k]}$, $y_{[k]}$ en $x_{[k]}$ zijn respectievelijk ingangs-, uitgangs- en inwendige toestandsvector op tijdstip k) is daarbij een geschikte modelkeuze, maar spijtig genoeg zijn er niet veel (betrouwbare) toestandsruimte-identificatietechnieken in de literatuur beschikbaar. In [Moonen et al. 1989a] werd zulke identificatietechniek beschreven, en dit wordt ook bondig herhaald in dit hoofdstuk. Een toestandsruimtemodel wordt berekend via de singuliere waardenontbinding van een blok-Hankelmatrix met in- en uitgangsgegevens. Het voorgestelde algoritme is daarom vergelijkbaar met gekende realisatiealgoritmen in [Ho en Kalman 1966, Zeiger en Mc Ewen 1974] en vooral [Kung 1978], waar gebruik gemaakt wordt van blok-Hankelmatrices met Markov-parameters (impulsresponsie).

Die Markovparameters zijn echter meestal moeilijk meetbaar (berekenbaar), en bovendien leveren de realisatiealgoritmen zich ook nauwelijks tot adaptieve implementatie. Het ontwikkelde identificatiealgoritme kan daarentegen zeer eenvoudig omgewerkt worden tot een adaptieve procedure, door gebruik te maken van adaptieve SWO-schema's. Tenslotte kan een efficiënte ruiscompensatie bekomen worden door gebruik te maken van quotiënt-SWO in plaats van 'gewone' SWO [Moonen & Vandewalle 1990b]. Bij wijze van voorbeeld wordt een identificatie beschreven van een flexibele robotarm (Figuur 1.6), waarbij de overdracht van ingang (stuurstroom) naar uitgang (hoeksnelheid) beschreven wordt met een (vijfde orde) toestandsruimtemodel. Figuur 1.9 toont de resulterende overdrachtsfunctie.

Tenslotte worden in dit hoofdstuk ook enkele **berekeningsalgoritmen** *besproken voor hogergenoemde matrixfactorisaties, met het oog op de afleidingen in verdere hoofdstukken. Het berekenen van een QR-factorisatie kan gebeuren door herhaaldelijk gebruik te maken van Givens transformaties, waarbij de matrixgetallen onder de hoofddiagonaal één na één geannihileerd worden. Analoog kan men ook direct een methode afleiden voor QR-updating [Gill et al. 1974], waarbij de matrix A recursief gedefinieerd is en waarbij op elk tijdstip de triangulaire factor $R_{[k]}$ herberekend wordt uit $R_{[k-1]}$*

$$A_{[k]} = \begin{bmatrix} \lambda_{[k]} \cdot A_{[k-1]} \\ a_{[k]}^T \end{bmatrix} = \underbrace{Q_{[k]}}_{k \times n} \cdot \underbrace{R_{[k]}}_{n \times n}.$$

Hierin is $\lambda_{[k]}$ $(0 < \lambda_{[k]} \leq 1)$ een 'weegfactor' en is $a_{[k]}$ de observatie op tijdstip k.

Voor de berekening van de singuliere waardenontbinding, wordt verder in dit hoofdstuk het **Kogbetliantz-algoritme** *besproken [Kogbetliantz 1955]. Voor sequentiële machines is dit zeker niet het meest efficiënte SWO-algoritme. Veeleer zal men dan gebruik maken van het standaardalgoritme van Golub, Reinsch en Kahan [Golub & Kahan 1965, Golub & Reinsch 1970], waarvoor ook kwaliteitssoftware beschikbaar is. Nochtans geniet de Kogbetliantz-methode een hernieuwde interesse, en dit in verband met de beschikbaarheid van parallelle computers. Deze methode leent zich inderdaad wonderwel voor parallelle implementatie, en ook voor de updatingproblemen in hoofdstuk 2 biedt ze een geschikte strategie. Het Kogbetliantz-algoritme is een uitbreiding voor niet-symmetrische matrices van het originele* **Jacobi-algoritme** *(symmetrisch eigenwaardenprobleem). Jacobi-methoden spelen*

zoals gezegd een belangrijke rol in dit proefschrift. In verdere hoofdstukken zullen dan ook gelijkaardige algoritmen voorgesteld worden, telkens wanneer de gelegenheid zich voordoet, bijvoorbeeld voor product- en quotiënt-SWO, voor matrixinversie, enz.

Hoofdstuk 2: SWO-updating - algoritmen en analyse

In hoofdstuk 1 werd een methode beschreven voor QR-updating wanneer in iedere tijdstap een nieuwe observatie wordt toegevoegd. Elke update vraagt slechts $O\{n^2\}$ bewerkingen, terwijl een volledige QR-factorisatie normaal $O\{n^3\}$ bewerkingen vraagt. QR-updating is niet alleen efficiënt, maar ook numeriek stabiel [Gentleman 1975] en bovendien bijzonder geschikt voor parallelle implementatie [Gentleman & Kung 1981]. QR-updating is daarom een praktisch werktuig voor tal van ingenieurstoepassingen.

Het **SWO-updatingprobleem** *is wellicht nog belangrijker, maar kennelijk veel moeilijker oplosbaar. Singuliere waardenontbinding wordt in de praktijk dan ook nauwelijks gebruikt, door het gebrek aan efficiënte adaptieve algoritmen. In dit hoofdstuk wordt een adaptieve techniek ontwikkeld voor het SWO-updatingprobleem, waarbij een matrix A weer recursief gedefinieerd is*

$$A_{[k]} = \begin{bmatrix} \lambda_{[k]} \cdot A_{[k-1]} \\ a_{[k]}^T \end{bmatrix} = \underbrace{U_{[k]}}_{k \times n} \cdot \underbrace{\Sigma_{[k]}}_{n \times n} \cdot \underbrace{V_{[k]}^T}_{n \times n}.$$

Het QR-updatingalgoritme wordt daartoe gecombineerd met de SWO- procedure, die de triangulaire factor telkens herleidt tot een diagonale matrix. Wanneer een exacte decompositie berekend wordt in elke tijdstap, is de berekeningscomplexiteit noodzakelijk $O\{n^3\}$. Nochtans volstaat het dikwijls te kunnen beschikken over een benaderende decompositie, waarbij dan slechts enkele iteratiestappen in de iteratieve SWO-procedure dienen uitgevoerd binnen elke tijdstap. Op die manier kan de berekeningscomplexiteit tot $O\{n^2\}$ herleid worden, vergelijkbaar dus met de berekeningscomplexiteit van alleen maar QR-updating.

Belangrijk is dan te kunnen beschikken over een schatting voor de **nauwkeurigheid** *van de berekende decompositie in een willekeurige tijdstap. Het adaptief algoritm werd daartoe geanalyseerd voor deelruimteproblemen (E:*

subspace tracking). *In het identificatiealgoritme bijvoorbeeld wordt de SWO gebruikt voor het scheiden van een 'signaalruimte' en een 'ruisruimte'. Hetzelfde geldt voor toepassingen van adaptieve antennes, spectraalanalyse, MUSIC, ESPRIT [Paulraj et al. 1986], enz. Deze deelruimtes komen overeen met deelmatrices in $V_{[k]}$, en voor verdere berekeningen is dan alleen die informatie vereist. Er wordt dan aangetoond dat wanneer gebruik gemaakt wordt van een $O\{n^2\}$ adaptief SWO-algoritme, de benaderingsfout voor de deelruimtes kleiner is dan de variatie van diezelfde deelruimtes in n tijdstappen. Voor traagvariërende systemen kan zulks volstaan. Enkele voorbeelden zijn toegevoegd om dit te illustreren (flexibele robotarm), waarbij telkens een adaptieve systeemidentificatie uitgevoerd wordt, gebaseerd op adaptieve singuliere waardenontbinding.*

Tenslotte wordt ook een gedetailleerde foutenanalyse uitgevoerd, die aantoont dat de ontwikkelde SWO-updatingtechniek numeriek stabiel is, op voorwaarde dat de $V_{[k]}$-matrix benaderend orthogonaal blijft. Daartoe wordt dan een re-orthogonalisatieprocedure ontwikkeld, die eenvoudig te combineren valt met het SWO-updatingschema, en die geen beduidende verhoging van de vereiste rekentijd meebrengt.

Hoofdstuk 2 is gebaseerd op [Moonen et al. 1989b]. Het SWO-updatingalgoritme en de re-orthogonalisatieprocedure, zowel als de nauwkeurigheids- en foutenanalyse zijn nieuw. Tijdens de voorbereiding van de tekst vonden we gelijkaardige –maar minder volledige– resultaten in [Ferzali & Proakis 1990], waar ook naar ons werk gerefereerd wordt.

Hoofdstuk 3: Systolische roosters voor SWO-updating

In hoofdstuk 2 werd een SWO-updatingprocedure ontwikkeld, door enkele elementaire matrixalgoritmen te combineren. QR-factorisatie wordt gecombineerd met een SWO-algoritme, en ook zijn nog een aantal matrix-vectorvermenigvuldigingen vereist. Elke update vraagt $O\{n^2\}$ operaties, hetgeen bijvoorbeeld voor radar of beeldverwerkingstoepassingen –gezien de hoge bemonsteringsfrequenties– nog steeds een bijzonder probleem vormt. Om aan de extreem hoge eisen van reële tijd-verwerking te voldoen, zal men dan ook dikwijls gebruik moeten maken van parallelle rekenstructuren. Voor het QR-updatingprobleem werd een efficiënte parallelle rekenstructuur ontwikkeld [Gentleman & Kung 1981]. Met $O\{n^2\}$ processoren wordt dan een

doorvoersnelheid bereikt die onafhankelijk is van de systeemorde, i.e. $O\{n^0\}$.

In dit hoofdstuk wordt aangetoond hoe de **SWO-updatingprocedure** *op een gelijkaardig rooster kan geïmplementeerd worden, met eveneens $O\{n^2\}$ processoren. Wanneer echter de afzonderlijke bewerkingen in deze procedure als dusdanig geïmplementeerd worden, valt de doorvoersnelheid terug tot $O\{n^{-1}\}$, door de aanwezigheid van een 'kritisch pad'. Daarom wordt een techniek ontwikkeld die toelaat de verschillende operaties toch te combineren op eenzelfde rooster, ten koste van enkele supplementaire bewerkingen. Verschillende operaties corresponderen dan met verschillende 'berekeningsgolven', en telkens wanneer twee golven elkaar kruisen worden enkele bewerkingen toegevoegd voor het bijstellen van intermediaire resultaten. Op die manier kan het beschikbare parallellisme ten volle benut worden, met andere woorden kan een $O\{n^0\}$ doorvoersnelheid bekomen worden, zoals het geval was bij QR-updating. Het resulterende rooster is op het eerste gezicht gecompliceerd, maar toch perfect modulair en repetitief. Een schematisch overzicht wordt gegeven in Figuur 3.6. De correctheid werd geverifieerd aan de hand van computersimulaties.*

In verband met praktische uitvoeringen van deze systolische roosters, ligt het voor de hand gebruik te maken van orthogonaal geconnecteerde netwerken [Stewart 1985, O'Leary & Stewart 1985] (Figuur 3.3). Nochtans wordt in dit hoofdstuk ook een alternatieve partitionering voorgesteld, resulterend in een **hexagonaal geconnecteerd rooster** *(Figuur 3.4-3.5). Er wordt aangetoond dat deze partitionering meer dan vier keer efficiënter is dan de originele configuratie.*

Tenslotte wordt in dit hoofdstuk ook aangetoond hoe het berekenen van vierkantswortels in het SWO-updatingalgoritme kan vermeden worden. **Wortelberekeningen** *vormen inderdaad een niet-triviale aangelegenheid voor bijvoorbeeld elementaire signaalprocessoren, en de werking van een systolisch rooster kan daardoor aanzienlijk vertraagd worden. Analoge technieken voor QR-updating zijn voldoende bekend [Gentleman 1973, Hammarling 1974], waarbij gebruik gemaakt wordt van rij-scaleringen van de triangulaire factor. Voor het SWO-updatingprobleem kan op dezelfde manier gebruik gemaakt worden van zowel rij- als kolom-scaleringen, en dit wordt uitvoerig beschreven.*

Hoofdstuk 3 is gebaseerd op [Moonen et al. 1989c] en [Moonen & Van-

dewalle 1990a]. De systolische implementatie, de hexagonale partitionering en tenslotte de wortelvrije algoritmen zijn nieuw.

Hoofdstuk 4: Veralgemeende SWO-updating

In dit hoofdstuk wordt de SWO-updatingprocedure veralgemeend voor veralgemeende SWO's, met name de **quotiënt-SWO** *(QSWO) [Van Loan 1976, Paige & Saunders 1981] en de* **product-SWO** *(PSWO) [Fernando & Hammarling 1987, De Moor 1989]. De nadruk ligt hier op algoritmen en de manier waarop ze parallel kunnen geïmplementeerd worden. Nauwkeurigheids- en foutenanalyse werden achterwege gelaten.*

QSWO en PSWO zijn matrixfactorisaties voor matrixparen (A, B). In het kort kunnen QSWO en PSWO als volgt beschreven worden

QSWO

$$\begin{aligned} \underbrace{A}_{m\times n} &= \underbrace{U_A}_{m\times m} \cdot \underbrace{\Sigma_A}_{m\times n} \cdot \underbrace{X^{-1}}_{n\times n} \\ \underbrace{B}_{p\times n} &= \underbrace{U_B}_{p\times p} \cdot \underbrace{\Sigma_B}_{p\times n} \cdot \underbrace{X^{-1}}_{n\times n} \end{aligned}$$

PSWO

$$\begin{aligned} \underbrace{A}_{m\times n} &= \underbrace{U_A}_{m\times m} \cdot \underbrace{\Sigma_A}_{m\times n} \cdot \underbrace{X^{T}}_{n\times n} \\ \underbrace{B}_{p\times n} &= \underbrace{U_B}_{p\times p} \cdot \underbrace{\Sigma_B}_{p\times n} \cdot \underbrace{X^{-1}}_{n\times n} \end{aligned}$$

waarbij U_A en U_B orthogonaal zijn, en X niet-singulier. Om het belang van deze technieken in ingenieurstoepassingen aan te tonen, wordt eerst ingegaan op één belangrijke toepassing van QSWO, met name **QSWO-gebaseerde systeemidentificatie**. *De identificatieprocedure van hoofdstuk 1 (gebaseerd op 'gewone' SWO), wordt hier geherformuleerd in het QSWO-raamwerk, waarbij dan ruiscovariantiematrices van de in- en uitgangsmetingen expliciet in rekening kunnen gebracht worden. Matrix A bevat dan de meetgegevens, B bevat informatie over de ruiskleuring. In de praktijk zal bijvoorbeeld het gebruik van filters aanleiding geven tot ruiskleuring (Figuur 4.1), en zal het inderdaad nodig zijn daarmee ook rekening*

te houden. Het QSWO-gebaseerd algoritme geeft dan consistente resulaten, in tegenstelling tot het originele SWO-algoritme. Tenslotte, voor adaptieve identificatie van tijdsvariante systemen, kan het QSWO-identificatiealgoritme weer omgewerkt worden in een adaptief schema, waarbij dan gebruik dient gemaakt van QSWO-updating.

Bij de P- en QSWO-updatingproblemen in dit hoofdstuk, worden in elke tijdstap observaties toegevoegd aan ofwel één ofwel beide matrices, die dan weer recursief gedefinieerd zijn

$$\begin{aligned} A_{[k]} &= \begin{bmatrix} \lambda \cdot A_{[k-1]} \\ a_{[k]}^T \end{bmatrix} \\ B_{[k]} &= \begin{bmatrix} \lambda \cdot B_{[k-1]} \\ b_{[k]}^T \end{bmatrix}. \end{aligned}$$

Zulke problemen bleven tot dusver onbehandeld in de literatuur.

Voor het **PSWO-updating***probleem wordt gebruik gemaakt van het PSWO-algoritme in [Heath et al. 1986], alweer een Jacobi-algoritme. Voor het* **QSWO-updating***probleem wordt gebruik gemaakt van een gelijkaardig QSWO-algoritme in [Paige 1986]. Updatingschema's worden bekomen door de respectievelijke Jacobi-algoritmen te combineren met QR-updating (QL-updating) en de nodige matrix-vectorvermenigvuldigingen toe te voegen.* **Systolische implementaties** *worden op dezelfde manier bekomen als voor het SWO-updatingschema. Schematische voorstellingen worden getoond in Figuren 4.6 en 4.8 voor QSWO en PSWO respectievelijk. Tenslotte wordt ook beschreven hoe door het gebruik van rij- en kolomscaleringen weer alle* **wortelberekeningen** *kunnen vermeden worden.*

Hoofdstuk 4 is gebaseerd op [Moonen et al. 1989d] en [Moonen & Vandewalle 1990b]. De voorgestelde algoritmen en hun parallelle implementatie zijn nieuw.

Hoofdstuk 5: Jacobi-algoritmen voor signaalverwerking, systeemtheorie en regeltechiek

Het principe van verschillende Jacobi-algoritmen te combineren tot complexere schema's, zoals we dat toegepast hebben op het (veralgemeend)

SWO-updatingprobleem[1], *blijkt nu ook bruikbaar voor tal van andere belangrijke updatingproblemen in signaalverwerking en systeemtheorie. In dit hoofdstuk worden drie specifieke voorbeelden behandeld. Telkens worden nieuwe of meer efficiënte algoritmen en/of parallelle implementaties verkregen.*

Riccati- en Lyapunov-vergelijkingen

Als A, B, V en W (V en W symmetrisch) gegeven matrices zijn, wordt volgende vergelijking met onbekende X de algebraische Riccati-vergelijking genoemd

$$XA + A^T X + XBW^{-1}B^T X + V = 0.$$

Wanneer $B = 0$, herleidt dit zich tot de (lineaire) Lyapunov-vergelijking. Riccati- en Lyapunov-vergelijkingen spelen een centrale rol in tal van problemen in regeltechniek en filteren. Het numeriek oplossen erven kreeg dan ook de nodige aandacht, zie bijvoorbeeld [Bartels & Stewart 1972, Byers 1987, Gardiner & Laub 1986, Golub et al. 1979, Hammarling 1982, Laub 1979, Van Dooren 1981]. Een recent algoritme, gebaseerd op 'matrix-sign'-recursies, is bijzonder geschikt voor parallelle implementatie. Een gefactoriseerde versie hiervan werd ontwikkeld in [Charlier & Van Dooren 1989b], waar ook een parallelle rekenstructuur hiervoor beschreven wordt. De basisoperaties die hieraan te pas komen zijn matrixinversies en triangulaire factorisaties.

In dit hoofdstuk wordt eerst een Jacobi-algoritme voor matrixinversie afgeleid, en daarna aangetoond hoe dit kan gecombineerd worden met de triangulaire factorisaties. De resulterende parallelle implementatie is dan analoog aan de eerder ontwikkelde rekenstructuren voor SWO-updating, en is ongeveer drie keer efficiënter dan de implementatie beschreven in [Charlier & Van Dooren 1989b].

Recursieve kleinste kwadraten-berekeningen

Het kleinste kwadraten-probleem dat hier beschouwd wordt, bestaat erin de beste oplossing te berekenen –volgens een kleinste kwadratencriterium– voor een overgedetermineerd stelsel lineaire vergelijkingen $A \cdot x = b$

$$x_{KK} = \min_x \|A \cdot x - b\|_2.$$

[1] *QR*-updating is nl. in zekere zin even goed een Jacobi-algoritme

In het recursieve probleem zijn A en b recursief gedefinieerd

$$A_{[k]} = \begin{bmatrix} \lambda \cdot A_{[k-1]} \\ a_{[k]}^T \end{bmatrix} \qquad b_{[k]} = \begin{bmatrix} \lambda \cdot b_{[k-1]} \\ \beta_{[k]} \end{bmatrix}$$

en wordt $x_{KK_{[k]}}$ in elke tijdstap herberekend. Hierbij kan gebruik gemaakt worden van QR-updating, waarbij dan de kleinste kwadratenoplossing telkens bekomen wordt door oplossen van een vierkant stelsel

$$R_{[k]} \cdot x_{KK_{[k]}} = y_{[k]}.$$

In elke tijdstap wordt dus een QR-update uitgevoerd, gevolgd door een terugsubstitutie. Probleem hierbij is dat die twee stappen niet tesamen kunnen geïmplementeerd worden op bijvoorbeeld een systolisch rooster. In het QR-updatingrooster [Gentleman & Kung 1981] loopt een QR-update van boven naar beneden (Figuur 3.1), terwijl een terugsubstitutie noodzakelijk in de omgekeerde richting loopt. Dit kritisch pad haalt de doorvoersnelheid omlaag tot $O\{n^{-1}\}$ (in plaats van $O\{n^0\}$). Efficiënte parallelle implementatie van het recursieve KK-schema bleef dus een onopgelost probleem.

Zeer recent werden enkele nieuwe algoritmen ontwikkeld die toelaten hieraan te verhelpen. Allereerst zijn er de algoritmen die gebruik maken van **inverse factorisatie**, *waarbij naast $R_{[k]}$ ook telkens $R_{[k]}^{-1}$ herberekend wordt [Pan & Plemmons 1989, Shepherd & McWhirter 1990]. Dit resulteert in een parallelle implementatie met $O\{n^0\}$ doorvoersnelheid, weliswaar ten koste van een verdubbeling van de nodige hardware [Shepherd & McWhirter 1990] (Figuur 5.1). In dit hoofdstuk wordt echter aangetoond dat deze procedure onstabiel is en m.a.w. divergeert. Stabiliteit kan dan bekomen worden door het invoeren van een correctieschema, gebaseerd op een Jacobi-algoritme voor matrixinversie. Het resulterende recursieve KK-schema is dan analoog aan het SWO-updatingschema, met nu Jacobi-inversie in plaats van Jacobi-SWO. Bijgevolg kan ook weer een parallelle implementatie afgeleid worden, analoog aan de implemenatie voor SWO-updating. Een schematisch overzicht is weergegeven in Figuur 5.3.*

Het hogergenoemde recursieve KK-schema is weliswaar stabiel en parallel implementeerbaar, maar weinig efficiënt. Daarom werd een tweede

parallelle implementatie ontwikkeld, gebaseerd op het **orthogonaal invers-updating**algoritme *in [Pan & Plemmons 1989]. Daarbij wordt alleen $R_{[k]}^{-1}$ berekend, hetgeen resulteert in een veel efficiënter schema, maar waarvoor blijkbaar geen parallelle implementatie beschikbaar was. In dit hoofdstuk wordt aangetoond hoe –weer door gebruik te maken van kruisende berekeningsgolven in een Jacobi-partitionering– toch een efficiënte implementatie kan bekomen worden. Een schematisch overzicht is weergegeven in Figuur 5.5.*

Kalman-filteren

Het Kalman-filter is een 'toestandsschatter', die de beste (kleinste variantie) schatting geeft van de inwendige toestand in een toestandsruimtemodel, uitgaande van in- en uitgangsobservaties en de gekende systeemmatrices en ruiscovarianties. Recentelijk werden enkele parallelle implementaties beschreven, die het Kalman-filter ook bruikbaar moeten maken voor snelle reële tijd-toepassingen. Meestal wordt daarbij gebruik gemaakt van 'square root'-algoritmen, die het probleem herleiden tot een gestructureerd QR-updatingprobleem. De kunst bestaat er dan in die gegeven matrixstructuur zo efficiënt mogelijk te benutten. Een rechttoe-rechtaan implementatie resulteert bijvoorbeeld in een uitermate inefficiënt processorgebruik (17%). De meest efficiënte structuur is beschreven in [Kung 1988], met een processorgebruik tussen 50% en 60 %. Een QR-updatingrooster wordt aangevuld met een gegevensbuffer. De connecties tussen het triangulair rooster en de gegevensbuffer lopen noodzakelijk rond het triangulair rooster, hetgeen de regulariteit van de structuur verstoort. Een alternatieve en eenvoudigere implementatie in [Gaston & Irwin 1989] heeft een processorgebruik tussen 33% en 37.5%.

In dit hoofdstuk wordt aangetoond hoe door combineren van alweer QR-updating met een Jacobi-algoritme voor deze keer QR-factorisatie, een rekenstructuur bekomen wordt met een processorgebruik tussen 37.5% en 50%. De voorgestelde architectuur vertoont een regelmatige structuur (dezelfde als die van het SWO-updatingrooster), zonder gegevensbuffers enz., en biedt daarom een bruikbaar alternatief voor hardware-implementatie.

Hoofdstuk 5 is gebaseerd op [Moonen & Vandewalle 1989e], [Moonen & Vandewalle 1990c], [Moonen & Vandewalle 1990d] en [Moonen & Vandewalle 1990f]. De voorgestelde algoritmen en parallelle structuren zijn nieuw.

Algemeen besluit

In dit proefschrift ging onze aandacht naar efficiënte algoritmen voor signaalverwerking, systeemidentificatie en regeltechniek. We vertrokken van een acuut probleem, met name de berekeningscomplexiteit van de singuliere waardenontbinding, die tot dusver een aanzienlijke belemmering vormde voor het praktisch gebruik van SWO in reële tijd-toepassingen. Daarom werd eerst een efficiënt **adaptief SWO-algoritme** *ontwikkeld, waarbij de berekeningscomplexiteit van $O\{n^3\}$ herleid werd tot $O\{n^2\}$.Telkens wordt een benaderende ontbinding berekend, met een benaderingsfout die beperkt blijft tot de tijdsvariatie in n tijdstappen. Dit SWO-updatingalgoritme kan dan bijvoorbeeld gebruikt worden voor recursieve systeemidentificatie, zoals aangetoond.*

Daarnaast ging onze aandacht ook naar **parallelle implementatie** *van deze algoritmen, met name op systolische roosters. Een nieuwe procedure werd ontwikkeld die toelaat verschillende matrixoperaties te combineren op één rekenstructuur, waarbij door het invoeren van een aantal bijkomende bewerkingen het beschikbare parallellisme ten volle benut wordt. Concreet betekent dit dat voor het SWO-updatingalgoritme een $O\{n^0\}$ doorvoersnelheid kan bereikt worden, voor $O\{n^2\}$ parallellisme, net zoals dat het geval was voor QR-updating.*

Tenslotte werd aangetoond dat de technieken die we gebruikten voor het construeren van het parallelle algoritme voor SWO-updating, ook bijzonder bruikbaar zijn voor het samenstellen van **updatingalgoritmen** *voor tal van gelijkaardige problemen. Jacobi-algoritmen spelen daarbij een centrale rol. Afgezien van de veralgemeningen voor P- en QSWO, werden zulke parallelle algoritmen afgeleid voor Kalman-filteren, voor recursieve kleinste kwadratenberekeningen, en voor het oplossen van Riccati- en Lyapunov-vergelijkingen.*

Suggesties voor verder onderzoek

Verder onderzoek moet nu in twee richtingen evolueren.

- *Enerzijds in de richting van algoritmen en rekenstructuren voor complexere problemen, bvb. systeemidentificatie, adaptief regelen, enz.,*

door verder exploreren van de ontwikkelde combinatietechnieken. We denken bijvoorbeeld aan H_∞-regeltechniek. Naarmate meer problemen en oplossingen herdacht worden in termen van toestandsruimtemodellen, wordt de vraag naar efficiënte matrixalgoritmen des te groter. Optimisatie –een ander voorbeeld– is hieraan verwant, en geeft eveneens aanleiding tot interessante matrixproblemen. Algoritmisch ontwerp met het oog op parallelle implementatie is hier duidelijk aan de orde, en voor sommige problemen bieden de hier ontwikkelde technieken misschien een uitweg.

– *Anderzijds is ook een verdere ontwikkeling nodig op een lager niveau, waarbij aandacht besteed wordt aan de uitbouw van de architecturen die vereist zijn voor de implementatie van hogergenoemde algoritmen. Rekening houdend met de beperkingen van de hedendaagse VLSI-technologie, is een systolische implementatie van de voorgestelde algoritmen waarschijnlijk nog veraf. Een ASIC-implementatie lijkt al meer haalbaar op korte termijn, maar of het 'fijne korrel'-parallellisme van een systolische implementatie hier ook een efficiënte partitionering uitwijst, is nog een open vraag. Bovendien, zal bij praktische implementaties de voorkeur gegeven worden aan algoritmen met vaste woordlengte. Woordlengte-optimisatie is dus weer aan de orde. Bij het ontwerpen van rekenelementen zal bvb. de keuze van de woordlengte rechtstreeks invloed hebben op de stabiliteit van het globale algoritme. Bij het ontwerp van algoritmen op het hoogste niveau dient dus weldegelijk rekening gehouden met beperkingen op het laagste (processor) niveau. Dit onderzoek zal verder uitgevoerd worden in het kader van het NANA-project –ESPRIT Basic Research Action 3280: Novel Algorithms for new real-time VLSI Architectures. Het gecombineerd ontwerp van algoritmen en architecturen is niet alleen een belangrijk fundamenteel wetenschappelijk onderwerp –getuige hiervan ook de groeiende internationale interesse en wetenschappelijke activiteit in vooraanstaande onderzoekscentra– maar daarbij ook uitgesproken praktijkgericht. Het levert de noodzakelijke softwareproducten en chipimplementaties voor tal van industriële toepassingen van doorgedreven automatisatie, kwaliteitsverbetering, enz., waarbij dan via het efficiënt benutten van massaal parallellisme aan de extreem hoge eisen van reële tijd-verwerking kan voldaan worden.*

Introduction

Linear algebra and matrix computation play a central role in modern systems theory and digital signal processing. Representative areas in which matrix computation is important include beamforming, direction finding, spectrum analysis, image processing, systems identification, robust control, etc. The use of such advanced techniques however involves substantial computational requirements, especially when a real time implementation is aimed at. In this thesis fast matrix algorithms are developed for such demanding real time applications.

For many engineering applications, linear algebra not only allows for an elegant problem statement, but furthermore it provides a numerically sound solution to the problem. As problems have grown in difficulty and complexity, numerical issues such as error propagation and problem sensitivity have grown more and more important. The impact of numerical linear algebra or numerical analysis is apparent through the increased use of orthogonal techniques such as QR factorizations and especially singular value decompositions (SVD's) [Golub & Van Loan 1983, 1988]. As an example, one of the advanced theories of particular interest these days is that of eigenvector processing of time series for high resolution frequency estimation. The theory relies on the estimation of a set of parameters known as eigenvalues and an associated set of eigenvectors [Pisarenko 1973]. On the other hand, when dealing with inexact data and data records of finite duration, it is recommended to use an SVD for actually computing these parameters [Tufts & Kumaresan 1982, Tufts et al. 1982, Kung et al 1983]. More generally, the SVD has become more or less the computational tool for statistics, control engineering, etc., where numerous theoretical concepts have been reformulated in the language of the SVD [Golub & Van Loan 1983, 1988]. Orthogonal matrix factorizations, QR decomposition and especially SVD, therefore play a central role in this thesis. Unfortunately –due to the computational com-

plexity involved– these techniques are hardly employed in practice, e.g. for real time applications. One of our main points of focus is therefore fast adaptive SVD or SVD updating for real time applications.

Our approach to SVD updating mainly consists in combining a Jacobi-type algorithm for the SVD with QR updating, the latter essentially being a Jacobi-type algorithm as well. It appears that in general, a broad range of similar algorithms can be pieced together with a fairly small set of elementary matrix algorithms, primarily Jacobi-type algorithms. These combined Jacobi-type algorithms are a second point of focus. Such algorithms are developed for Kalman filtering, for solving Riccati and Lyapunov equations and for recursive least squares computations.

Finally, for extremely high throughput applications such as image and radar processing, parallel algorithms and architectures are indispensable. Systolic arrays or wavefront arrays then provide a means of exploiting large amounts of parallelism and hence achieving orders of magnitude improvement in performance, consistent with the requirements. While practical parallel algorithms for most of the key matrix operations are by now available, implementing combined algorithms appears to be a major problem. Therefore, we also focus on systolic arrays, and develop such arrays for the above mentioned combined Jacobi-type algorithms.

A general overview

The main points of focus are thus i) Jacobi-type algorithms for SVD updating and ii) for various related updating problems, and iii) parallel (systolic) implementations of these. The following overview is meant to give a flavour of all of these topics.

SVD updating

In many signal processing applications, it is necessary to continuously update matrix decompositions as new observations are continuously added to the problem. Such problems frequently occur in beamforming, direction finding, and spectral analysis. Efficient updating techniques have long been known for the QR-decomposition [Gill et al. 1974] and the systolic implementation of this is well known by now [Gentleman & Kung 1981]. The similar and equally important, but more difficult problem of updating the SVD when new rows are appended, has only recently been addressed, see [Comon & Golub 1989]

for an overview.

Previously described techniques for SVD row updating mostly reduce to solving the corresponding rank-one modified symmetric eigenvalue problem [Bunch et al. 1978, Bunch & Nielsen 1978, Golub 1973]. In each time step an exact decomposition is computed from a previous decomposition. The operation count is $O\{n^3\}$ where n is the problem size. For many practical applications this is a serious impediment. Subspace methods attempt to reduce this, for instance by averaging the smaller singular values (noise cluster) [Karasalo 1986, Schreiber 1986, DeGroat & Roberts 1988, 1990]. The operation count is reduced to $O\{nd^2\}$, where d is the number of large singular values, but on the other hand in each time step an additional error is introduced in adjusting the smaller singular values.

As an alternative to these methods, we derive a **Jacobi-type SVD updating procedure** as a combination of a Jacobi-type SVD procedure [Kogbetliantz 1954, 1955, Luk 1986a] with QR updating. Apparently only a few SVD steps after each QR update suffice in order to restore an acceptable approximation for the SVD, hence the computational saving. This updating procedure is shown to be not only cheap –$O\{n^2\}$, similar to mere QR updating– but also fairly accurate and particularly suited for parallel implementation. Furthermore this procedure is easily extended to various generalized SVD updating problems –QSVD updating for instance– problems which had not been looked at before, as far as we know.

The theory is illustrated by means of a practical application from systems theory, namely **systems identification.** First an SVD-based state space identification procedure is derived, where a state space model is computed from an SVD of a block Hankel matrix constructed with input/output data. This algorithm is related to Kung's realization algorithm [Kung 1978], which is similar but goes via the Markov parameters first. Furthermore a QSVD-based counterpart for the coloured noise case is developed. These algorithms are also worked into adaptive algorithms for tracking time-varying systems. Essentially this then boils down to SVD and QSVD updating.

Related updating problems

While the SVD updating problem is important in its own right, the suggested updating procedure seems to be only one representative of

a whole class of important (updating) algorithms for signal processing and systems theory. The key tool is the Jacobi approach. For almost all matrix decompositions, Jacobi-type algorithms are available, or existing algorithms can be reformulated and worked into Jacobi-type algorithms, which are based on locally computed plane transformations. Apart from the SVD algorithm [Luk 1986a], efficient Jacobi-type algorithms have been derived for the symmetric eigenvalue problem [Brent & Luk 1985], for the Schur [Stewart 1985] and generalized Schur decomposition [Charlier & Van Dooren 1989a], for the quotient SVD [Luk 1985, Paige 1986] and the product SVD [Heath et al. 1986], for the QR decomposition [Luk 1986b], the LDU and Cholesky decomposition [Moonen et al. 1990e], for inverting triangular matrices –see Chapter 5. The primary appeal of the Jacobi approach is its simplicity, which e.g. translates into an efficient and elegant implementation on parallel processor arrays.

Many practical algorithms in e.g. control and filtering essentially consist of a combination of elementary matrix operations, most of which can be recast as Jacobi-type operations. The SVD updating algorithm is only one example. In general, we can assemble elementary Jacobi-type algorithms into combined Jacobi-type algorithms. The resulting algorithms are then regularly structured and have potential for efficient parallel implementation. Apart from the SVD updating algorithm, we derive Jacobi-type algorithms for Kalman filtering, for solving Riccati and Lyapunov equations and for recursive least squares computations.

Systolic arrays

Systolic arrays are special-purpose structured grids of elementary processing elements, where each processor performs short computations and regularly pumps data in an out so that a regular data flow is kept up in the network [Kung & Leiserson 1980]. The parallelism provides an adequate means to speed up computation or data treatment for several real time applications with high throughput rates.

All of the elementary Jacobi-type algorithms can efficiently be implemented on systolic arrays. The same holds true for implementing combined Jacobi-type algorithms. Sometimes, additional operations have to be implemented such as matrix-vector multiplications or triangular backsolves, e.g. in the SVD updating procedure. We develop a new

procedure for interlacing these operations with the Jacobi-type operations. Different operations then correspond to different computational fronts. Whenever two such fronts cross, additional computations have to be performed for updating the intermediate results. The computational overhead is however particularly small, while on the other hand the resulting implementation fully exploits the available parallelism, resulting in the highest possible throughput. The obtained arrays are extremely efficient and mostly outperform previously developed systolic implementations.

Chapter by chapter overview

Chapter 1 *is introductory and contains necessary* **background material.** *The mathematical theory of SVD and QRD is briefly reviewed, and the significance of these decompositions in engineering problems is illustrated with a number of theorems and applications, in particular by means of our state space identification procedure. Algorithms for computing and updating the QRD are given. Kogbetliantz's SVD algorithm [Kogbetliantz 1954, 1955] is explained, wherewith at the same time the Jacobi approach to computing various matrix decompositions is introduced.*

Chapter 2 *exclusively deals with* **SVD updating.** *First of all a fast $O\{n^2\}$ updating algorithm is derived, where the Jacobi-type SVD algorithm is interlaced with matrix-vector products and QR updates whenever a new observation is worked in. This algorithm is an alternative to $O\{n^3\}$ algorithms. It yields an approximate decomposition after each update, with a tracking error which is shown to be $O\{n\}$ for subspace tracking applications. Accuracy is thus traded off for speed. An error analysis is included, wherewith the updating algorithm is proved to be stable when supplemented with a certain re-orthogonalization procedure. The updating algorithm as well as the performance and error analysis are new.*
This chapter is based on [Moonen et al. 1989b].

Chapter 3 *focuses on the* **parallel implementation** *of the SVD updating algorithm. It is shown how the SVD procedure, intertwined with*

matrix-vector multiplications and QR updates, can be implemented on a systolic array with $O\{n^2\}$ processors. A perfect pipelining is obtained after adding a few additional computations at the crossings of different computational fronts. The available parallelism is fully exploited, and the throughput is $O\{n^0\}$, much like it is the case for mere QR updating. The idea of combining different operations on the same systolic array –with intermingled computational fronts and a small computational overhead for an increased throughput– is apparently new.

Furthermore, it is shown how a square root free implementation can be obtained by combining modified Givens rotations with approximate SVD schemes. This extends the well known square root free QR updating procedure [Gentleman 1973, Hammarling 1974].

Finally, while Jacobi-type algorithms are mostly implemented on orthogonally connected processor arrays [Stewart 1985, O'Leary & Stewart 1985], we give an alternative partitioning, resulting in a hexagonally connected processor grid which is over four times more efficient.
This chapter is based on [Moonen et al. 1989c] and [Moonen & Vandewalle 1990a].

Chapter 4 *extends the SVD updating procedure to* **generalized SVD updating problems.** *We focus on PSVD and QSVD row updating. Sequential algorithms are readily obtained by combining appropriate Jacobi-type algorithms with matrix-vector multiplications and QR updating (QL updating sometimes). We end up with a fairly complete set of updating algorithms. Most of these problems had not been looked at before.*

The systolic implementation of these algorithms highlights the usefulness of our approach. In all cases, all operations are perfectly pipelined with a small computational overhead. The parallel algorithms sometimes differ from the sequential algorithms, but the difference is irrelevant for slowly time-varying systems.
This chapter is based on [Moonen et al. 1989d].

Chapter 5 *extends the principle of combining Jacobi-type algorithms to various related updating problems, resulting in a whole set of useful algorithms and systolic arrays for systems theory and signal processing. Three convincing cases are presented :*

1.- *Recently developed* **recursive least squares (RLS) schemes with inverse factorization** *[Shepherd et al. 1990, Pan & Plemmons 1989] are known to be particularly suited for parallel implementation, but are unfortunately unstable. We show how the procedure can be stabilized by including an error correction scheme based on Jacobi-type matrix inversion. The RLS scheme –QR updating mainly– and the correction scheme can be combined on a systolic array, much like it is the case for the SVD updating procedure. Both the algorithm and its systolic implementation are new.*

A second –and much more efficient– systolic RLS implementation is derived, which is based on the **orthogonal-inverse updating scheme** *of [Pan & Plemmons 1989]. This systolic implementation again has crossing computational fronts in a Jacobi-type partitioning. As far as we know it is the most (only) efficient systolic implementation for RLS.*

2.- *Second, a triangular processor array for* **square root Kalman filtering** *is developed starting from well known updating formulas. The QR updating is combined with a Jacobi-type algorithm for QR factorization. The resulting array is slightly less efficient than the array in [Kung 1988] –which seems to be the most efficient array ever described in the literature– but on the other hand possesses a much more regular structure than the latter and is therefore thought to offer a useful alternative in view of hardware implementation.*

3.- *Finally, a systolic algorithm is devised for* **solving Riccati and Lyapunov equations**. *The recursive matrix sign function algorithm is known to provide a suitable iterative method for solving these equations. In a recent paper a factored version of this algorithm is given, which is particularly amenable to parallel implementation [Charlier & Van Dooren 1989b]. The basic operations involved are triangular matrix inversions and triangular updates. This algorithm is worked into a Jacobi-type algorithm, by combining Jacobi-type algorithms for inverting and updating triangular matrices. The resulting array is conceptually simpler and furthermore roughly three times more efficient than the array in [Charlier & Van Dooren 1989b].*

Chapter 5 is based on [Moonen & Vandewalle 1989e], [Moonen & Vandewalle 1990c], [Moonen & Vandewalle 1990d] and [Moonen & Vandewalle 1990f].

Finally the **state space identification** *procedure provides us with a leading thread, which is used to illustrate the theoretical results in the different chapters. In Chapter 1, the identification scheme is outlined, and an SVD-based implementation is given. In Chapter 2 the corresponding adaptive algorithm is derived –based on SVD updating– and used to check the theoretical performance analysis for the SVD updating itself. Finally a QSVD-based counterpart for the coloured noise case is developed in Chapter 4 to illustrate the QSVD updating procedure. These algorithms are new.*
Artificial examples are included, as well as real-life examples with data from a flexible one-link robot.
The sections on state space identification are based on [Moonen et al. 1989a] and [Moonen & Vandewalle 1990b].

Apart from a collection of new algorithms and corresponding systolic arrays –which are important in their own right– we believe the main contribution of this thesis is in the development of a framework for devising combined algorithms –based on the Jacobi approach– linked up with the development of an apparently new type of systolic array for implementing these combined algorithms. As such, the thesis especially highlights the benefit of developing algorithms and architectures together.

Chapter 1

Background material and motivation

Both the **QR decomposition** *and the* **singular value decomposition** *are important matrix factorizations, with many applications throughout modern engineering. The following chapters are mainly concerned with SVD related issues, where the QRD comes out as an indispensable computational tool. This chapter contains necessary background material, and serves as a glossary of standard concepts and notation. It leisurely introduces the mathematical theory of both the QRD and the SVD, together with a selection of properties and algorithms to be used in the sequel.*

In order to illustrate things and motivate further developments, we dwell upon the use of matrix techniques –especially SVD– for **systems identification.** *Identification aims at finding a mathematical model from a measurement record of inputs and outputs of a system. A state space model is a most obvious choice, but reliable general purpose state space identification schemes have not become standard tools so far, mostly due to the computational complexity involved. We give an outline of an identification scheme, where a state space model is computed from a block Hankel matrix with I/O data [Moonen et al. 1989a]. In a way, it is related to the realization algorithms in [Ho and Kalman 1966, Zeiger and Mc Ewen 1974] and especially [Kung 1978], where a Hankel matrix with Markov parameters is used. The Markov parameters might be hard to obtain though*[1]*, and furthermore –as far as we know– the latter algorithms do not allow for tracing the effect of*

[1] for infinite impulse response systems, especially with unstable poles, computing a deconvolution directly –i.e. by solving a set of linear equations– is impossible

inexact data. Our algorithm is easier to trace, and can be worked into an adaptive identification scheme by making use of SVD updating, as will be shown in Chapter 2. Even the colouring of additive noise can be taken into account through the use of QSVD and QSVD updating, as demonstrated in Chapter 4. A few artificial examples are included, as well as a real-life example with data from a flexible one-link robot.

As for the numerical computation of the SVD, only **Kogbetliantz's algorithm** *is considered [Kogbetliantz 1955]. Although on single processor machines, this algorithm is not nearly as efficient as the standard Golub-Reinsch-Kahan algorithm [Golub & Kahan 1965, Golub & Reinsch 1970], it has increasingly become a popular method for multiprocessor systems. Also for SVD updating problems to be discussed in the next chapters, Kogbetliantz's algorithm is 'the obvious thing'. In explaining Kogbetliantz's method we are also given an opportunity to introduce a larger class of extremely simple, yet extremely useful matrix algorithms, broadly referred to as* **Jacobi-type methods.** *The Jacobi approach is one of the recurring themes in nearly every chapter. Instead of giving an encyclopaedic overview of all possible Jacobi-type algorithms, in this chapter we confine ourselves to the SVD case. Similar algorithms for generalized SVD's, for matrix inversion, etc. are discussed in subsequent chapters, whenever occasion arises.*

In section 1.1, some basic matrix algebra is recalled and illustrated by means of the state space identification procedure. Then we move on to actual matrix computations. The QR decomposition is reviewed in section 1.2. In section 1.3 the QR updating is discussed, for keeping track of the QRD in adaptive settings. The SVD is reviewed in section 1.4, while in section 1.5 specific SVD algorithms are discussed into some more detail, with emphasis on Jacobi-type methods. Finally in section 1.6, the identification procedure is worked into an SVD-based identification algorithm to be used in the sequel.

1.1 Vectors & matrices, state space identification

Let $\Re^{m\times n}$ denote the vector space of real m-by-n matrices. When a capital letter is used to denote a matrix –e.g. A– the corresponding lowercase letter with subscript ij refers to the (i,j) component

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}.$$

$\Re^m$ denotes $\Re^{m\times 1}$ and for these column vectors we use lowercase letters –e.g. x and y– and denote individual components with single subscripts.

Matrices abundantly appear in systems theory and control, especially through the use of **state space models** for representing linear systems. A state space representation for a linear discrete-time system with m inputs and l outputs –see **Figure 1.1**– consists of a set of first order difference equations in an auxiliary variable vector x, plus an output equation as follows

$$\begin{aligned} x_{[k+1]} &= \underbrace{A}_{n\times n}\cdot x_{[k]} + \underbrace{B}_{n\times m}\cdot u_{[k]} \\ y_{[k]} &= \underbrace{C}_{l\times n}\cdot x_{[k]} + \underbrace{D}_{l\times m}\cdot u_{[k]} \end{aligned}$$

where

$u_{[k]} \in \Re^m$ is the vector of inputs at time instant k

$y_{[k]} \in \Re^l$ is the vector of outputs at time instant k

$x_{[k]} \in \Re^n$ is the state vector at time instant k, n being the minimal system order

A, B, C and D are the system matrices.

The state space model of a linear system is not unique in a sense that an equivalent state space model is obtained by inserting a non-singular matrix T as follows:

$$\begin{aligned} \underbrace{Tx_{[k+1]}}_{z_{[k+1]}} &= \underbrace{TAT^{-1}}_{A^*}\cdot\underbrace{Tx_{[k]}}_{z_{[k]}} + \underbrace{TB}_{B^*}\cdot u_{[k]} \\ y_{[k]} &= \underbrace{CT^{-1}}_{C^*}\cdot\underbrace{Tx_{[k]}}_{z_{[k]}} + D\cdot u_{[k]}. \end{aligned}$$

Only the system matrices and the internal states have been changed while the input-output relation has not been affected. This implies that only for very specific choices of coordinate systems in the state space[2], the components of the state vectors might have a physical meaning.

[2] i.e choices of T

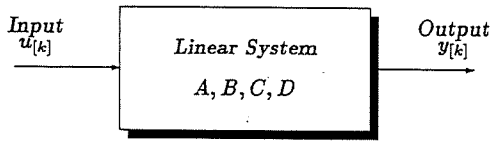


Figure 1.1: Generic linear system.

There are two important subspaces associated with a matrix $A \in \Re^{m\times n}$: the range of A, which is generated by its column vectors

$$\begin{aligned} \mathcal{R}\{A\} &= span\{a_1, \ldots, a_n\} \\ &= span_{column}\{A\} \end{aligned}$$

where

$$A = \left[\begin{array}{|c|}\hline a_1 \\ \hline\end{array}\;\begin{array}{|c|}\hline a_2 \\ \hline\end{array}\;\cdots\;\begin{array}{|c|}\hline a_n \\ \hline\end{array}\right]$$

and the null space, which is orthogonal onto all of the row vectors

$$\mathcal{N}\{A\} = span^{\perp}_{row}\{A\}.$$

The rank of A is of course the dimension of its column space[3]

$$rank\{A\} = dim\{\mathcal{R}\{A\}\}.$$

The singular value decomposition provides a numerically sound way of computing both the matrix rank and the fundamental subspaces. As an example, the matrix rank equals the number of non-zero singular values, see section 1.4.

The following **state space identification procedure** illustrates the use of these concepts in systems theory.

Suppose we are given a record of I/O-data of a linear system, but the system itself –i.e. both the system order and the system matrices– is unknown. **Figure 1.2** exhibits such data record for an (unknown) third order system

[3] equal to the dimension of the row space

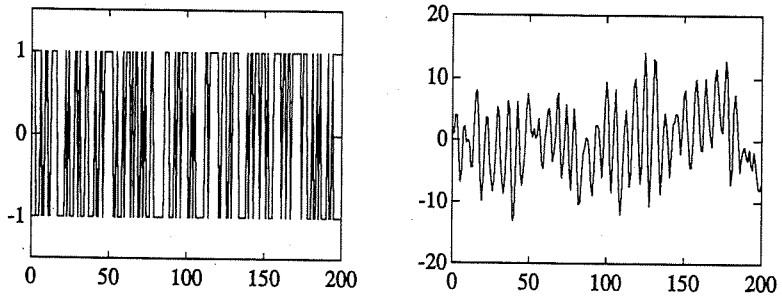


Figure 1.2: *Observed in- and output of a linear system.*

$$x_{[k+1]} = \begin{bmatrix} .9 & 0 & 0 \\ 0 & .5 & .8 \\ 0 & -.8 & .5 \end{bmatrix} \cdot x_{[k]} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot u_{[k]}$$
$$y_{[k]} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \cdot x_{[k]} + \begin{bmatrix} 1 \end{bmatrix} \cdot u_{[k]}.$$

Systems identification aims at recovering the model –up to a similarity transform– from the I/O-data $u_{[1]}, u_{[2]}, \ldots, y_{[1]}, y_{[2]}, \ldots$.

For a start, we can construct a block Hankel matrix with the given I/O data as follows

$$\underbrace{H}_{j\times(2mi+2li)} = \begin{bmatrix} z_{[1]}^T & z_{[2]}^T & \cdots & z_{[2i]}^T \\ z_{[2]}^T & z_{[3]}^T & \cdots & z_{[2i+1]}^T \\ \vdots & \vdots & & \vdots \\ z_{[j]}^T & z_{[j+1]}^T & \cdots & z_{[j+2i-1]}^T \end{bmatrix}$$

where

$$\underbrace{z_{[k]}^T}_{1\times(m+l)} = \begin{bmatrix} \underbrace{u_{[k]}^T}_{1\times m} & \underbrace{y_{[k]}^T}_{1\times l} \end{bmatrix}.$$

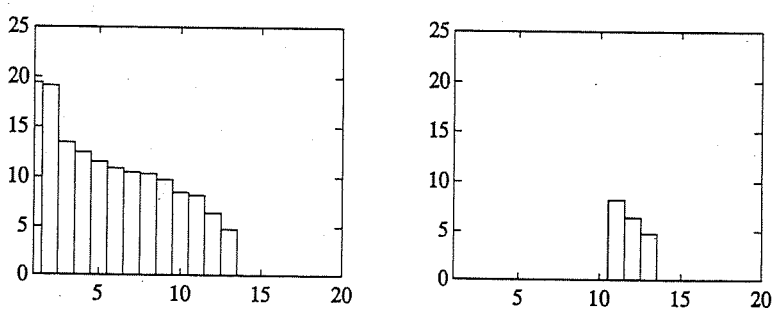


Figure 1.3: *Left: full singular value spectrum. Right: 'order revealing part'.*

In [Moonen et al. 1989a], it is shown that under mild conditions [4]

$$\text{rank}\{H\} = \underbrace{2mi}_{\text{rank input part}} + n.$$

This rank property allows to recover the system order n. *For the given I/O data for instance, an* $H_{100\times 20}$*-matrix with* $i = 5$ *and* $j = 100$ *has a singular value spectrum as in* **Figure 1.3** *(left-hand). The number of non-zero singular values is clearly 13, so that the system order must be* $13 - 2mi = 3$. *The right-hand figure shows the 'order revealing part' of the spectrum, starting from the* $2mi + 1$*-th singular value. Let us only mention that both* i *and* j *should be large enough for the rank property to be satisfied. Furthermore, for an efficient noise cancellation –see below– the best choice is to have* $j \gg i$, *in other words to use 'tall thin matrices' with many more rows than columns.*

Suppose we now cut the matrix H *lengthwise into two pieces* H_1 *and* H_2

$$\underbrace{H}_{j\times(2mi+2li)} = \left[\begin{array}{c|c} \underbrace{H_1}_{j\times(mi+li)} & \underbrace{H_2}_{j\times(mi+li)} \end{array}\right].$$

[4] it is e.g. assumed that the system is persistently excited.

The rank property equally holds true for H_1 and H_2 (assuming i is large enough)

$$\text{rank}\{H_1\} = mi + n$$

$$\text{rank}\{H_2\} = mi + n.$$

Hence

$$\begin{aligned} & dim\{\mathcal{R}\{H_1\} \cap \mathcal{R}\{H_2\}\} \\ &= dim\{\mathcal{R}\{H_1\}\} + dim\{\mathcal{R}\{H_2\}\} - \underbrace{dim\{\mathcal{R}\{H_1\} + \mathcal{R}\{H_2\}\}}_{dim\{\mathcal{R}\{H\}\}} \\ &= (mi+n) + (mi+n) - (2mi+n) \\ &= n \end{aligned}$$

The ranges of H_1 and H_2 thus have an n-dimensional intersection, which is remarkable. Furthermore, in [Moonen et al. 1989a], it is shown that

$$\mathcal{R}\{H_1\} \cap \mathcal{R}\{H_2\} = \mathcal{R}\{\mathcal{X}\}$$

where

$$\underbrace{\mathcal{X}}_{j \times n} = \begin{bmatrix} \boxed{x_{[i+1]}^T} \\ \boxed{x_{[i+2]}^T} \\ \vdots \\ \boxed{x_{[i+j]}^T} \end{bmatrix}.$$

The matrix $\mathcal{X}$ thus corresponds to a certain state vector sequence. Any basis for $\mathcal{R}\{H_1\} \cap \mathcal{R}\{H_2\}$ corresponds to a valid realization for $\mathcal{X}$. The arbitrariness in the choice of this basis corresponds to the arbitrariness of the matrix T in the state space model. The intersection of the column spaces of H_1 and H_2 can be computed by means of an SVD (see section 1.6). In our example, these computations result in an 100×3 matrix $\mathcal{X}$.

Once the state vector sequence $\mathcal{X}$ is known, the system matrices A, B, C and D can be computed from a set of linear equations as follows

$$\begin{bmatrix} x_{[i+2]} & \cdots & x_{[i+j]} \\ y_{[i+1]} & \cdots & y_{[i+j-1]} \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} x_{[i+1]} & \cdots & x_{[i+j-1]} \\ u_{[i+1]} & \cdots & u_{[i+j-1]} \end{bmatrix}.$$

This set is overdetermined, but for exact data it has an exact solution. For our example, we finally obtain a state space model as follows

$$\begin{aligned} x_{[k+1]} &= \begin{bmatrix} .4481 & -.8403 & .0904 \\ .7205 & .4925 & .2657 \\ -.0528 & -.1424 & .9194 \end{bmatrix} \cdot x_{[k]} + \begin{bmatrix} .4410 \\ -.0437 \\ -.1818 \end{bmatrix} \cdot u_{[k]} \\ y_{[k]} &= \begin{bmatrix} 5.6733 & .1993 & -2.7983 \end{bmatrix} \cdot x_{[k]} + \begin{bmatrix} 1 \end{bmatrix} \cdot u_{[k]}. \end{aligned}$$

This state space model corresponds to the original one, up to a transformation T. For instance, it is readily checked that the eigenvalues of the identified A-matrix equal 0.9 and $0.5 \pm j0.8$, i.e. the original system poles. At the end of this chapter, a real-life identification example is given, with data form a **flexible one-link robot**.

In conclusion, we have derived an elegant **two-step identification scheme**. *First a state vector sequence is computed as the intersection of the column spaces of two block Hankel matrices constructed with I/O-data. Then the system matrices are obtained at once from a set of linear equations. This is only a crude outline of course. When it comes to actually computing things –in a numerically sound way– we will have to make use of reliable matrix techniques. Some of these are introduced in the next sections.*

1.2 QR decomposition

Let A be a given $m \times n$ matrix ($m \geq n$). Below, we show how to compute an orthogonal[5] matrix $Q_{m \times m}$, together with an upper triangular factor $R_{n \times n}$ such that

[5] $Q^T Q = I_{m \times m}$

$$A = Q \cdot \begin{bmatrix} R \\ 0 \end{bmatrix}.$$

We refer to this as the QR factorization or QR decomposition (QRD) of the matrix A [Golub & Van Loan 1983, pp 146-147]. Briefly, computing the QRD is a possible step in finding the determinant of a square matrix, in solving linear equations, in solving linear least squares problems, and in many other numerical matrix computations. Furthermore –and in view of our further exposition– efficient SVD algorithms mostly start off from an initial QR decomposition. Before we give an algorithm for computing the QRD, we discuss these aspects in some more detail.

QRD in linear algebra

The QR factorization essentially reduces to the matrix formulation of the Gram-Schmidt process for orthonormalizing the columns of A [Golub & Van Loan 1983, pp 150-152]. Hence, the QRD is useful for 'full rank problems', where the data matrix A is known to have full column rank. The Q-matrix then provides an orthogonal basis for the range of A, as well as for its orthogonal complement:

$$\begin{aligned} \mathcal{R}\{A\} &= \text{span}\{q_1, \ldots, q_n\} \\ \mathcal{R}^{\perp}\{A\} &= \text{span}\{q_{n+1}, \ldots, q_m\} \end{aligned}$$

where

$$Q = \left[\boxed{q_1} \; \boxed{q_2} \; \cdots \; \boxed{q_m} \right].$$

For rank deficient matrices[6]*, the QRD is not even uniquely determined [Björck 1987, p 26]. For instance, for any θ, we have*

$$\underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}}_{Q} \cdot \underbrace{\begin{bmatrix} 0 & \sin\theta \\ 0 & \cos\theta \end{bmatrix}}_{R}.$$

Clearly, in this example Q does not provide a basis for the range of A. Furthermore, for nearly rank deficient matrices, the QRD fails to clearly exhibit the near rank deficiency. Therefore, the QRD is sometimes modified into

[6] like H_1 and H_2 in the identification procedure of *section 1.1*

$$A \cdot \Pi = Q \cdot \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where Π is a suitably chosen permutation matrix, so that near rank deficiency is exhibited in R, having a small lower right block. This is referred to as the rank revealing QR [Chan 1987, Chan & Hansen 1990]. We will however not pursue this in the sequel.

QRD for practical applications, least squares problems

A feature of the QR factorization which is particularly important in view of practical applications is the following. It is readily seen that

$$A^T \cdot A = R^T \cdot R.$$

The matrix R is then referred to as the Cholesky factor [Golub & Van Loan 1983, pp 88-89] of the normal matrix $A^T \cdot A$, sometimes also referred to as the square root of $A^T \cdot A$. It can be computed directly from the data matrix A (QR decomposition), without explicitly having to form the inner product $A^T \cdot A$, which is known to be extremely damaging from a numerical point of view [Hammarling 1985]. In other words, R gives a convenient and compact representation of the normal matrix $A^T \cdot A$, the latter being much more sensitive to perturbations in the data than the data matrix A itself. Algorithms, which conceptually use $A^T \cdot A$, but explicitly work with R, are referred to as square root algorithms.

As an example, the QRD is used for solving full rank **linear least squares (LS) problems.** *The unique solution to the least squares problem*

$$\min_x \|b - A \cdot x\|_2$$

with $b \in \Re^m$, $A \in \Re^{m \times n}$ and $\text{rank}\{A\} = n$, is known to be

$$x_{LS} = (A^T \cdot A)^{-1} \cdot A^T \cdot b.$$

The QRD then provides a numerically sound alternative for computing this solution, wherewith forming the normal matrix is avoided [Golub 1965]. Substituting the QRD of A in the above expression for x_{LS} results in

$$x_{LS} = R^{-1} \cdot y$$

where

$$Q^T \cdot b = \begin{bmatrix} y \\ z \end{bmatrix}.$$

The LS problem is probably one of the most recurring problems in applied mathematics [Lawson and Hanson 1974, Björck 1987]. As an example, the second step in the identification procedure boils down to solving an overdetermined set of equations, preferably in a least squares sense. The QRD approach has –or should have– by now become the most widely applied solution method. For the general case however, where A is rank deficient or nearly rank deficient –and then so is R– it is appropriate to use the SVD, see below.

QRD and SVD

Finally, the QRD provides a useful preprocessing step for several SVD algorithms. Computing an SVD is an iterative process. The preprocessing step is to reduce the given matrix to a form –upper triangular in this case– which allows for a cheaper subsequent iterative reduction, see [Chan 1982, Luk 1986a] or section 1.5 for further details. In our systems identification example for instance, the state space model was computed from the SVD of a 100×20 matrix H. An initial QRD reduces this matrix to a much smaller 20×20 matrix R, carrying all the necessary information for a successful identification as well, see section 1.6. Subsequent computations are then much cheaper.

The QRD can be computed in several ways. As for the number of operations, the **Givens method** *[Givens 1958] to be presented next is slightly less efficient than e.g. the Householder [Householder 1958] or Gram-Schmidt method, see e.g. [Wilkinson 1965, pp 244-246], but it has definite advantages when updating problems are considered. The reason for this is that the entries in A are zeroed in a much more selective manner. Furthermore the method is very satisfactory because of its numerical stability and the simplicity of coding it.*

The basic tool for zeroing matrix entries is a **Givens rotation** *which is a rank-two modification to the identity of the form*

$$G\{i,j,\theta\} \stackrel{def}{=} \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} & & i & & j \\ & & \downarrow & & \downarrow \end{array} \\ \begin{bmatrix} I_{i-1} & 0 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & \sin\theta & 0 \\ 0 & 0 & I_{j-i-1} & 0 & 0 \\ 0 & -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & I_{m-j} \end{bmatrix} \end{array} \begin{array}{l} \\ \\ \leftarrow i \\ \\ \leftarrow j \end{array}.$$

Premultiplication of an m-vector x by $G\{i,j,\theta\}$ amounts to a rotation of θ degrees in the $\{i,j\}$ coordinate plane:

$$\tilde{x} = G\{i,j,\theta\} \cdot x$$

with $\tilde{x}$ components as follows

$$\begin{aligned} \tilde{x}_i &= \cos\theta \cdot x_i + \sin\theta \cdot x_j \\ \tilde{x}_j &= -\sin\theta \cdot x_i + \cos\theta \cdot x_j \end{aligned}$$

and

$$\tilde{x}_l = x_l \qquad \text{for } l \neq i,j.$$

From these formulas it is clear that we can e.g. force $\tilde{x}_j$ to be zero by setting

$$\tan\theta = \frac{x_j}{x_i}.$$

Thus, a Givens rotation can be used to zero a specific entry in a vector.

Let us now return to our QRD computation for a given matrix A. The general idea is to zero all A-entries below the main diagonal by applying successive Givens rotations. The resulting triangular matrix delivers the required R-factor

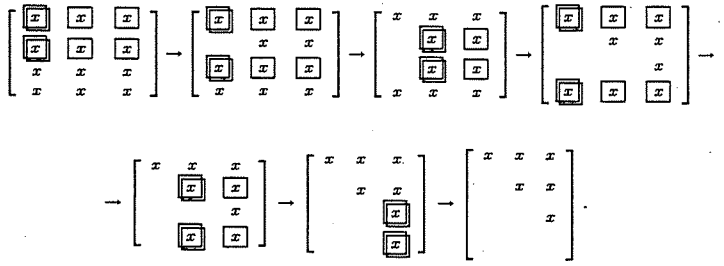
$$\begin{bmatrix} R \\ 0 \end{bmatrix} \leftarrow G\{i_k,j_k,\theta_k\} \ldots G\{i_2,j_2,\theta_2\} \cdot G\{i_1,j_1,\theta_1\} \cdot A$$

$$k = (m-n)n + \frac{1}{2}n(n-1) = \sharp\text{elements below the main diagonal}$$

while the Givens rotations can be accumulated to give Q

$$Q \leftarrow G\{i_1, j_1, \theta_1\}^T \cdot G\{i_2, j_2, \theta_2\}^T \ldots G\{i_k, j_k, \theta_k\}^T.$$

The A-entries should however be zeroed in a certain ordering, such that no fill-ins occur. The following 4×3 example illustrates the general idea. In each step, a row transformation $G\{i, j, \theta\}$ is applied in order to zero the (j, i) entry. 'Active' rows are framed, double framed elements are used to compute the rotation. Elements are seen to be zeroed in a rowwise fashion.



REMARKS

1. *As the number of elements to be zeroed is $O\{nm\}$ and each row transformation requires $O\{n\}$ operations on the average, the operation count for a complete QR factorization is roughly $O\{n^2 m\}$ for an $m \times n$ matrix, $O\{n^3\}$ for a square $n \times n$ matrix.*

2. *For structured matrices –e.g Toeplitz, Toeplitz-like or sparse matrices– tailor-made QRD algorithms have been developed that for instance for the Toeplitz case take only $O\{n^2\}$ operations, see e.g. [Sweet 1984, Bojanczyk et al. 1986, Cybenko 1987, Chun et al 1987]*

3. *The above QRD algorithm can be re-organized into a Jacobi-type QRD algorithm, which is more suited for parallel implementation [Luk 1986b].*

4. *For an error analysis of the QR factorization, see [Wilkinson 1965, Gentleman 1975].*

1.3 QR updating

Given that we have the QR factorization of a matrix A, we may need to calculate the QR factorization of a matrix $\underline{A}$ that is obtained by appending a new row to A

$$\underline{A} = \begin{bmatrix} A \\ a^T \end{bmatrix}.$$

This QR updating is used e.g. for solving recursive least squares problems for adaptive beamforming [McWhirter 1983, Shepherd & McWhirter 1987], etc. In Chapter 2, the QR updating is combined with an SVD procedure to give an SVD updating algorithm.

Instead of computing this factorization from scratch, it is much more efficient to start from the known QRD of A and update it accordingly [Gill et al. 1974]. From the QRD of A, it follows that

$$\underline{A} = \begin{bmatrix} Q & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \begin{matrix} 0 & \ldots & 0 \end{matrix} & 1 \end{bmatrix} \cdot \begin{bmatrix} R \\ 0 \\ a^T \end{bmatrix}.$$

Only the last row in the rightmost matrix needs to be zeroed in order to restore the triangular structure. This can obviously be done by applying a series of Givens rotations. The new row is combined with the first row to zero its first component, then with the second row to zero its second component, etc. If we let Q_a denote the accumulation of the applied Givens rotations, we end up with

$$\underline{A} = \underbrace{\begin{bmatrix} Q & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \begin{matrix} 0 & \ldots & 0 \end{matrix} & 1 \end{bmatrix} \cdot Q_a^T}_{\underline{Q}} \cdot \underbrace{Q_a \cdot \begin{bmatrix} R \\ 0 \\ a^T \end{bmatrix}}_{\begin{bmatrix} \underline{R} \\ 0 \end{bmatrix}}.$$

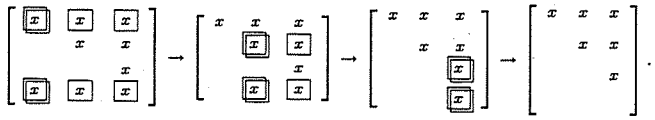
In on-line applications, a new updating is often to be performed after each sampling. The data matrix at time step k is e.g. defined in a recursive manner

$$A_{[k]} = \begin{bmatrix} \lambda_{[k]} \cdot A_{[k-1]} \\ a^T_{[k]} \end{bmatrix}$$

where $\lambda_{[k]}$ $(0 < \lambda_{[k]} \leq 1)$ is a weighting factor for exponential windowing, and $a_{[k]}$ is the observation vector at time instant k. In most cases the $Q_{[k]}$-matrix –with growing matrix dimensions– need not be computed explicitly, and only $R_{[k]}$ is explicitly updated. From the above formulas, it then follows that the relevant updating problem can be recast as follows

$$\begin{bmatrix} R_{[k]} \\ 0 \end{bmatrix} \leftarrow Q_{[k]} \cdot \begin{bmatrix} \lambda_{[k]} \cdot R_{[k-1]} \\ a^T_{[k]} \end{bmatrix}$$

where $Q_{[k]}$ need not to be computed explicitly. The general procedure is then as follows



REMARKS

1. *The QR updating thus amounts to triangularizing a structured matrix, a triangular matrix plus one extra row. As a result of the available structure, only little extra effort is required in order to obtain the factorization. The operation count is reduced substantially, namely to $O\{n^2\}$ per update.*

2. *For an error analysis of the QR updating procedure, we refer to [Gentleman 1975]. The error analysis for the SVD updating in Chapter 2 is heavily based on these results.*

3. *A triangular systolic array for QR updating is developed in [Gentleman & Kung 1981]. We refer to Chapter 3 for a brief description, where this array is combined with an SVD array into an SVD updating array.*

4. *In [Gentleman 1973, Hammarling 1974, Barlow & Ipsen 1987] it is shown how to avoid the square roots in the computation of the rotation angles[7], by making use of a row scaling of the R-factor. This procedure is briefly reviewed in Chapter 3, where a similar square root free SVD updating procedure is developed.*

5. *Downdating amounts to deleting rows from a previously computed QRD or Cholesky factorization, and is thus closely related to updating [Gill et al. 1974, Bojanczyk et al. 1987].*

1.4 SVD theorems and applications

The singular value decomposition of an $m \times n$ matrix A $(m \geq n)$, is a factorization of A into a product of three matrices. It provides a diagonal form of A under an orthogonal equivalence transformation. In other words, whereas a QR factorization amounts to reducing a given matrix to triangular form by applying orthogonal transformations to the left, an SVD amounts to reducing that same matrix to a diagonal one, by applying orthogonal transformations both to the left and to the right.

Theorem 1 *singular value decomposition (for real matrices)*

Let $A \in \Re^{m \times n}$ be a matrix of rank r. Then there exist orthogonal matrices $U \in \Re^{m \times m}$ and $V \in \Re^{n \times n}$ such that

$$A = U \cdot \Sigma \cdot V^T, \qquad \Sigma = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix}$$

where $\Sigma \in \Re^{m \times n}$ and

$$\Sigma_r = \mathrm{diag}\{\sigma_1, \sigma_2, \ldots, \sigma_r\}.$$

The σ_i are called the **singular values**, *which are unique, non-negative and by convention ordered such that*

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0.$$

If we write

$$U = \begin{bmatrix} \boxed{u_1} & \boxed{u_2} & \cdots & \boxed{u_m} \end{bmatrix}$$

[7] $\cos\theta = \frac{1}{\pm\sqrt{1+\tan^2\theta}}$

$$V = \left[\boxed{v_1} \quad \boxed{v_2} \quad \cdots \quad \boxed{v_n} \right]$$

the column vectors u_i and v_i $(i = 1 \ldots r)$ are respectively the **left and right singular vectors**, *corresponding to σ_i, and furthermore*

$$\begin{aligned} \mathcal{R}\{A\} &= span\{u_1, \ldots, u_r\} \\ \mathcal{R}^{\perp}\{A\} &= span\{u_{r+1}, \ldots, u_m\} \\ \mathcal{N}^{\perp}\{A\} &= span\{v_1, \ldots, v_r\} \\ \mathcal{N}\{A\} &= span\{v_{r+1}, \ldots, v_n\}. \end{aligned}$$

The singular vectors corresponding to simple singular values are unique *(up to a sign). For multiple singular values, the corresponding singular vectors can be chosen as any orthonormal basis for the* unique subspace *that they span. Once u_j has been chosen, the corresponding v_j is uniquely determined from*

$$A^T \cdot u_j = \sigma_j \cdot v_j.$$

Similarly, given v_j, the vector u_j is determined from

$$A \cdot v_j = \sigma_j \cdot u_j.$$

REMARK

In the sequel, we mostly consider rectangular matrices $A_{m \times n}$ where $m \geq n$. For such matrices, it is appropriate to define a reduced SVD as follows

$$A = U_n \cdot \Sigma_n \cdot V^T, \qquad \Sigma_n = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix}$$

where now $\Sigma_n \in \Re^{n \times n}$ and $U_n \in \Re^{m \times n}$. The matrix U_n is formed by the first n columns in U. Furthermore, we will often drop the subscript n and use this reduced SVD without explicitly mentioning this.

The rest of this section is devoted to SVD theorems. All of these are well known by now, but we believe that such an overview –although far from complete– highlights the significance of the SVD in engineering applications, see also [Klema & Laub 1980, Hammarling 1985, Deprettere 1988, Vandewalle & De Moor 1988, Biglieri & Yao 1989]

SVD and the symmetric EVD

First of all, the SVD is clearly related to the eigenvalue decomposition (EVD) of the gramians $A^T \cdot A$ and $A \cdot A^T$. From the SVD of A it follows that

$$A^T \cdot A = V \cdot \Sigma^T \cdot \Sigma \cdot V^T$$

$$A \cdot A^T = U \cdot \Sigma \cdot \Sigma^T \cdot U^T.$$

Hence

Theorem 2

The singular singular values of A are the square roots of the nonzero eigenvalues of $A^T \cdot A$ and $A \cdot A^T$, and the left and right singular vectors are the eigenvectors of $A \cdot A^T$ and $A^T \cdot A$ respectively.

Similar to what we had with the QRD, the SVD of A thus allows for inferring information on the normal matrices without having to compute these explicitly. Many signal processing algorithms conceptually use $A^T \cdot A$-type data covariance matrices, while for explicit computations, use is made the SVD of the original data matrix A. The greater accuracy associated with the SVD approach is due to fact that the eigenvalues of $A^T \cdot A$ have twice the dynamic range of the singular values of A, such that the relative inaccuracy of the computed smaller eigenvalues of $A^T \cdot A$ might be huge. Applications include high resolution tracking of narrow-band signals [Tufts & Kumaresan 1982, Tufts et al. 1982, Kung et al. 1983, Rao 1988, Arun 1989], direction finding [Roy 1987, Paulraj et al. 1986], systems identification [Larimore & Luk 1988], etc.

Matrix norms and conditioning of linear systems

The SVD plays a crucial role in numerical analysis through the definition of matrix norms [Golub & Van Loan 1983, pp 14-16]. First of all, from the above relationship to the EVD, it is not too surprising that the singular values have a minmax characterization, similar to the Courant-Fisher characterization for eigenvalues :

$$\sigma_i = \min_{\dim\{S\}=n-i+1} \max_{x \in S_0} \frac{\|Ax\|_2}{\|x\|_2}$$

where S is linear subspace of $\Re^n$, and $S_0 = S \setminus \{0\}$. Hence, the induced 2-norm of A equals its largest singular value

$$\begin{aligned} \|A\|_2 &\stackrel{def}{=} \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} \\ &= \sigma_1. \end{aligned}$$

Another frequently used matrix norm is the Frobenius norm

$$\begin{aligned} \|A\|_F &\stackrel{def}{=} \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} \\ &= \sqrt{\sum_{i=1}^{n} \sigma_i^2}. \end{aligned}$$

The notion of matrix norms allows for assessing the size of a matrix, which is indispensable in analyzing matrix algorithms, see e.g. [Wilkinson 1965, Golub & Van Loan 1983]. As an example, it is well known that for a square matrix $A_{n\times n}$, the sensitivity (or conditioning) of the linear system $Ax = b$ is defined in terms of the 2-norm of both A and its inverse. The relative error in x is at most κ times the relative error in A and b, where κ is the condition number

$$\begin{aligned} \kappa &= \|A\|_2 \cdot \|A^{-1}\|_2 \\ &= \frac{\sigma_1}{\sigma_n}. \end{aligned}$$

Matrix rank and approximation problems

The minmax characterization may furthermore be used to establish a relation between the singular values of two matrices A and $\bar{A} = A + \Delta E$

Theorem 3

Let A and $\bar{A} = A + \Delta E$ have singular values σ_i and $\bar{\sigma}_i$ respectively, then [Stewart 1973b]

$$|\sigma_i - \bar{\sigma}_i| \leq \|\Delta E\|_2.$$

This theorem shows that the singular values are insensitive to perturbations on A. The perturbation of the singular values is smaller than –or at most equal to– the perturbation of the A-entries.

At once, this brings us to one of the most valuable aspects of the SVD. More specifically, the SVD allows us to deal sensibly with the **concept of matrix rank** *[Klema & Laub 1980]. Rounding errors and fuzzy data make rank determination a nontrivial task in practice. In general, if a rank-r matrix is slightly perturbed, the $(r+1)$-th singular value of the perturbed matrix remains particularly small and therefore discloses the near rank deficiency. Let us briefly return to our identification example. Suppose that measuring the I/O-data is subject to error, additive measurement noise as depicted in* **Figure 1.4.** *Instead of H, only a perturbed version $\bar{H}$ is available, with a singular value spectrum e.g. as in* **Figure 1.5.** *All the singular values are non-zero so that the numerical rank of $\bar{H}$ equals 20. However, the gap in the spectrum indicates that the effective rank is only 13, equal to the rank of H. Hence we can confidently decide on the system order being three. The identified third order system then has slightly perturbed system poles, namely $0.5129 \pm i0.7991$, 0.8783 (original system : $0.5 \pm i0.8$, 0.9).*

The way the SVD allows for a quantification of the near rank deficiency, is once again displayed in the following theorem.

Theorem 4

Let $A \in \Re^{m\times n}$ with SVD $A = U\Sigma V^T$. If $k < r = \text{rank}\{A\}$ and

$$A_k \stackrel{def}{=} \sum_{i=1}^{k} u_i \cdot \sigma_i \cdot v_i^T$$

then

$$\min_{\text{rank}\{B\}=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

$$\min_{\text{rank}\{B\}=k} \|A - B\|_F = \|A - A_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2}.$$

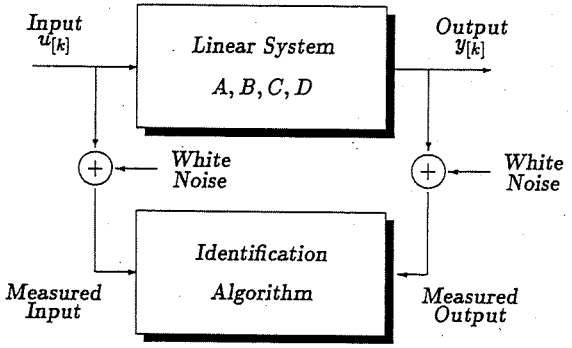


Figure 1.4: *Identification set-up -white noise case.*

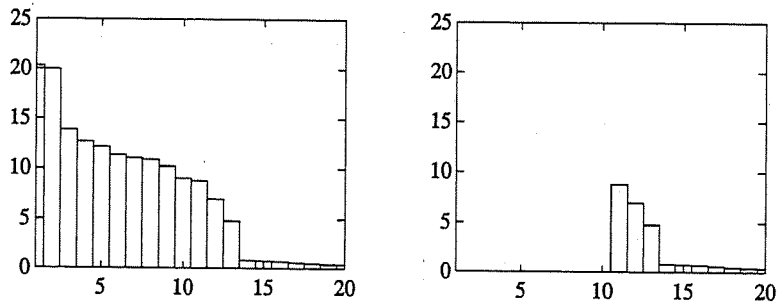


Figure 1.5: *Left: full singular value spectrum. Right: 'order revealing part'.*

This was originally proved for $\|\cdot\|_F$ in [Eckart and Young 1936], and extended to $\|\cdot\|_2$ in [Mirsky 1960]. The theorem gives a measure of how far A is from the nearest rank-k matrix, and states that amongst such matrices, A_k is closest to A.

The above theorems illustrate the importance of the SVD as a key tool for a number of approximation problems, e.g. for model reduction [Kung 1978, Larimore 1983], principal component analysis [Moore 1981], image coding [Andrews & Patterson 1976a, 1976b], etc.

> *In digital image processing for instance, an image is represented by an $N \times N$ matrix, obtained after two-dimensional spatial sampling and quantizing the optical intensity at each particular point. In image coding, the motivation is to effectively reduce the number of bits necessary to represent the image and yet maintain an acceptable quality of reconstruction. Let the SVD of the image matrix A be given as*
>
> $$A = U \cdot \Sigma \cdot V^T = \sum_{i=1}^{N} u_i \cdot \sigma_i \cdot v_i^T$$
>
> *then one of the early techniques consisted in truncating the outer product expansion after say R terms, corresponding to the largest singular values*
>
> $$A_R = \sum_{i=1}^{R} u_i \cdot \sigma_i \cdot v_i^T.$$
>
> *The storage requirement is then reduced from N^2 to $R(2N+1)$.*

Pseudo-inverse, LS and TLS

The SVD is known to be the most reliable tool for **least squares (LS) system solving.** *If the data matrix has full column rank, a QRD can be used, see section 1.2. For the general case where A can be rank deficient or ill-conditioned, it is appropriate to use the SVD.*

Theorem 5

> *Let $b \in \Re^m$ and $A \in \Re^{m \times n}$ with SVD $A = U\Sigma V^T$ and $\text{rank}\{A\} = r$. Then the unique solution to the least squares problem*
>
> $$\min_x \|b - A \cdot x\|_2$$

for which $\|x\|_2$ is minimum, equals

$$x = \underbrace{V \cdot \begin{bmatrix} \Sigma_r^{-1} & 0 \\ 0 & 0 \end{bmatrix} \cdot U^T}_{A^\dagger} \cdot b.$$

The matrix $A^\dagger$ is referred to as the pseudo-inverse or Moore-Penrose inverse of A [Golub & Van Loan 1983, pp 139-140]. Let it suffice to say that the concept of pseudo-inversion generalizes the common matrix inversion to rectangular and rank deficient matrices. In the case of rank-deficient or ill-conditioned matrices, only non-zero or sufficiently large singular values in Σ_r are inverted, while the zero or close-to-zero singular values are kept back from the inversion (regularization).

As an example, in image processing [Andrews & Patterson 1976], an $N \times N$ object can be represented by an $N^2 \times 1$ vector f. The observed image g is degraded by noise and image blur

$$g = H \cdot f$$

where H is the blur matrix. Image restoration amounts to compensating for the image blur, or reconstructing the object from the image by inverting the blur matrix

$$f_{LS} = H^\dagger \cdot g.$$

The SVD is furthermore used for solving **total least squares (TLS) problems**, *an alternative to least squares system solving [Golub & Van Loan 1980, Van Huffel 1987a]. For an overdetermined set of linear equations*

$$\underbrace{A}_{m \times n} \cdot \underbrace{X}_{n \times d} = \underbrace{B}_{n \times d}$$

a TLS solution is the unique minimum norm solution X_{TLS} to the adjusted set

$$\hat{A} \cdot X_{TLS} = \hat{B}$$

where $\hat{A}$ and $\hat{B}$ are such that

$$span_{column}\{\hat{B}\} \subset span_{column}\{\hat{A}\}$$

and

$$\| \begin{bmatrix} A & B \end{bmatrix} - \begin{bmatrix} \hat{A} & \hat{B} \end{bmatrix} \|_F$$

is minimal.

Theorem 6

The TLS solution to $A \cdot X = B$ follows from the SVD of the augmented matrix

$$\begin{bmatrix} A & B \end{bmatrix} = U \cdot \Sigma \cdot V^T$$

and is given by

$$X_{TLS} = -V_{12} \cdot V_{22}^{-1}$$

if V_{22} is invertible and when the V-matrix is partitioned as follows

$$V = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}$$

$$V_{11} \in \Re^{n \times n}, V_{12} \in \Re^{n \times d}, V_{21} \in \Re^{d \times n}, V_{22} \in \Re^{d \times d}.$$

Many techniques in systems identification and parameter estimation reduce to solving an overdetermined set of linear equations. When both A and B are subject to inaccuracies –measurement errors for instance– the TLS solution technique is preferable to e.g. the least squares technique, where inaccuracies are confined only to the right-hand side. We refer to [Van Huffel 1987a] for an in-depth analysis of the TLS technique, together with a few applications from mechanics, biomedicine, etc. For other TLS applications, see e.g. [Roy 1987, Zoltowski 1987, Arun 1989, Moonen et al. 1989a].

1.5 Kogbetliantz's SVD algorithm

Nearly all implemented SVD codes used today are based on the algorithm of Golub-Reinsch-Kahan [Golub and Kahan 1965, Golub and Reinsch 1970]. It is considered as a most accurate and efficient method for computing the SVD of a general dense matrix. This algorithm however does not lend itself

to parallelization. It has requirements for global communication, as well as several separate phases to the reduction, each of them requiring a different architecture. With the advent of parallel processing, older methods have gained renewed interest because of their properties of parallelism and vectorizability. One of these is Kogbetliantz's method [Kogbetliantz 1954, 1955], which ironically enough seems to be one of the oldest numerical methods for computing the SVD. Furthermore, it is this algorithm we will use in Chapter 2 when constructing an efficient SVD updating procedure. We therefore now exclusively focus on Kogbetliantz's method, referring to the literature for a description of other methods, see e.g. [Hesteness 1958, Golub and Kahan 1965, Golub and Reinsch 1970, Nash 1975, Luk 1980, Golub et al. 1981, Cuppen 1981, Chan 1982, Dongarra & Sorensen 1987].

Kogbetliantz's algorithm

Kogbetliantz's method for computing the SVD is a generalization of Jacobi's method for computing the eigenvalue decomposition of a symmetric matrix [Jacobi 1846]. For the sake of brevity, we only consider the case where A is square, i.e. $m = n$.

The idea is to systematically reduce the quantity

$$off\{A\} = \sum_i \sum_{j \neq i} a_{ij}^2$$

which we refer to as the off norm of A. In each step, a **row transformation** *$G\{i,j,\theta\}$ and a corresponding* **column transformation** *$G\{i,j,\phi\}^T$ are applied*

$$A \leftarrow G\{i,j,\theta\} \cdot A \cdot G\{i,j,\phi\}^T.$$

Only rows i and j and columns i and j are affected. The elements on the crossings of these are readily seen to transform into

$$\underbrace{\begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix}}_{\{A\}_{i,j}} \leftarrow \underbrace{\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}}_{\{G\{i,j,\theta\}\}_{i,j}} \cdot \underbrace{\begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix}}_{\{A\}_{i,j}} \cdot \underbrace{\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}}_{\{G\{i,j,\phi\}^T\}_{i,j}}$$

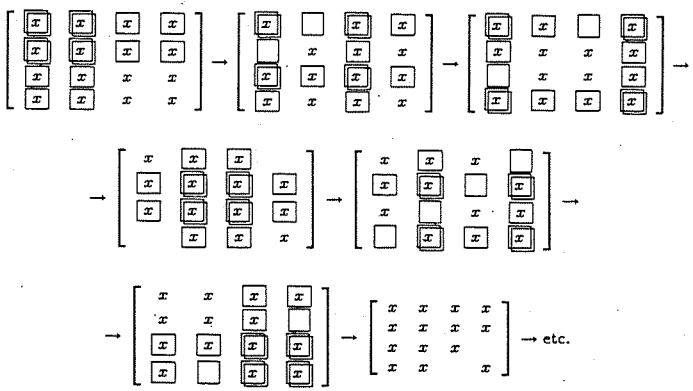
where $\{M\}_{i,j}$ is a 2×2 submatrix of M, on the intersection of rows i and j and columns i and j. The **rotation angles** *θ and ϕ can then*

be chosen such that a_{ij} and a_{ji} are zeroed (this amounts to solving a 2×2 SVD problem). One can easily verify that the off norm is reduced correspondingly

$$off\{A\} \leftarrow off\{A\} - a_{ij}^2 - a_{ji}^2.$$

All off-diagonal elements can then be zeroed following a certain ordering, e.g. cyclic-by-rows or cyclic-by columns. An element that is zeroed in one step reappears in the next step, although mostly with a reduced modulus. The following 4×4 example illustrates the general idea, where:

$$\begin{array}{rcccc} [i,j] & = & [1,2] & [1,3] & [1,4] \\ & & & [2,3] & [2,4] \\ & & & & [3,4] \end{array}$$



A sequence of transformations where all off diagonal elements are zeroed once, is called a **sweep**. *In general –for 'small' matrices ($n < 100$) and if inner rotations are used throughout, see below– at most 10 sweeps are needed to reduce the off norm to the order of the machine precision. The convergence is linear at first [Fernando 1989, Hari & Veselić 1987], quadratic in the final stages [Paige & Van Dooren 1986] in the case of simple singular values. Table 1.1 exhibits a small example with an arbitrary 20×20 matrix.*

# sweeps	off$\{A\}$
	3.5e-01
1	7.4e-02
2	1.2e-02
3	1.2e-03
4	5.8e-06
6	4.3e-12
7	-

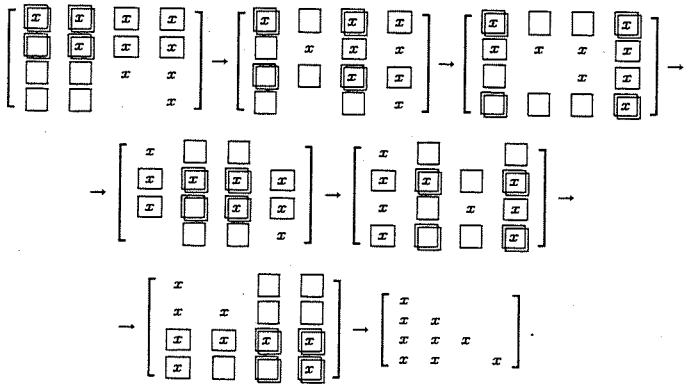
Table 1.1 : off-norm reduction per sweep

Upon convergence, the obtained diagonal matrix has the singular values along the main diagonal. The Givens transformations are accumulated to give U and V.

$$\begin{aligned} U &\leftarrow G\{i_1,j_1,\theta_1\}^T \cdot G\{i_2,j_2,\theta_2\}^T \ldots G\{i_k,j_k,\theta_k\}^T \\ V &\leftarrow G\{i_1,j_1,\phi_1\}^T \cdot G\{i_2,j_2,\phi_2\}^T \ldots G\{i_k,j_k,\phi_k\}^T. \end{aligned}$$

Kogbetliantz's algorithm for triangular matrices

Suppose we had a triangular matrix (e.g. resulting from a QRD) and apply Kogbetliantz's SVD algorithm. Surprisingly enough, it is seen that one sweep (in a cyclic by rows or columns ordering) turns an **upper** *triangular matrix into a* **lower** *triangular matrix [Heath et al. 1986]. Once an element in the upper triangular part is zeroed, it stays zero as it is only combined with other zero-elements in the subsequent steps.*



The next sweep returns the matrix to the original configuration (upper triangular), etc. Clearly the triangular structure is restored after each sweep.

The aim is now to exploit this remarkable property in order to end up with an increased efficiency. Instead of applying only Givens transformations, let us additionally apply row and column **permutations** $\Pi\{i,j\}$

$$A \leftarrow \Pi\{i,j\} \cdot G\{i,j,\theta\} \cdot A \cdot G\{i,j,\phi\}^T \cdot \Pi\{i,j\}^T$$

where

$$\Pi\{i,j\} \stackrel{def}{=} \begin{array}{c} \begin{matrix} & & i & & j \\ & & \downarrow & & \downarrow \end{matrix} \\ \begin{bmatrix} I_{i-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & I_{j-i-1} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{n-j} \end{bmatrix} \end{array} \begin{matrix} \\ \\ \\ \leftarrow i \\ \\ \leftarrow j \end{matrix} .$$

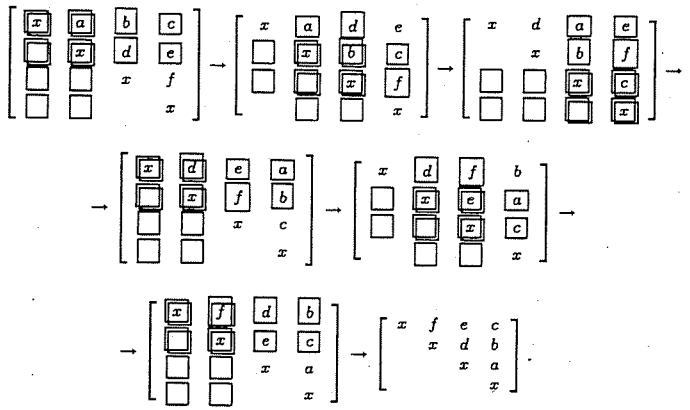
Furthermore, we perform a sequence of transformations where the index i (referred to as the pivot index in the sequel) subsequently takes up the following values

$$i = \begin{matrix} 1 & 2 & 3 \\ 1 & 2 \\ 1 \end{matrix}$$

while on the other hand

$$j = i + 1$$

throughout. In other words, only elements in the first super diagonal are zeroed. In this way, no fill-ins occur in the lower triangular part such that the upper triangular structure is preserved throughout. On the other hand, the permutations bring about that all off-diagonal elements show up in the first superdiagonal exactly once in a sweep. It is observed that such a procedure exactly corresponds to the original scheme without the permutations, apart from location of all elements and operations within the matrix [Stewart 1985, Luk 1986a]. The following example illustrates this, where an original $a-b-c-d-e-f$ configuration is explicitly indicated after each step.



The original configuration is seen to be flipped over its anti-diagonal after the **forward sweep**. *One can verify that a subsequent* **backward sweep**, *where the pivot index takes up the values*

$$\begin{array}{rcccc} i & = & & & 3 \\ & & & 2 & 3 \\ & & 1 & 2 & 3 \end{array}$$

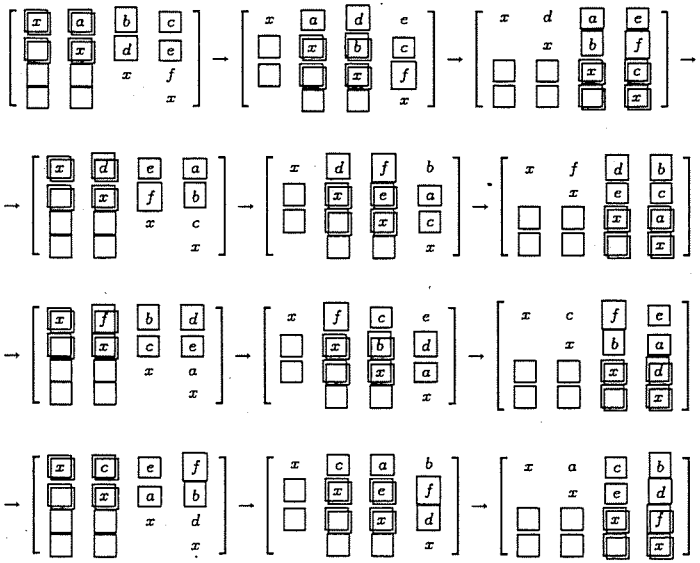
$$j = i + 1$$

restores the original configuration. Finally, a forward and a backward sweep can be combined (pipelined) into a **double sweep** *[Stewart 1985], where the pivot index takes up the values*

$$\begin{array}{rcccc} i & = & 1 & 2 & 3 \\ & & 1 & 2 & \vert 3 \\ & & 1 & \vert 2 & 3 \\ & & \vert 1 & 2 & 3 \end{array}$$

$$j = i + 1.$$

The bars separate between the forward and the backward sweep. It thus suffices to let the pivot index cycle through all possible values. A double sweep consists of n sequences $i = 1, \ldots, n-1$. In a double sweep, each element is zeroed twice, as illustrated below:



resulting in

$$\begin{bmatrix} x & a & b & c \\ & x & d & e \\ & & x & f \\ & & & x \end{bmatrix}.$$

The above re-organized SVD scheme has a number of important advantages over the original scheme:

1. *The upper triangular structure is preserved throughout, so that the storage requirement is roughly cut in half.*
2. *Only contiguous rows and columns are combined, so that e.g. on a systolic array global communication is avoided (see Chapter 3).*
3. *For triangular matrices with multiple or clustered singular values, the ultimate quadratic convergence of the Jacobi procedure is proved in [Charlier & Van Dooren 1987]. A similar proof for full square matrices is apparently not available.*

4. *For the triangular 2×2 SVD problem there exist efficient approximate formulas [Charlier et al. 1988]. In Chapter 3, these are used to derive square root free algorithms.*
5. *Finally, the procedure for triangular matrices can be extended for generalized SVD's [Heath et al. 1986, Paige 1986], see also Chapter 4.*

Inner and outer rotations

Finally, let us briefly focus on the 2×2 SVD's to be solved in the SVD procedure. For a triangular matrix in the re-organized scheme, the relevant formula is as follows (with implicit permutations)

$$\underbrace{\begin{bmatrix} a_{i,i} & 0 \\ 0 & a_{i+1,i+1} \end{bmatrix}}_{\{A\}_{i,i+1}} \leftarrow \underbrace{\begin{bmatrix} -\sin\theta & \cos\theta \\ \cos\theta & \sin\theta \end{bmatrix}}_{\{\Pi\{i,i+1\}G\{i,i+1,\theta\}\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} a_{ii} & a_{i,i+1} \\ 0 & a_{i+1,i+1} \end{bmatrix}}_{\{A\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} -\sin\phi & \cos\phi \\ \cos\phi & \sin\phi \end{bmatrix}}_{\{G\{i,i+1,\phi\}^T\Pi\{i,i+1\}^T\}_{i,i+1}}$$

from which it follows that

$$\tan 2\theta = \frac{-2a_{i+1,i+1}a_{i,i+1}}{a_{i,i}^2 - a_{i+1,i+1}^2 + a_{i,i+1}^2}$$

$$\tan\theta = \frac{\tan 2\theta}{1 \pm \sqrt{1+\tan^2 2\theta}}$$

and

$$\tan\phi = \frac{a_{i+1,i+1}\tan\theta - a_{i,i+1}}{a_{i,i}}.$$

If $\tan\theta$ is chosen to be the smaller root in the above expression, i.e.

$$\tan\theta = \frac{\tan 2\theta}{1 + \sqrt{1+\tan^2 2\theta}}$$

then the angle θ satisfies

$$\theta \in [-\frac{\pi}{4}, \frac{\pi}{4}]$$

and we refer to this as the **inner rotation**. *It is readily seen that (upon convergence) the rotation angles become very small, so that the transformations actually reduce to permutations. The SVD procedure is proved to converge if inner rotations are used throughout.*

*Alternatively, one could also leave out the permutations, and choose the larger rotation angle (***outer rotation***, corresponding to the minus sign in the formula for $\tan\theta$) for the 2×2 SVD*

$$\underbrace{\begin{bmatrix} a_{i,i} & 0 \\ 0 & a_{i+1,i+1} \end{bmatrix}}_{\{A\}_{i,i+1}} \leftarrow \underbrace{\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}}_{\{G\{i,i+1,\theta\}\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} a_{ii} & a_{i,i+1} \\ 0 & a_{i+1,i+1} \end{bmatrix}}_{\{A\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}}_{\{G\{i,i+1,\phi\}^T\}_{i,i+1}}.$$

We will however not pursue this in the sequel.

REMARKS

1. *Needless to say, for a rectangular matrix A, one should first compute the QRD and then apply the SVD procedure to the R-factor. The SVD then obviously follows from*

$$\begin{aligned} A &= Q \cdot \begin{bmatrix} R \\ 0 \end{bmatrix} \\ &= Q \cdot \begin{bmatrix} U_R \cdot \Sigma_R \cdot V_R^T \\ 0 \end{bmatrix} \\ &= \underbrace{Q \cdot \begin{bmatrix} U_R & 0 \\ 0 & I \end{bmatrix}}_{U} \cdot \underbrace{\begin{bmatrix} \Sigma_R \\ 0 \end{bmatrix}}_{\Sigma} \cdot \underbrace{V_R^T}_{V^T}. \end{aligned}$$

   Alternatively, for square matrices, one could also directly apply the original SVD procedure, possibly with additional permutations when parallelism is important (e.g. on a systolic array).

2. *A triangular systolic array for the SVD is developed in [Luk 1986a]. We refer to Chapter 3 for a brief description, where this array is combined with a Gentleman-Kung QR updating array into an SVD updating array.*

3. *In Chapter 3 it is also shown how to avoid the square roots in the computation of the rotation angles, by making use of a generalized Gentleman procedure [Gentleman 1973], together with approximate SVD schemes [Charlier et al. 1988]*

4. *The Kobgetliantz procedure can be applied for computing generalized SVD's as well, e.g. the quotient SVD [Paige 1986] and the product SVD [Heath et al. 1986]. In Chapter 4 these algorithms are briefly reviewed and turned into generalized SVD updating procedures.*

1.6 State space identification revisited

In section 1.1 we gave an outline of a state space identification scheme. With the mathematical background of the previous sections, we are now able to work this scheme into a practical algorithm. The following derivation –which is slightly different from the one in [Moonen et al. 1989a]– shows how the computations can be carried out quite easily, resulting in a consistent (see below) **double SVD identification algorithm.**

First of all, the intersection of the ranges of H_1 and H_2 can be recovered from the null space of H, which in turn follows from the SVD of H

$$\begin{aligned} H &= \left[\begin{array}{c|c} H_1 & H_2 \end{array}\right] \\ &= U_H \cdot \Sigma_H \cdot V_H^T \\ &= U_H \cdot \left[\begin{array}{cc} \Sigma_{11} & 0 \\ 0 & 0 \end{array}\right] \cdot \left[\begin{array}{cc} V_{11} & V_{12} \\ V_{21} & V_{22} \end{array}\right]^T \end{aligned}$$

$$\begin{aligned} V_{11} &\in \Re^{(mi+li)\times(2mi+n)} \\ V_{12} &\in \Re^{(mi+li)\times(2li-n)} \\ V_{21} &\in \Re^{(mi+li)\times(2mi+n)} \\ V_{22} &\in \Re^{(mi+li)\times(2li-n)} \\ \Sigma_{11} &\in \Re^{(2mi+n)\times(2mi+n)}. \end{aligned}$$

From

$$H_1 \cdot V_{12} = -H_2 \cdot V_{22}$$

it follows that the range of $H_1 \cdot V_{12}$ equals the required intersection . However, $H_1 \cdot V_{12}$ contains $2li - n$ column vectors[8], only n of which are linearly independent [9]. Thus, it remains to select n suitable combinations of these column vectors.

[8] cfr. V_{12}'s dimensions

[9] cfr. the dimension of the intersection

By making use of a CS-decomposition[10], one can show that

$$V_{12} = \left[\begin{array}{ccc} \underbrace{V_{12}^{(1)}}_{li-n} & \underbrace{V_{12}^{(2)}}_{n} & \underbrace{V_{12}^{(3)}}_{li-n} \end{array}\right] \cdot \left[\begin{array}{ccc} I & & \\ & C & \\ & & 0 \end{array}\right] \cdot V_\star^T$$

$$V_{22} = \left[\begin{array}{ccc} \underbrace{V_{22}^{(1)}}_{li-n} & \underbrace{V_{22}^{(2)}}_{n} & \underbrace{V_{22}^{(3)}}_{li-n} \end{array}\right] \cdot \left[\begin{array}{ccc} 0 & & \\ & S & \\ & & I \end{array}\right] \cdot V_\star^T$$

$$\begin{aligned} C &= diag\{c_1, \ldots, c_n\} \\ S &= diag\{s_1, \ldots, s_n\} \\ C^2 + S^2 &= I_{n\times n}. \end{aligned}$$

Clearly, only $V_{12}^{(2)}$ delivers useful combinations for the computation of the intersection[11], and we can take

$$\boxed{\mathcal{X} = H_1 \cdot V_{12}^{(2)}.}$$

The above expressions for V_{12} and V_{22} are in itself SVD's of these matrices, and can be computed as such. It thus suffices to compute e.g. the SVD of V_{12} and select the left singular vectors corresponding to non-zero singular values different from 1. Computing $\mathcal{X}$ then reduces to computing two successive SVD's (for H and V_{12} respectively).

REMARK

The above derivation essentially reduces to a double SVD approach to computing the QSVD of the matrix pair $\{H_1^T, H_2^T\}$, following from the constructive QSVD proof in [Paige & Saunders 1981], see also Chapter 4. Therefore one might be tempted to immediately apply a one stage QSVD-procedure to the matrix pair, as developed in [Paige 1986]. This latter method would however compute the exact intersection of the column spaces, which

[10] essentially a simultaneous SVD of V_{12} and V_{22} [Golub & Van Loan 1983, pp 22-24]

[11] for instance $V_{12}^{(1)}$ constitutes the $(li-n)$-dimensional null space of H_1, etc.

in the presence of noise turns out to reduce to the null vector (generically). The outcome of applying Paige's algorithm would then be a zero dimensional intersection, as could be guessed beforehand. The difference between these methods turns out to be the intermediate rank decision after the first SVD in the first approach (double SVD), that fixes the dimension of the approximate intersection to be computed next. Although this (possibly difficult) intermediate rank decision has been a main motive for developing a one stage QSVD algorithm, for our purpose it is somehow inevitable.

In the second step, the system matrices are to be identified from a set of linear equations. Much like it was done in [Moonen et al. 1989a], it can straightforwardly be shown that the system matrices can be computed from the following reduced set as well (obtained after discarding the common orthogonal factor U_H)

$$\begin{bmatrix} V_{12}^{(2)T} \cdot \{V_H\}_{m+l+1:(i+1)(m+l),1:2mi+n} \cdot \Sigma_{11} \\ \{V_H\}_{mi+li+m+1:(m+l)(i+1),1:2mi+n} \cdot \Sigma_{11} \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} V_{12}^{(2)T} \cdot \{V_H\}_{1:mi+li,1:2mi+n} \cdot \Sigma_{11} \\ \{V_H\}_{mi+li+1:mi+li+m,1:2mi+n} \cdot \Sigma_{11} \end{bmatrix}$$

where $\{V_H\}_{r:s,v:w}$ is a submatrix of V_H at the intersection of rows $r, r+1, \ldots, s$ and columns $v, v+1, \ldots, w$. This set of equations can be solved in a least squares sense –if the data are inexact– by making use of a QRD or an SVD, see sections 1.2 and 1.4

The fact that the system matrices can be computed from a reduced set of equations is important for four reasons :

1. *Obviously, there is a significant computational saving being made in the second stage. For the example of section 1.1, the largest dimension in the set of equations is 13 ($= 2mi+n$) instead of 100 ($= j$).*
2. *As only V_H and Σ_H are needed from the first (largest) SVD of H, the computational effort in the first stage is cut down as well. More specifically, this means that H can be reduced to a smaller triangular factor first (QR decomposition), which can then be used for subsequent computations (SVD etc.)*

3. *As a consequence, an adaptive identification algorithm can be pieced together, by using QR-updating whenever a new row (a new measurement) is to be worked in, etc. Essentially this boils down to our SVD updating algorithm, cfr. Chapter 2.*
4. *Finally, as A, B, C and D are computed from a reduced set, the identification scheme can be proved to provide consistent results if the number of rows in H tends to infinity and if the input-output measurements are corrupted with additive white measurement noise*[12]*, see Figure 1.4. For that case, it can indeed be shown that the right singular vectors V_H are computed consistently, unlike e.g. Σ_H and U_H [De Moor 1988]. As the system matrices are next computed essentially from V_H only*[13]*, the model estimate is clearly consistent.*

In Chapter 2 the above SVD-based identification algorithm is worked into an adaptive algorithm based on SVD updating, and then used to check the theoretical performance analysis for the SVD updating itself. The QSVD-based counterpart for the coloured noise case is developed in Chapter 4 to illustrate the QSVD updating procedure.

Example:

We end this chapter with a real-life identification example, which is also used in the next chapter. The data come from a **flexible one-link robot**, *a schematic representation of which is exhibited in* **Figure 1.6.** *It consists of a flexible beam with a payload, connected to a flexible torsional beam, which itself is connected to a direct drive brushless DC-motor. The control input is a voltage between -10 Volt and +10 Volt, which is linearly converted by the power supply of the motor into a current. A built-in encoder measures the angular motor position. We refer to [Swevers et al. 1990] for an in depth analysis of this setup. The in- and output data of* **Figure 1.7** *correspond to the input voltage and angle velocity respectively. The sampling rate is 200 Hz, and the data were additionally filtered by means of a 10th order Butterworth low-*

[12] in other words, if the rows of H are subject to independently and identically distributed errors with zero mean and common error covariance matrix equal to the identity matrix, up to a factor of proportionality

[13] The matrix Σ_{11} in the reduced set imposes weights on the different equations. This does not influence the outcome if the set of equations can be solved exactly, which is particularly the case under the assumed conditions.

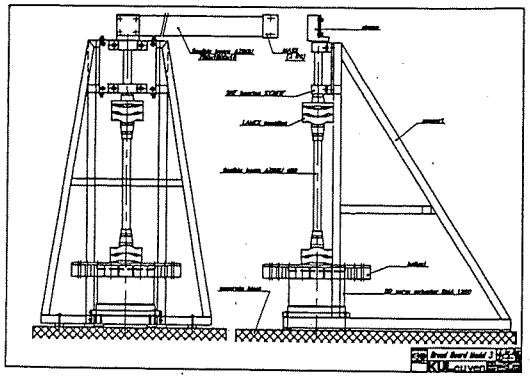


Figure 1.6: *Experiment setup.*

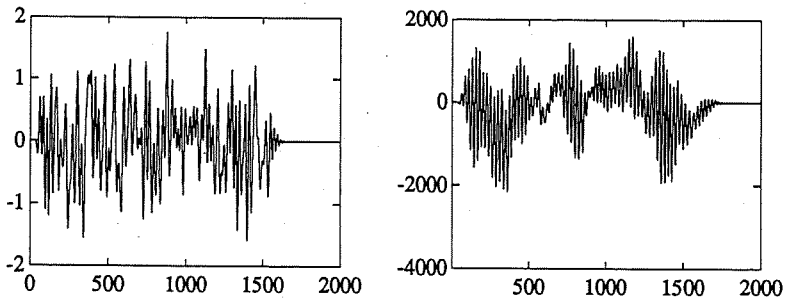


Figure 1.7: *Input-output data record.*

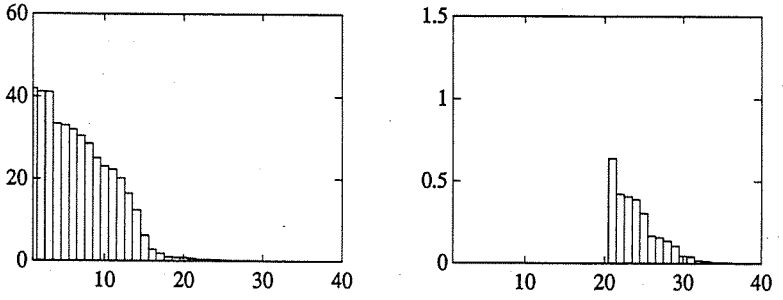


Figure 1.8: *Left: full singular value spectrum. Right: 'order revealing part'.*

pass filter, cut-off frequency 10 Hz. The first part of the data record was used for the identification. The spectrum of the corresponding 40×1000 *block Hankel matrix is exhibited in* **Figure 1.8**, *from which we can decide on the system order being five. The transfer function of the identified 5th order system is shown in* **Figure 1.9**, *which matches earlier results [Swevers et al. 1990]. Finally,* **Figure 1.10** *shows both the measured output and the simulated output, when use is made of the identified model and the measured input. The mismatch at the end of the data record (extrapolation!) corresponds to non-linear friction, but overall it is seen to be reasonably small.*

In Chapter 2, this example will be taken up again, when we consider adaptive identification for time varying systems, where the payload is then subject to sudden changes.

1.7 Conclusions

In this chapter, we briefly introduced the QRD and the SVD. A QR factorization amounts to reducing a given matrix to triangular form by applying orthogonal transformations to the left, whereas an SVD amounts to reducing that same matrix to a diagonal one, by applying orthogonal transformations both to the left and to the right. The significance of these decompositions has been illustrated by a number of theorems and practical applications, in particular by means of a state space identification procedure. Furthermore,

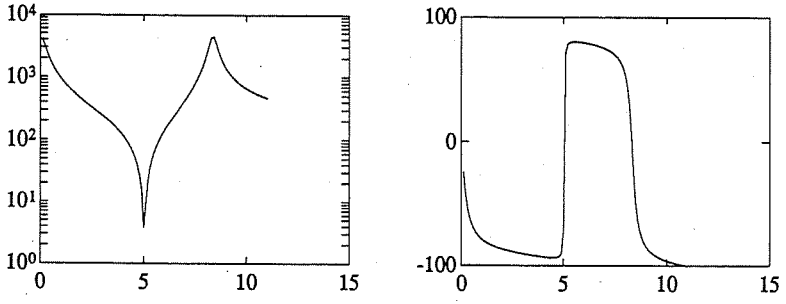


Figure 1.9: *Transfer function of the model. Left: amplitude. Right: phase.*

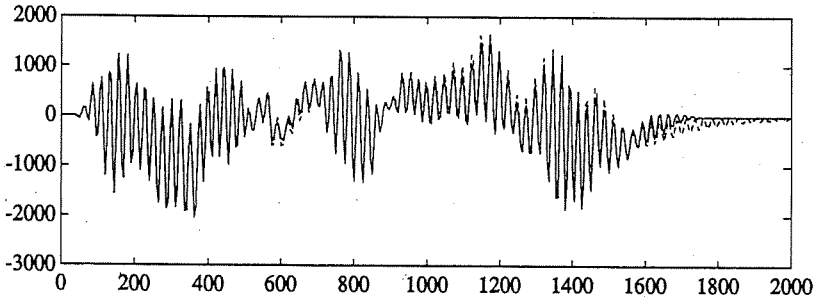


Figure 1.10: *Solid line: measured output. Dashed line: simulated output.*

we discussed algorithms for computing the decompositions, which in the next chapters will be used e.g. for deriving an efficient SVD updating procedure.

Finally, as for the available software, suffice it to say that there are a number of sources of quality software available for computing these matrix decompositions, and for many other linear algebra problems. Let us only mention the NAG library, EISPACK [Smith et al. 1970, Garbow et al. 1972] and LINPACK [Dongarra et al. 1978], and the easy-to-use MATLAB package [Moler et al. 1980].

Chapter 2

SVD updating - algorithms & analysis

In the previous chapter, we have described an algorithm for updating the QRD whenever a new observation vector is added to the problem. Each update takes only $O\{n^2\}$ operations, whereas computing a QRD from scratch normally takes $O\{n^3\}$ operations. Apart from being cheap, the QR updating is stable, amenable to parallel implementation, etc. QR updating therefore has become an established practical tool, for instance for recursive least squares parameter estimation, etc.

The problem of **updating the singular value decomposition** *is even more important in practice, but clearly much more difficult to solve. Unlike QR updating, SVD updating remained largely unused in practice, apparently due to the lack of efficient algorithms. In this chapter, we develop a new updating technique for the SVD. The QR updating is then supplemented with an SVD procedure applied to the triangular factor. Computing an exact SVD after each QR update, would again take $O\{n^3\}$ operations. For real-time applications this is a serious impediment. The computational load is alleviated if only an approximate SVD is computed after each QR update instead. The trade-off is between accuracy and efficiency. As for subspace tracking problems, it is shown how an $O\{n^2\}$ approximate SVD updating algorithm keeps the tracking error down to approximately the time variation in $O\{n\}$ time steps. For applications with slowly time-varying systems this is acceptable. As an example, we consider* **adaptive systems identification,** *where SVD updating is combined with the identification procedure into an adaptive algorithm for model updating. Artificial examples are included, as*

well as a real-life example with data from the flexible one-link robot.

The SVD updating is seen to exhibit all of the desirable properties of mere QR updating. Apart from being cheap and particularly amenable to parallel implementation –see Chapter 3– the SVD updating is also shown to be stable when supplemented with a certain re-orthogonalization procedure, which elegantly combines with the updating scheme.

The SVD updating procedure is developed in section 2.1 and supplemented with a re-orthogonalization scheme in section 2.2. In section 2.3 a performance analysis for subspace tracking is sketched and verified by means of a few identification examples. Finally, error build-up issues are addressed in section 2.4.

While the updating algorithm itself, as well as the performance and error analysis are new, during the preparation of this thesis our attention was drawn to (less complete) results along the same lines in [Ferzali & Proakis 1990], where reference is made to our work also. We should also mention [Callaerts 1989], which got us started on this subject.

2.1 A fast SVD updating algorithm

Updating the SVD after appending a new row, consists in computing the SVD of the modified matrix

$$\underline{A} = \begin{bmatrix} A \\ a^T \end{bmatrix} = \underbrace{\underline{U}}_{(m+1)\times n} \cdot \underbrace{\underline{\Sigma}}_{n\times n} \cdot \underbrace{\underline{V}^T}_{n\times n}.$$

Instead of computing this SVD from scratch, one can make use of the original SVD of A. In on-line applications –e.g. for adaptive systems identification, see below– a new updating is to be performed after each sampling, and the data matrix at time step k is defined in a recursive manner

$$A_{[k]} = \begin{bmatrix} \lambda_{[k]} \cdot A_{[k-1]} \\ a_{[k]}^T \end{bmatrix} = \underbrace{U^\circ_{[k]}}_{k\times n} \cdot \underbrace{\Sigma^\circ_{[k]}}_{n\times n} \cdot \underbrace{V^{\circ T}_{[k]}}_{n\times n}.$$

Factor $\lambda_{[k]}$ $(0 < \lambda_{[k]} \leq 1)$ is a weighting factor, and $a_{[k]}$ is the observation vector at time instant k. For the sake of brevity, we consider only the case where $\lambda_{[k]}$ is a constant λ, although everything can easily be recast for the case where it is time-varying. As later on we will consider approximate decompositions, we use an additional superscript $^\circ$ here to denote exact

decompositions. Finally, in most applications the $U^{\circ}_{[k]}$-matrix –with growing matrix dimensions– need not be computed, and only $V^{\circ}_{[k]}$ and $\Sigma^{\circ}_{[k]}$ are explicitly updated.

Suppose that at a certain time step $k-1$, the $A_{[k-1]}$-matrix is reduced to $R_{[k-1]}$ –upper triangular, almost diagonal [1]– with corresponding matrices $U_{[k-1]}$ and $V_{[k-1]}$, i.e.

$$A_{[k-1]} = U_{[k-1]} \cdot R_{[k-1]} \cdot V^T_{[k-1]}.$$

After appending a new row $a^T_{[k]}$, one has a decomposition of the type

$$\begin{aligned} A_{[k]} &= \begin{bmatrix} \lambda \cdot A_{[k-1]} \\ a^T_{[k]} \end{bmatrix} \\ &= \begin{bmatrix} \boxed{U_{[k-1]}} & \boxed{\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}} \\ \boxed{0 \ \dots \ 0} & \boxed{1} \end{bmatrix} \cdot \begin{bmatrix} \lambda \cdot R_{[k-1]} \\ a^T_{[k]} \cdot V_{[k-1]} \end{bmatrix} \cdot V^T_{[k-1]}. \end{aligned}$$

From this it follows that the updating can be carried out in the following three steps (for infinite precision arithmetic):

1.- matrix-vector multiplication and exponential weighting *where the input vector $a_{[k]}$ is transformed to $\tilde{a}_{[k]}$, by making use of the current $V_{[k-1]}$-matrix:*

$$\begin{aligned} \tilde{R}_{[k-1]} &= \lambda \cdot R_{[k-1]} \\ \tilde{a}^T_{[k]} &= a^T_{[k]} \cdot V_{[k-1]}. \end{aligned}$$

2.- QR-updating *with $\tilde{a}_{[k]}$ in order to restore the triangular structure:*

$$\begin{bmatrix} \hat{R}_{[k]} \\ 0 \end{bmatrix} = Q^T \cdot \begin{bmatrix} \tilde{R}_{[k-1]} \\ \tilde{a}^T_{[k]} \end{bmatrix}$$

[1] see below

such that

$$A_{[k]} = \underbrace{\begin{bmatrix} \boxed{U_{[k-1]}} & \boxed{\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}} \\ \boxed{0 \ \dots \ 0} & \boxed{1} \end{bmatrix}}_{\tilde{U}_{[k]}} \cdot Q_{[k]} \cdot \begin{bmatrix} I_{n \times n} \\ 0 \end{bmatrix} \cdot \hat{R}_{[k]} \cdot V^T_{[k-1]}.$$

Note that the QR-updating does not alter the V-matrix. The U-matrix does change, but it does not have to be stored anyway, as we are only interested in R and V.

3.- SVD steps *in order to obtain a diagonal matrix. This diagonalization procedure consists in applying a sequence of plane rotations as follows (see Chapter 1 for details) :*

$$\begin{aligned} R_{[k]} &\Leftarrow \hat{R}_{[k]} \\ V_{[k]} &\Leftarrow V_{[k-1]} \\ &\textbf{for } l = 1, \dots, r \\ &\quad \textbf{for } i = 1, \dots, n-1 \\ R_{[k]} &\Leftarrow \Theta^T_{[i,l,k]} \cdot R_{[k]} \cdot \Phi_{[i,l,k]} \\ V_{[k]} &\Leftarrow V_{[k]} \cdot \Phi_{[i,l,k]} \\ &\quad \textbf{end} \\ &\textbf{end} \end{aligned}$$

where i is the pivot index, and the matrices $\Theta_{[i,l,k]}$ and $\Phi_{[i,l,k]}$ represent rotations in the $\{i, i+1\}$-plane. The SVD procedure then consists in performing r sequences of $n-1$ such plane rotations, where the pivot index repeatedly takes up all the values $i = 1, \dots, n-1$. Each rotation reduces the off-norm in $R_{[k]}$, and $R_{[k]}$ eventually [2] converges to a diagonal matrix.

With the above procedure, the diagonal structure of R can be restored after each update. With $r = O\{n\}$, the operation count is $O\{n^3\}$ per update. In practice however, it suffices to keep $R_{[k]}$ "close" to a diagonal matrix, instead of completely reducing it to a diagonal matrix in each time step. In some sense [3] $V_{[k]}$ is then "close" to $V^{\circ}_{[k]}$ as well.

[2] with $r = O\{n\}$

[3] see *section 2.3*

Hence, the number of rotations $r(n-1)$ in a certain time step can be fixed, turning an iterative algorithm into a seemingly non-iterative one. For an $O\{n^2\}$ updating algorithm, r should only be $O\{n^0\}$. The crux is then to show how only a few SVD steps after each QR update can restore an acceptable approximation at every stage.

From now on, we make a fairly arbitrary choice and set r equal to 1, so that after each QR update, the pivot index takes up all values $i = 1, \ldots, n-1$ only once. In this case, both the QR updating and the rotations following the update take $O\{n^2\}$ operations, which in particular results in an elegant parallel implementation, cfr. Chapter 3. It should however be stressed that all of our further results can straightforwardly be recast for other choices for r. In section 2.3, the problem is addressed of "how closely" the obtained estimates then approximate the exact decomposition –in particular for the subspace tracking problem– for this particular choice for r.

For the time being, the updating procedure is thus summarized as follows :

Initialization

$$V_{[0]} \Leftarrow I_{n\times n}$$

$$R_{[0]} \Leftarrow O_{n\times n}$$

Loop

for $k = 1, \ldots, \infty$

1. input new measurement vector $a_{[k]}$

$$\begin{aligned} \tilde{a}^T_{[k]} &\Leftarrow a^T_{[k]} \cdot V_{[k-1]} \\ \tilde{R}_{[k-1]} &\Leftarrow \lambda \cdot R_{[k-1]} \end{aligned}$$

2. QR-updating

$$\begin{bmatrix} \hat{R}_{[k]} \\ 0 \end{bmatrix} \Leftarrow Q^T_{[k]} \cdot \begin{bmatrix} \tilde{R}_{[k-1]} \\ \tilde{a}^T_{[k]} \end{bmatrix}$$

$$\begin{aligned} R_{[k]} &\Leftarrow \hat{R}_{[k]} \\ V_{[k]} &\Leftarrow V_{[k-1]} \end{aligned}$$

3. SVD steps

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

$$\begin{aligned} R_{[k]} &\Leftarrow \Theta^T_{[i,k]} \cdot R_{[k]} \cdot \Phi_{[i,k]} \\ V_{[k]} &\Leftarrow V_{[k]} \cdot \Phi_{[i,k]} \end{aligned}$$

end

end

2.2 Re-orthogonalization

In view of round-off accumulation (finite precision arithmetic), the above updating algorithm has one shortcoming. The stored matrix V is iteratively updated by orthogonal column transformations according to

$$V_{[k]} \Leftarrow V_{[k]} \cdot \Phi_{[i,k]}.$$

While $V_{[0]}$ is orthogonal through the initialization, $V_{[k]}$ $(k \gg 1)$ is probably not, due to round-off. The deviation from orthogonality apparently grows linearly with k, as can be verified experimentally. Keeping $V_{[k]}$ close to orthogonal is however crucial for the overall error propagation stability, cfr.section 2.4. Therefore, including some kind of re-orthogonalization procedure is clearly indispensable. An efficient procedure that elegantly combines with the updating scheme can be constructed as follows.

Suppose two vectors x_p and x_q are almost orthonormal, in a sense that

$$\begin{aligned} \|x_p\|_2 &= 1 + O(\epsilon) \\ \|x_q\|_2 &= 1 + O(\epsilon) \\ x_p^T \cdot x_q &= O(\epsilon) \end{aligned}$$

where ϵ is a small number. One can easily verify that a (symmetrized) Gram-Schmidt-like [4] transformation

$$\begin{bmatrix} \boxed{\tilde{x}_p} & \boxed{\tilde{x}_q} \end{bmatrix} = \begin{bmatrix} \boxed{x_p} & \boxed{x_q} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\|x_p\|} & -\frac{x_p^T \cdot x_q}{2} \\ -\frac{x_p^T \cdot x_q}{2} & \frac{1}{\|x_q\|} \end{bmatrix}$$

yields two new vectors, that satisfy

$$\begin{aligned} \|\tilde{x}_p\|_2 &= 1 + O(\epsilon^2) \\ \|\tilde{x}_q\|_2 &= 1 + O(\epsilon^2) \\ \tilde{x}_p^T \cdot \tilde{x}_q &= O(\epsilon^2). \end{aligned}$$

For an $n \times n$ close-to-orthogonal matrix

$$X = \begin{bmatrix} \boxed{x_2} & \boxed{x_2} & \cdots & \boxed{x_n} \end{bmatrix}$$

one can straightforwardly apply the 2×2 transformations in a cyclic manner. One forward sweep consists in computing transformations as follows

for $p = 1, \ldots, n-1$
  for $q = p+1, \ldots, n$

$$\begin{bmatrix} \boxed{x_p} & \boxed{x_q} \end{bmatrix} \leftarrow \begin{bmatrix} \boxed{x_p} & \boxed{x_q} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\|x_p\|} & -\frac{x_p^T \cdot x_q}{2} \\ -\frac{x_p^T \cdot x_q}{2} & \frac{1}{\|x_q\|} \end{bmatrix}$$

  end
end

[4] Note that an exact Gram-Schmidt orthogonalization is computationally somewhat more involved, while on the other hand it yields only marginally better results.

The above algorithm is seen to be a one sided Jacobi-type procedure –with non-unitary transformations–, or two-sided if we consider the effect on $X^T \cdot X$. Furthermore, if $\|X^T \cdot X - I\|_F = O(\epsilon)$ for some small ϵ, the 2×2 transformations are ϵ-close to $I_{2\times 2}$. Due to this, one can prove that the procedure **converges quadratically**, *in other words, that $\|X^T \cdot X - I\|_F = O(\epsilon^2)$ after one sweep. It suffices to copy Wilkinson's proof [Wilkinson 1962] with appropriate substitutions. In Table 2.1 we show the effect of this procedure on a random 10×10 X-matrix –i.e. not even close to an orthogonal one.*

# sweeps	$\|X^T \cdot X - I\|_F$
	1.7335e-01
1	2.0767e-03
2	1.5657e-06
3	0.6442e-13
4	-

Table 2.1 : orthogonalization of a random matrix.

Let us now return to the SVD updating algorithm. If we choose

$$X = V$$

and interlace the above re-orthogonalization with the updating procedure, the former no longer converges quadratically, due to the updating transformations $X \Leftarrow X \cdot \Phi_{[i,k]}$ that clearly change $X^T \cdot X$ as well and thus interfere with the re-orthogonalization. However, if we choose

$$X = V^T$$

in other words if we apply the re-orthogonalization scheme onto the rows of V, the updating transformations do not change $X^T \cdot X$, at least not for (local) infinite precision arithmetic ($X^T \cdot \Phi_{[i,k]} \cdot \Phi_{[i,k]}^T \cdot X = X^T \cdot X$), so that apparently both processes do not interfere. In finite precision, these processes of course do interfere. Both the updating transformations and the re-orthogonalization steps introduce new round-off

errors, which have to be annihilated by the re-orthogonalization itself. On the other hand the re-orthogonalization (slightly) changes the $V_{[k]}$ matrix in an abitrary manner with respect to the data. These issues are analysed in detail in section 4.

The updating and the re-orthogonalization can now be interlaced. If alternately one updating rotation and one re-orthogonalization step are performed, the inner "for"-loop in the updating algorithm reads as follows

$\vdots$

3. SVD steps & re-orthogonalization

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

$$R_{[k]} \Leftarrow \Theta_{[i,k]}^T \cdot R_{[k]} \cdot \Phi_{[i,k]}$$
$$V_{[k]} \Leftarrow T_{[i,k]} \cdot V_{[k]} \cdot \Phi_{[i,k]}$$

end

$\vdots$

where $T_{[i,k]}$ is a re-orthogonalization in the $\left(p_{[i,k]}, q_{[i,k]}\right)$ -plane. Here the number of pairwise re-orthogonalizations equals the number of updating transformations. Again, this is an arbitrary choice, which in some cases might be overly conservative. Further results on the obtained accuracy can however easily be recast for other choices. Furthermore, e.g. in a systolic array implementation (Chapter 3) with a seperate V-array and R-array, the above additional re-orthogonalizations do not introduce any computational overhead, but rather a load-balancing between the two arrays, so that there is no point in reducing the number of re-orthogonalizations.

Finally, in view of efficient (parallel) implementation (see also Chapter 3), the cyclic re-orthogonalization can of course be re-ordered, by making use of additional permutations (outer transformations) and a pipelining of the forward and backward sweep (so that $p_{[i,k]} = i$ and $q_{[i,k]} = i+1$). For the time being however we do not pursue this, as it would considerably complicate our notation in the subsequent analysis.

2.3 SVD updating for subspace tracking

In this section, we analyze the performance of the SVD updating scheme when applied to subspace tracking. It is shown how an approximate updating

scheme, with only $O\{n^2\}$ operations per update instead of $O\{n^3\}$, yields an $O\{n\}$ tracking error. For applications with slowly time-varying systems, this error is sufficiently small. In the above result the trade-off between accuracy and efficiency is clearly illustrated.

First of all, a data model is put forward, which applies to popular signal processing applications, such as harmonic retrieval and direction finding (e.g. MUSIC, ESPRIT [Paulraj et al. 1986]) and systems identification (Chapter 1). For such applications, an SVD step is used to separate a signal subspace from a noise subspace corresponding to certain submatrices of $V_{[k]}^{\circ}$. Further information –e.g. the angles of arrival for ESPRIT or the state space model in systems identification– can then be computed from the knowledge of these submatrices only. As for an approximate SVD –with an approximately (block) diagonal triangular factor $R_{[k]}$– the subspace separation error (tracking error) is shown to be related to the norm of some off-diagonal block of cross terms in the triangular factor. The effect of the SVD procedure on this norm is investigated and finally, all this is applied to the updating problem.

We assume that all computations are performed with **infinite precision**, so that the re-orthogonalization is superfluous. Round-off errors introduce second order effects, which are left out for the sake of simplicity.

Data model

The data model we consider, only assumes that at each time step k the data matrix $A_{[k]}$ has a fixed number d of large singular values

$$\sigma_{i_{[k]}} \qquad i = 1, \ldots, d$$

and a remaining number of small singular values

$$\sigma_{i_{[k]}} \qquad i = d+1, \ldots, n.$$

The ratio

$$SN_{[k]} = \frac{\sigma_{d_{[k]}}}{\sigma_{d+1_{[k]}}}$$

can be interpreted as a **signal-to-noise ratio**, and is assumed to be large, e.g. at least 10 or 100, and lower bounded by a constant SN

$$SN_{[k]} \geq SN.$$

The corresponding submatrices $Vs^{\circ}_{[k]}$ and $Vn^{\circ}_{[k]}$ in $V^{\circ}_{[k]}$ define the **signal subspace** $\mathcal{R}\{Vs^{\circ}_{[k]}\}$ *and the* **noise subspace** $\mathcal{R}\{Vn^{\circ}_{[k]}\}$, *orthogonal to* $\mathcal{R}\{Vs^{\circ}_{[k]}\}$.

As we consider time-varying systems, we need to define a measure of time variation one way or another. In view of the applications at hand, it is obvious to make use of the canonical angles θ_i between $\mathcal{R}\{Vs^{\circ}_{[k-1]}\}$ and $\mathcal{R}\{Vs^{\circ}_{[k]}\}$, the cosines of which are the singular values of a corresponding matrix product [Golub & Van Loan 1983]

$$\cos\theta_i = \sigma_i\{Vs^{\circ T}_{[k-1]} \cdot Vs^{\circ}_{[k]}\}.$$

We then define the **time variation** *from time step $k-1$ to time step k –for a prespecified choice for d– as the distance between the corresponding signal subspaces, which in turn can be defined in terms of the above canonical angles as follows*

$$\begin{aligned} TV_{[k-1]\rightarrow[k]} &\stackrel{def}{=} dist\{\mathcal{R}\{Vs^{\circ}_{[k-1]}\}, \mathcal{R}\{Vs^{\circ}_{[k]}\}\} \\ &\stackrel{def}{=} \sqrt{\sum_{i=1}^{d} \tan^2\theta_i}. \end{aligned}$$

The reason for this will become clear in the sequel.

The adaptive SVD algorithm at each time step stores a triangular factor $R_{[k]}$ of the data matrix $A_{[k]}$, as well as an approximation $V_{[k]}$ for the matrix of right singular vectors. Suppose that at a certain time step k, $R_{[k]}$ and $V_{[k]}$ can be split up as follows

$$\begin{aligned} R_{[k]} &= \begin{bmatrix} Rs_{[k]} & Rsn_{[k]} \\ 0 & Rn_{[k]} \end{bmatrix} \\ V_{[k]} &= \begin{bmatrix} Vs_{[k]} & Vn_{[k]} \end{bmatrix} \end{aligned}$$

where $Rn_{[k]}$ and $Rsn_{[k]}$ are "small". For the time being, we thus assume that at time instant k, the large diagonal elements in $R_{[k]}$ occupy adjacent positions, as this considerably simplifies our further analysis. Later on we return to this.
It is now clear that $Vs_{[k]}$ provides an approximate basis for the signal

subspace. We can then define the **tracking error** *at time step k as follows*

$$TE_{[k]} \stackrel{def}{=} dist\{\mathcal{R}\{Vs_{[k]}\}, \mathcal{R}\{Vs^{\circ}_{[k]}\}\}.$$

Our aim is to derive a useful estimate for the tracking error TE in terms of the time variation TV.

Tracking error bounds

In order to obtain such an estimate, we can make use of a well known property, relating the tracking error to the distance of $R_{[k]}$ from a block diagonal matrix, as follows :

Theorem 1 *[Charlier & Van Dooren 1987, Stewart 1973b]*

If ε denotes the Frobenius norm of the matrix with cross terms

$$\varepsilon = \|Rsn_{[k]}\|_F$$

and δ is the gap between the singular values of $Rs_{[k]}$ and $Rn_{[k]}$

$$\delta = \sigma_{\min}\{Rs_{[k]}\} - \sigma_{\max}\{Rn_{[k]}\}$$

and furthermore if

$$2\varepsilon < \delta$$

then the singular values of $Rn_{[k]}$ and $Rs_{[k]}$ are $\frac{\varepsilon^2}{\delta}$-close to the singular values of $R_{[k]}$, and furthermore there exists a matrix Q, with

$$\|Q\|_F < 2\frac{\varepsilon}{\delta}$$

such that

$$\begin{bmatrix} Vs^{\circ}_{[k]} & Vn^{\circ}_{[k]} \end{bmatrix} = \begin{bmatrix} Vs_{[k]} & Vn_{[k]} \end{bmatrix} \cdot \begin{bmatrix} I & -Q^T \\ Q & I \end{bmatrix} \cdots \cdots \begin{bmatrix} (I+Q^TQ)^{-\frac{1}{2}} & 0 \\ 0 & (I+QQ^T)^{-\frac{1}{2}} \end{bmatrix}$$

The matrix Q reveals the distance between the approximations and the exact subspaces as follows [Stewart 1973b]. The cosines of the canonical angles θ_i between $\mathcal{R}\{Vs_{[k]}\}$ and $\mathcal{R}\{Vs^{\circ}_{[k]}\}$ are the singular values of the corresponding matrix product

$$\begin{aligned}\cos\theta_i &= \sigma_i\{Vs^T_{[k]} \cdot Vs^{\circ}_{[k]}\} \\ &= \sigma_i\{(I + Q^T \cdot Q)^{-\frac{1}{2}}\}.\end{aligned}$$

Let τ_i be the singular values of Q, then

$$\begin{aligned}\cos\theta_i &= (1+\tau_i^2)^{-\frac{1}{2}} \\ \tan\theta_i &= \tau_i.\end{aligned}$$

Thus, the singular values of Q are the tangents of the canonical angles between $\mathcal{R}\{Vs_{[k]}\}$ and $\mathcal{R}\{Vs^{\circ}_{[k]}\}$, and therefore

$$\begin{aligned}TE_{[k]} &= \sqrt{\sum_{i=1}^{d} \tan^2\theta_i} \\ &= \|Q\|_F.\end{aligned}$$

Hence

$$\boxed{TE_{[k]} < 2\tfrac{\varepsilon}{\delta}.}$$

A similar, although slightly sharper bound can be obtained for the case where

$$\begin{aligned}R_{[k]} &= \begin{bmatrix} Rn_{[k]} & Rsn_{[k]} \\ 0 & Rs_{[k]} \end{bmatrix} \\ V_{[k]} &= \begin{bmatrix} Vn_{[k]} & Vs_{[k]} \end{bmatrix}\end{aligned}$$

with $Rs_{[k]}$ in the lower right corner of $R_{[k]}$ this time. In Appendix A, it is proved that for this case

$$\boxed{TE_{[k]} < 2 \cdot \tfrac{\varepsilon}{\delta} \cdot \left(\tfrac{\sigma_{\max}\{Rn_{[k]}\}}{\sigma_{\min}\{Rs_{[k]}\}}\right).}$$

The additional factor is approximately equal to the inverse of the signal-to-noise ratio, defined earlier. In both case, the tracking error at time step k is seen to be inversely proportional to the distance of $R_{[k]}$ from a block diagonal matrix.

Kogbetliantz's algorithm

The tracking error is thus inversely proportional to the norm of the cross terms in $Rsn_{[k]}$. Suppose we would now perform a few sweeps of Kogbetliantz's SVD algorithm –for the time being without any interlaced QR updates !– and then again check the norm of the cross terms. Classical convergence results for Kogbetliantz's algorithm turn out to be useless in this respect [5]. Jacobi-type algorithms are considered extremely fast, but for the general case no estimates are available whatsoever as for the speed of the initial convergence. For our specific subspace separation problem however, it is possible to derive a useful estimate for the reduction of the cross terms in each sweep. In appendix B, the following rule of thumb is obtained.

> *If the cross terms are small compared to the gap and the large diagonal elements are grouped, each* **single sweep** *in Kogbetliantz's SVD algorithm reduces the cross terms by a factor of approximately $\frac{1}{SN}$.*

The derivation of this result relies on a few simplifying assumptions, in order to avoid tedious mathematics and overly conservative results. Our results on the other hand turn out to be practical, and easy to verify by experiments.

[5] Linear convergence bounds are extremely conservative [Forsythe & Henrici 1960, Henrici 1958, Hari & Veselić 1987, Fernando 1989] while ultimate quadratic convergence results [Wilkinson 1962, Paige & Van Dooren 1986, Charlier & Van Dooren 1987] do not apply to the initial convergence, where the off-diagonal elements in $Rs_{[k]}$ can be very large.

Example 1: *For a triangular factor* $R_{[k]}$

$$\begin{array}{ccccccccccc}
3.705 & 0.492 & 0.431 & 1.199 & 0.810 & -.0051 & -.0032 & .0009 & -.0019 & -.0014 \\
 & 3.044 & 1.096 & 1.185 & 2.503 & .0099 & -.0018 & .0038 & -.0093 & -.0061 \\
 & & 2.870 & 1.376 & 0.662 & .0050 & .0026 & -.0024 & .0013 & -.0020 \\
 & & & 1.431 & 1.137 & .0080 & .0027 & .0058 & .0055 & .0008 \\
 & & & & 2.591 & .0017 & -.0006 & -.0021 & .0070 & .0042 \\
 & & & & & .0133 & .0028 & .0031 & .0003 & .0072 \\
 & & & & & & .0130 & .0011 & .0027 & .0004 \\
 & & & & & & & .0057 & .0031 & .0035 \\
 & & & & & & & & .0089 & .0047 \\
 & & & & & & & & & .0056
\end{array}$$

with singular values

$$\begin{aligned}
\sigma \;=\; & 5.4467, 3.5381, 2.7569, 1.9772, 1.1424 && \leftarrow \textit{signal} \\
& 0.0173, 0.0126, 0.0104, 0.0051, 0.0043 && \leftarrow \textit{noise}
\end{aligned}$$

SN is approximately equal to 100. If the SVD procedure were carried out as such, the cross terms would be reduced much faster than could be predicted with the above rule of thumb, as the convergence pretty soon turns into quadratic convergence. If however in each sweep only the cross terms are being annihilated while all other rotations are skipped [6] [7]*, the predicted behavior is much more clearly displayed –***Table 2.2.** *Each sweep clearly reduces the cross terms by a factor* $\frac{1}{SN}$.

# sweeps	ε
	2.3145e-02
1	0.8819e-04
2	0.7311e-06
3	0.5001e-08
4	0.6931e-10
5	0.5369e-12
6	0.7499e-14

Table 2.2 : cross-term reduction per sweep.

[6] cfr. *Part b* in *Appendix B*

[7] Note however that in our updating algorithm, all the rotations are performed, such that the cross-term reduction is indeed much faster. The point is that no (sharper) bound is available for this faster convergence.

From the above rule of thumb, we can infer an estimate for the reduction of the tracking error after a sweep. This latter is known to be bounded by the norm of the cross terms $\|Rsn_{[k]}\|_F$*, but by looking at an example we see that there is clearly more to it:*

Example 1 *-continued: For the same initial triangular factor, the reduction of the tracking error after each single sweep is as follows*

# sweeps	$TE_{[k]}$
	1.0247e-02
1	0.4923e-06
2	0.5670e-06
3	0.4225e-10
4	0.5871e-10
5	0.4572e-14
6	0.6325e-14

Table 2.3 : tracking error reduction per sweep.

The first sweep reduces the tracking error by a factor $\frac{1}{SN^2}$*, while the second one keeps it at roughly the same level, etc.*

The above behavior is readily explained by the error bounds of the previous section. Each sweep changes the configuration, so that different bounds for the tracking error apply before and after the sweep. Briefly, we can conclude that

> *Each* **double sweep** *in Kogbetliantz's SVD algorithm reduces the tracking error by a factor of approximately* $\frac{1}{SN^2}$.

Example 2: *Similar to the experiment in Example 1, we checked the reduction of the tracking error per double sweep (for three successive double sweeps), for different triangular matrices. The matrix dimension n ranges from 10 through 50, while the singular value spectra were chosen to be*

$$\begin{aligned}
\sigma \;=\; & \frac{n}{2}, \frac{n}{2}-1, \ldots, 2, 1 \\
& \frac{1}{SN}, \frac{\frac{n}{2}-1}{\frac{n}{2}.SN}, \ldots, \frac{1}{\frac{n}{2}.SN}
\end{aligned}$$

for SN= 10 and SN= 100. For all cases, the reduction factor is seen to be approximately equal to $\frac{1}{SN^2}$. Apparently, the matrix dimension has only little influence on this –**Table 2.4.**

	$n = 10$	$n = 20$	$n = 50$
SN= 10	$2.8100e-02$	$2.0356e-02$	$1.8847e-02$
	$8.2947e-05$	$8.6065e-05$	$5.8207e-05$
	$3.4249e-07$	$5.6355e-07$	$4.2585e-07$
	$1.4534e-09$	$3.7391e-09$	$3.5933e-09$
SN= 100	$2.0213e-02$	$2.01087-02$	$1.9357e-02$
	$4.7201e-07$	$5.8845e-07$	$8.9264e-07$
	$2.3465e-11$	$3.5926e-11$	$7.4129e-11$
	$1.1801e-15$	$2.6241e-15$	$6.9677e-15$

Table 2.4 : tracking error reduction per double sweep.

Subspace tracking

Let us now return to the SVD updating algorithm, applied to subspace tracking. In the adaptive algorithm, a pipelined double sweep of n times $n-1$ rotations is interlaced with n QR updates, one after each series of $n-1$ rotations. If all these updates were performed after the double sweep, one would end up with an inequality as follows

$$dist\{\mathcal{R}\{Vs_{[k+n]}\}, \mathcal{R}\{Vs^{\circ}_{[k+n]}\}\} \leq \frac{1}{SN^2} \cdot dist\{\mathcal{R}\{Vs_{[k]}\}, \mathcal{R}\{Vs^{\circ}_{[k]}\}\} + dist\{\mathcal{R}\{Vs^{\circ}_{[k]}\}, \mathcal{R}\{Vs^{\circ}_{[k+n]}\}\}.$$

The "$\leq$"-sign is due to the fact that the different terms in the right hand side can partially eliminate each other. The first term in the right hand side, corresponds to the reduction of the subspace separation error in a double sweep (n time steps). The second term corresponds to the time variation from time instant k to time instant $k+n$.

If the QR updates are interlaced in the double sweep, the subspace separation error is expected to be even smaller, as time variations can then immediately be taken into account and corrected correspondingly. Although one can easily think of set ups where the above statement does not hold, in general it is seen to be confirmed by simulations, see below for examples. The above inequality can therefore be assumed to provide a reasonable estimate for the SVD updating scheme as well.

Furthermore, as we assumed that SN is fairly large (e.g 100), it follows that

$$\underbrace{dist\{\mathcal{R}\{Vs_{[k+n]}\}, \mathcal{R}\{Vs^{\circ}_{[k+n]}\}\}}_{TE_{[k+n]}} \leq \underbrace{dist\{\mathcal{R}\{Vs^{\circ}_{[k]}\}, \mathcal{R}\{Vs^{\circ}_{[k+n]}\}\}}_{TV_{[k]\rightarrow[k+n]}}.$$

In other words we can conclude that

The tracking error is bounded by the time variation in n time steps.

$$TE_{[k+n]} \leq TV_{[k]\rightarrow[k+n]}.$$

Hence

$$\text{“}TE_{[k]} = O\{n\}.\text{”}$$

REMARKS

1. *The above results were derived for the case where only one sequence of $n-1$ SVD rotations is performed after each QR update, in other words $r = 1$ in section 2.2. For other choices for r, one would end up with*

   $$TE_{[k+\frac{n}{r}]} \leq TV_{[k]\rightarrow[k+\frac{n}{r}]}$$

   as a double sweep is then performed in $\frac{n}{r}$ time steps. In other words, the tracking error is inversely proportional to r.

2. *Throughout the computations –see appendix B– we have assumed that the small diagonal elements in the triangular factor occupy adjacent positions. Notice that a similar assumption had to be made for the proof of the quadratic convergence for matrices with pathologically close or repeated singular values [Wilkinson 1962, Charlier & Van Dooren 1987]. Even in an adaptive scheme, obtaining such a set up is merely a matter of careful initialization* [8].

[8] once the small elements are grouped, they do not change their "affiliation", at least not for slowly time varying systems

Furthermore, by performing simulations, it is observed that even when the large and small diagonal elements are not grouped, the tracking error is still bounded by the time variation in n time steps. In other words, this latter bound seems to be conservative enough, so that it applies to the disadvantageous "non-grouped" case as well.

3. *The above derivation particularly applies to cases where the signal-to-noise ratio is fairly large, e.g 100 or more. In practice however, it is observed that the obtained rules of thumb deliver fairly reasonable estimates for even smaller values of SN, e.g. 10, see also Table 2.4.*

4. *For (extremely) large values of the matrix dimension, it can be expected that the performance slightly declines, much like it has been observed that the number of necessary sweeps in a classical SVD procedure is slightly larger for large matrices, e.g. $n > 100$. In these cases, one can easily e.g. double or triple the number of rotations after each QR update accordingly. Strictly speaking, the complexity of the updating algorithm could then become $O\{n^{2+\alpha}\}$, where α is (much) smaller than 1.*

Example 3: *The following example from systems theory illustrates the above results.*

The SVD-based identification algorithm of Chapter 1 can be worked into an adaptive algorithm for model updating, based on SVD updating. The starting point was a block Hankel matrix with I/O data as follows

$$\underbrace{H}_{j\times(2mi+2li)} = \begin{bmatrix} z_{[1]}^T & z_{[2]}^T & \cdots & z_{[2i]}^T \\ z_{[2]}^T & z_{[3]}^T & \cdots & z_{[2i+1]}^T \\ \vdots & \vdots & & \vdots \\ z_{[j]}^T & z_{[j+1]}^T & \cdots & z_{[j+2i-1]}^T \end{bmatrix}.$$

In an adaptive setting, new measurements are available in each time step, so that in each time step a new row can be appended to H. In-

cluding exponential weighting then results in a time-varying H-matrix as follows

$$H_{[k]} = W_{[k]} \cdot \begin{bmatrix} z_{[1]}^T & z_{[2]}^T & \cdots & z_{[2i]}^T \\ z_{[2]}^T & z_{[3]}^T & \cdots & z_{[2i+1]}^T \\ \vdots & \vdots & & \vdots \\ z_{[k-2i+1]}^T & z_{[k-2i+2]}^T & \cdots & z_{[k]}^T \end{bmatrix}$$

$$W_{[k]} = diag\{\lambda^{k-2i}, \lambda^{k-2i-1}, \ldots, \lambda^1, \lambda^0\}.$$

Note that for a time invariant linear system with exact I/O data, the rank of H is exactly $2mi + n$. The exponential weighting does not change this, neither does it change the null space of H, which is used for further computations. In other words, the weighting in principle does not introduce any errors. For (slowly) time-varying systems on the other hand, with a good choice of λ, the 'signal rank' of the weighted block Hankel matrix is equal to $2mi+n$, so that we can separate a signal subspace from a noise subspace. As only $\mathcal{R}\{V\mathrm{n}_{[k]}^{\circ}\}$ (and $\Sigma_{[k]}$) is needed for the state space identification, we can obviously make use of our SVD updating procedure. In each time step, approximations for $\mathcal{R}\{V\mathrm{n}_{[k]}^{\circ}\}$ and $\Sigma_{[k]}$ are computed, which can then be used in the identification procedure. In this way, an approximate model is obtained in each time step. A complete algorithmic description is given in [Moonen et al. 1989a].

Suppose we are given a simple first order time-varying system, with state space equations

$$\begin{aligned} x_{[k+1]} &= .8 \cdot \cos(\frac{2\pi}{2000}k) \cdot x_{[k]} + u_{[k]} \\ y_{[k]} &= x_{[k]}. \end{aligned}$$

For every arbitrary input sequence, the output can be computed accordingly, by making use of the state space equations. The state space model at a time instant k can then approximately be recovered from the input-output data, by making use of the adaptive identification procedure. We performed an adaptive identification, where use was made of either an exact SVD scheme –with a complete computation of

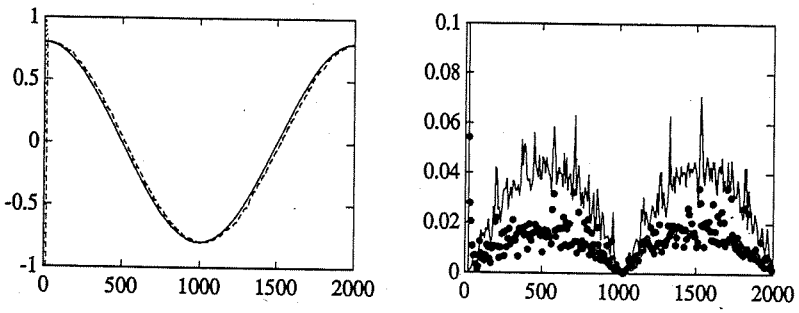


Figure 2.1: *Left: original and identified system pole vs.time. Right: time variation (-) and tracking error (•), $\lambda = 1 - 2^{-5}$.*

the SVD at each time instant– or an adaptive SVD scheme with e.g. $n-1$ rotations after each QR update. The parameter i was set equal to 3, so that the matrix size equals 12, with a 5-dimensional kernel.

Figure 2.1(left) *shows the original system pole $0.8 \cdot \cos(\frac{2\pi}{2000}k)$ –solid line– together with the identified pole both for the scheme with an exact SVD –dashed line– and for the adaptive scheme –dotted line, almost coinciding with the dashed line. The exponential weighting factor λ was set equal to $1 - 2^{-5}$.* **Figure 2.1(right)** *shows the time variation in $n = 12$ time steps $TV_{[k]\rightarrow[k+12]}$ –solid line– together with the tracking error at each time instant $TE_{[k]}$ –dotted line. Clearly the latter in general remains smaller than the former, confirming the above rule of thumb. Finally notice that the time variation of the data nicely reflects the time variation of the underlying system, viz. the system pole.*

In **Figure 2.2,** *the same quantities have been plotted for a different choice for the weighting factor $\lambda = 1 - 2^{-8}$. A different choice for λ is hardly influences the time variation and the approximation error here. As for the identified pole, the exponential weighting is seen to introduce a kind of time delay, which increases when λ approaches 1. The adaptive scheme however still delivers quite the same system pole as the exact scheme.*

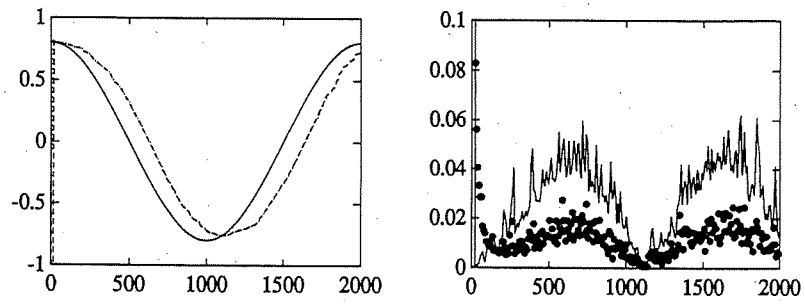


Figure 2.2: *Left: original and identified system pole vs.time. Right: time variation (-) and tracking error (•), $\lambda = 1 - 2^{-8}$.*

Example 4:

Finally, a similar experiment can be performed with the **flexible one-link robot.** *Each time the payload is changed, the dynamics change correspondingly, so that the system is indeed time-varying. The data record we used (from time instant -1000 through +1000), shows a payload change at time instant 0.*

For a start, we used 'cleaned up' output data, coming from simulations, such as the dashed line in Figure 1.10. In other words the data from time instant -1000 through 0 as well as the data from 0 through +1000 each correspond to a (noiseless) linear system. For an adaptive identification, with $i = 10$, $n = 5$ and $\lambda = 1 - 2^{-6}$, both the time-variation and the tracking error are shown in **Figure 2.3** *(left). The time-variation decreases to zero, due to the exponential weighting. The tracking error is indeed found to be of the same order of magnitude. The order revealing part of the spectrum (right-hand figure) has 5 nonzero singular values at $t = 0$. From $t = 700$ on, we again have 5 dominant singular values. Strictly speaking, our analysis thus applies from time instant 700 through 1000, where the system is 'slowly time-varying'.*

*With real output data, the picture is much messier -***Figure 2.4.** *The system order is not well defined as there are no dominant singular*

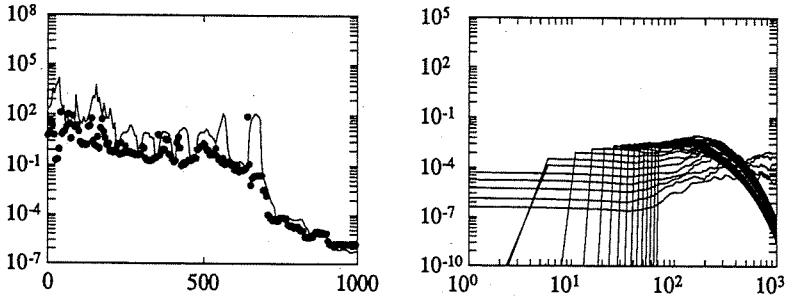


Figure 2.3: *Left: time variation (-) and tracking error (•) vs. time. Right: singular values (order revealing part) vs. time,* $\lambda = 1 - 2^{-6}$.

values (right-hand figure). Also, the time-variation does not decrease to zero, due to various fluctuations, noise, etc. Still, the tracking error is again found to be of the same order of magnitude as the time-variation.

2.4 Error analysis

Until now the important issue of round-off error build-up has not been addressed. The updating algorithm in principle keeps on running as long as new data vectors are fed in. Clearly, when **finite precision** *arithmetic is used, round-off error build-up is to be avoided at any cost. For QR updating, performing orthogonal transformations in combination with exponential weighting –de-emphasizing older data as well as accumulated errors– is found to be sufficiently stable in practice. For the SVD updating problem however, the orthogonal matrix V is stored as well and iteratively updated by orthogonal column transformations, along with the triangular factor. In this section, an error analysis is performed, proving that the algorithm is stable on condition that the stored V-matrix is kept close to orthogonal. Furthermore, it is shown how the re-orthogonalization procedure keeps the deviation from orthogonality sufficiently small in order to guarantee stability of the overall updating scheme.*

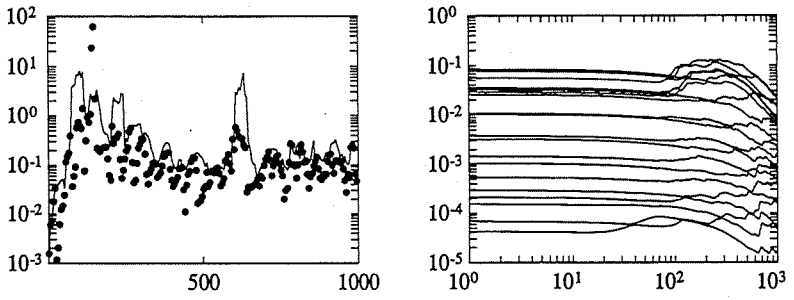


Figure 2.4: *Left: time variation (-) and tracking error (•) vs. time. Right: singular values (order revealing part) vs. time,* $\lambda = 1 - 2^{-8}$

Preliminaries

In the previous section, we analyzed the performance of the updating algorithm in infinite precision arithmetic, resulting in an upper bound for the distance between $Vs_{[k]}$ *and* $Vs^{\circ}_{[k]}$. *In finite precision arithmetic, there are a few sources of additional error. Apart from round-off of course, there is also the re-orthogonalization scheme. Both processes e.g. change the V-matrix in an arbitrary manner with respect to the original data. As we are only interested in the right singular vectors, together with the singular values, we define a relevant error matrix in this respect as follows*

$$\Delta_{[k]} \;=\; A^T_{[k]} \cdot A_{[k]} - (R_{[k]} \cdot V^T_{[k]})^T \cdot (R_{[k]} \cdot V^T_{[k]}).$$

This error matrix tells how far the information stored in $R_{[k]} \cdot V^T_{[k]}$ *has drifted off from the original data. If* $\Delta_{[k]}$ *is small, the singular values of* $R_{[k]}$ *are close to those of* $A_{[k]}$. *Furthermore, for large values of SN, the signal subspace of* $R_{[k]} \cdot V^T_{[k]}$ *is then also close to the signal subspace of* $A_{[k]}$.

In the sequel, upper bounds for the propagation of $\Delta_{[k]}$ *are derived, resulting in a first order difference equation*

$$\boxed{\Delta_{[k]} = \lambda^2 \cdot \Delta_{[k-1]} + \delta E_{[k]}}$$

where λ is the exponential weighting factor ($\lambda < 1$). As long as $V_{[k]}$ is close to an orthogonal matrix, $\delta E_{[k]}$ contains only bounded local errors, independent of $\Delta_{[k-1]}$. In other words, the norm

$$\|\Delta I_{[k]}\|_F \stackrel{\text{def}}{=} \|V_{[k]} \cdot V_{[k]}^T - I\|_F$$

(where I is the identity matrix) should be kept small, in order to guarantee stability of the error propagation. Therefore, we first derive an estimate for the above norm, by investigating the re-orthogonalization scheme of section 2.2. By making use of this, we then derive the above error propagation formula, showing that the overall updating procedure is stable.

An estimate for $\|V_{[k]} \cdot V_{[k]}^T - I\|_F$

In finite precision arithmetic both the updating and the re-orthogonalization steps introduce new errors in the stored $V_{[k]}$-matrix. On the other hand, the re-orthogonalization itself annihilates –up to machine precision– accumulated errors. If the number of re-orthogonalizations in the updating scheme is chosen to be equal to the number of SVD rotations, one double sweep in the re-orthogonalization procedure is performed after each n time steps. If we let $\Phi_{[ac]}$ and $T_{[ac]}$ denote the accumulated right and left transformations applied to V in time steps k through $k+n-1$, we have

$$\begin{aligned} V_{[k+n-1]} &= \mathbf{fl}(T_{[ac]} \cdot V_{[k-1]} \cdot \Phi_{[ac]}) \\ &= T_{[ac]} \cdot V_{[k-1]} \cdot \Phi_{[ac]} + \delta V_{[ac]} \end{aligned}$$

where $\delta V_{[ac]}$ is an input of local errors in these time steps, and $\mathbf{fl}(\cdot)$ refers to the computer result after a sequence of transformations (in the right order). For a first order analysis, the re-orthogonalizations can be considered as orthogonal transformations, so that Gentleman's analysis [Gentleman 1975] applies, resulting in an approximate upper bound for the local errors on V in one double sweep [9] *:*

$$\begin{aligned} \|\delta V_{[ac]}\|_F &\leq (4n-4) \cdot k_V \cdot \epsilon \cdot (1 + k_V \epsilon)^{4n-5} \cdot \|V\|_F \\ &\leq 4n \cdot k_V \cdot \epsilon \cdot \|V\|_F + O\{\epsilon^2\} \end{aligned}$$

[9] The double sweep –with SVD steps and re-orthogonalizations– consists of $4n-4$ different stages, where a *stage* consists of the simultaneous application of disjoint transformations [Gentleman 1975]

where ϵ is the largest number such that $\mathbf{fl}(1+\epsilon) = 1$ (relative machine precision) and k_V is a constant depending on the specific implementation of both the Givens rotations and the re-orthogonalizations.

From the formula for $V_{[k+n-1]}$, we derive a formal description for the error build-up process as follows

$$\begin{aligned} V_{[k+n-1]} \cdot V_{[k+n-1]}^T &= T_{[ac]} \cdot V_{[k-1]} \cdot \Phi_{[ac]} \cdot \Phi_{[ac]}^T \cdot V_{[k-1]}^T \cdot T_{[ac]}^T \\ &\quad + T_{[ac]} \cdot V_{[k-1]} \cdot \Phi_{[ac]} \cdot \delta V_{[ac]}^T \\ &\quad + \delta V_{[ac]} \cdot \Phi_{[ac]}^T \cdot V_{[k-1]}^T \cdot T_{[ac]}^T \\ &\quad + O\{\epsilon^2\}. \end{aligned}$$

Hence

$$\underbrace{\|V_{[k+n-1]} \cdot V_{[k+n-1]}^T - I\|_F}_{\|\Delta I_{[k+n-1]}\|_F} \leq \underbrace{\|T_{[ac]} \cdot V_{[k-1]} \cdot V_{[k-1]}^T \cdot T_{[ac]}^T - I\|_F}_{\eta_{[ac]} \cdot \|\Delta I_{[k-1]}\|_F} \cdots$$
$$\cdots + 2 \cdot \|\delta V_{[ac]}^T\|_F + O\{\epsilon^2\}$$

where $\eta_{[ac]}$ is a factor describing the effect of the re-orthogonalization steps.

As long as $\|\Delta I_{[k-1]}\|_F$ is small ($< \epsilon^{\frac{1}{4}}$), it is reduced to machine precision by the double sweep in the re-orthogonalization scheme. The input of local errors however interferes with this re-orthogonalization, and this introduces additional errors of the same order of magnitude.

$$\|\Delta I_{[k+n-1]}\|_F \leq \underbrace{\eta_{[ac]} \cdot \|\Delta I_{[k-1]}\|_F}_{O\{n\epsilon\}} + 2 \cdot \|\delta V_{[ac]}\|_F + O\{\epsilon^2\}.$$

With $\|V\|_F \simeq n^{\frac{1}{2}}$, this results in an estimate for $\|\Delta I_{[k]}\|_F$ for all k.

$$\boxed{\|\Delta I_{[k+n-1]}\|_F \simeq k_I \cdot n^{\frac{3}{2}} \cdot \epsilon.}$$

where e.g. $k_I = 8k_V$ for large values of n.

Error propagation

First of all, for the sake of conciseness, let us assume that there exists an upper bound $\|R\|_F$ such that for all k

$$\begin{aligned}\|R_{[k]}\|_F \le \sqrt{n}\|R_{[k]}\|_2 &\le \|R\|_F\\ \|\tilde{R}_{[k]}\|_F \le \sqrt{n}\|\tilde{R}_{[k]}\|_2 &\le \|R\|_F\\ \sqrt{n}\|a_{[k]}\|_2 &\le \|R\|_F.\end{aligned}$$

The error matrix at time step k, viz. $\Delta_{[k]}$, is then computed from the error matrix $\Delta_{[k-1]}$ at time step $k-1$ as follows. The results of this error analysis are framed on page 71.

In a **first step**, *appending a new row together with an exponential weighting, is described as follows*

$$\begin{bmatrix}\lambda\cdot R_{[k-1]}\cdot V_{[k-1]}^T\\ a_{[k]}^T\end{bmatrix} = \begin{bmatrix}\lambda\cdot R_{[k-1]}\\ a_{[k]}^T\cdot V_{[k-1]}\end{bmatrix}\cdot V_{[k-1]}^T + \underbrace{\begin{bmatrix}0\\ -a_{[k]}^T\cdot\Delta I_{[k-1]}\end{bmatrix}}_{\delta E_{[k]}^1}$$

where $\delta E_{[k]}^1$ is an "algorithmic" error due to $V_{[k-1]}$ not being orthogonal. Here we use the upper bound for ΔI of the previous section:

$$\begin{aligned}\|\delta E_{[k]}^1\|_2 &\le \|\Delta I_{[k-1]}\|_2\cdot\|a_{[k]}\|_2\\ &\le k_I n\epsilon\cdot\|R\|_F.\end{aligned}$$

Additional round-off errors are introduced by the explicit computation of

$$\begin{aligned}\tilde{R}_{[k-1]} &= \mathbf{fl}\{\lambda\cdot R_{[k-1]}\}\\ &= \lambda\cdot R_{[k-1]} + \delta\tilde{R}_{[k-1]}\\ \tilde{a}_{[k]}^T &= \mathbf{fl}\{a_{[k]}^T\cdot V_{[k-1]}\}\\ &= a_{[k]}^T\cdot V_{[k-1]} + \delta\tilde{a}_{[k]}^T.\end{aligned}$$

Substituting this in the above equation,

$$\begin{bmatrix}\lambda\cdot R_{[k-1]}\cdot V_{[k-1]}^T\\ a_{[k]}^T\end{bmatrix} = \begin{bmatrix}\tilde{R}_{[k-1]}\\ \tilde{a}_{[k]}^T\end{bmatrix}\cdot V_{[k-1]}^T + \delta E_{[k]}^1\cdots + \underbrace{\begin{bmatrix}-\delta\tilde{R}_{[k-1]}\\ 0\end{bmatrix}\cdot V_{[k-1]}^T}_{\delta E_{[k]}^2} + \underbrace{\begin{bmatrix}0\\ -\delta\tilde{a}_{[k]}^T\end{bmatrix}\cdot V_{[k-1]}^T}_{\delta E_{[k]}^3}$$

with first order upper bounds (see [Wilkinson 1965])

$$\begin{aligned}\|\delta E_{[k]}^2\|_2 &\le \epsilon\cdot\|\tilde{R}_{[k-1]}\|_F\\ &\le \epsilon\cdot\|R\|_F\\ \|\delta E_{[k]}^3\|_2 &\le n\epsilon\cdot\|a_{[k]}\|_2\cdot\|V_{[k-1]}\|_F\\ &\le n\epsilon\|R\|_F + O\{\epsilon^2\}.\end{aligned}$$

In a **second step**, *orthogonal row transformations are performed for the QR-update, introducing a round-off error $\delta\hat{R}_{[k]}$. The rotation angles are chosen such that the last row of the computer result transforms to zero*

$$\begin{aligned}\begin{bmatrix}\hat{R}_{[k]}\\ 0\end{bmatrix} &= \mathbf{fl}\{Q_{[k]}^T\cdot\begin{bmatrix}\tilde{R}_{[k-1]}\\ \tilde{a}_{[k]}^T\end{bmatrix}\}\\ &= Q_{[k]}^T\cdot\begin{bmatrix}\tilde{R}_{[k]}\\ \tilde{a}_{[k]}^T\end{bmatrix} + \delta\hat{R}_{[k]}.\end{aligned}$$

By making use of this, we obtain

$$\begin{aligned}&\begin{bmatrix}\lambda\cdot R_{[k-1]}\cdot V_{[k-1]}^T\\ a_{[k]}^T\end{bmatrix}\\ &= Q_{[k]}\cdot(Q_{[k]}^T\cdot\begin{bmatrix}\tilde{R}_{[k-1]}\\ \tilde{a}_{[k]}^T\end{bmatrix})\cdot V_{[k-1]}^T + \delta E_{[k]}^1 + \delta E_{[k]}^2 + \delta E_{[k]}^3\\ &= Q_{[k]}\cdot\begin{bmatrix}\hat{R}_{[k]}\\ 0\end{bmatrix}\cdot V_{[k-1]}^T + \delta E_{[k]}^1 + \delta E_{[k]}^2 + \delta E_{[k]}^3\cdots\\ &\quad\cdots\underbrace{-Q_{[k]}\cdot\delta\hat{R}_{[k]}\cdot V_{[k-1]}^T}_{\delta E_{[k]}^4}.\end{aligned}$$

Following the error analysis in [Gentleman 1975],

$$\begin{aligned}\|\delta E^4_{[k]}\|_2 &\leq k_G\epsilon \cdot n(1+k_G\epsilon)^{n-1} \cdot \|\hat{R}_{[k]}\|_F \\ &\leq k_G\epsilon \cdot n \cdot \|R\|_F + O\{\epsilon^2\}\end{aligned}$$

where k_G is a constant depending on the specific implementation of the Givens rotations.

In a **third step** *finally (SVD steps + re-orthogonalizations), a sequence of left and right transformations is applied. If we summarize these into left transformations $\Theta_{[k]}$ and $T_{[k]}$, and a right transformation $\Phi_{[k]}$, we can proceed as follows, introducing round-off errors $\delta R_{[k]}$ and $\delta V_{[k]}$*

$$\begin{aligned}R_{[k]} &= \mathrm{fl}\{\Theta^T_{[k]} \cdot \hat{R}_{[k]} \cdot \Phi_{[k]}\} \\ &= \Theta^T_{[k]} \cdot \hat{R}_{[k]} \cdot \Phi_{[k]} + \delta R_{[k]} \\ V_{[k]} &= \mathrm{fl}\{T_{[k]} \cdot V_{[k-1]} \cdot \Phi_{[k]}\} \\ &= T_{[k]} \cdot V_{[k-1]} \cdot \Phi_{[k]} + \delta V_{[k]} \\ &= V_{[k-1]} \cdot \Phi_{[k]} + (T_{[k]} - I) \cdot V_{[k-1]} \cdot \Phi_{[k]} + \delta V_{[k]}\end{aligned}$$

and

$$\begin{aligned}&\begin{bmatrix} \lambda \cdot R_{[k-1]} \cdot V^T_{[k-1]} \\ a^T_{[k]} \end{bmatrix} \\ &= Q_{[k]} \cdot \begin{bmatrix} \Theta_{[k]} \cdot (\Theta^T_{[k]} \cdot \hat{R}_{[k]} \cdot \Phi_{[k]}) \\ 0 \end{bmatrix} \cdot (\Phi^T_{[k]} \cdot V^T_{[k-1]}) \cdots \\ &\quad \cdots + \delta E^1_{[k]} + \delta E^2_{[k]} + \delta E^3_{[k]} + \delta E^4_{[k]} \\ &= Q_{[k]} \cdot \left(\begin{bmatrix} \Theta_{[k]} \cdot R_{[k]} \\ 0 \end{bmatrix} + \begin{bmatrix} -\Theta_{[k]} \cdot \delta R_{[k]} \\ 0 \end{bmatrix} \right) \\ &\qquad \cdot (V^T_{[k]} - \delta V^T_{[k]} - \Phi^T_{[k]} V^T_{[k-1]} (T_{[k]} - I)^T) \cdots \\ &\quad \cdots + \delta E^1_{[k]} + \delta E^2_{[k]} + \delta E^3_{[k]} + \delta E^4_{[k]} \\ &= Q_{[k]} \cdot \begin{bmatrix} \Theta_{[k]} \cdot R_{[k]} \\ 0 \end{bmatrix} \cdot V^T_{[k]} \cdots \\ &\quad \cdots + \delta E^1_{[k]} + \delta E^2_{[k]} + \delta E^3_{[k]} + \delta E^4_{[k]} \cdots\end{aligned}$$

$$\cdots + \underbrace{Q_{[k]} \cdot \begin{bmatrix} -\Theta_{[k]} \cdot \delta R_{[k]} \\ 0 \end{bmatrix} \cdot V^T_{[k]}}_{\delta E^5_{[k]}} + \underbrace{Q_{[k]} \cdot \begin{bmatrix} -\Theta_{[k]} \cdot R_{[k]} \\ 0 \end{bmatrix} \cdot \delta V^T_{[k]}}_{\delta E^6_{[k]}} \cdots$$

$$\cdots + \underbrace{Q_{[k]} \cdot \begin{bmatrix} -\Theta_{[k]} \cdot R_{[k]} \\ 0 \end{bmatrix} \cdot \Phi^T_{[k]} \cdot V^T_{[k-1]} \cdot (T_{[k]} - I)^T}_{\delta E^7_{[k]}} + O\{\epsilon^2\}.$$

Again applying the error analysis in [Gentleman 1975] results in

$$\begin{aligned}\|\delta E^5_{[k]}\|_2 &\leq k_G\epsilon \cdot (2n-2)(1+k_G\epsilon)^{2n-3} \cdot \|\hat{R}_{[k]}\|_F \\ &\leq k_G\epsilon \cdot (2n-2)\|R\|_F + O\{\epsilon^2\} \\ \|\delta E^6_{[k]}\|_2 &\leq k_G\epsilon \cdot (2n-2)(1+k_G\epsilon)^{2n-3} \cdot \|V_{[k]}\|_F \cdot \|\hat{R}_{[k]}\|_2 \\ &\leq k_G\epsilon \cdot (2n-2)\sqrt{n} \cdot \|\hat{R}_{[k]}\|_2 + O\{\epsilon^2\} \\ &\leq k_G\epsilon \cdot (2n-2)\|R\|_F + O\{\epsilon^2\}.\end{aligned}$$

As for $\delta E^7_{[k]}$, we estimate an upper bound as follows

$$\|\delta E^7_{[k]}\|_2 \leq \|T_{[k]} - I\|_F \cdot \|R_{[k]}\|_2.$$

Note that $\delta E^7_{[k]}$ represents an additional error which is introduced by the re-orthogonalization scheme. The re-orthogonalization indeed changes the V-matrix in an arbitrary manner with respect to the original data, and therefore contributes to $\Delta_{[k]}$. One can easily check that $\|T_{[k]} - I\|_F$ must have a similar upper bound as $\|V_{[k]}V^T_{[k]} - I\|_F$, so that finally

$$\begin{aligned}\|\delta E^7_{[k]}\|_2 &\leq k_T \cdot n\sqrt{n}\epsilon \cdot \|R_{[k]}\|_2 \\ &\leq k_T \cdot n\epsilon\|R\|_F.\end{aligned}$$

Adding all the above upper bounds, we obtain

$$\begin{aligned}\begin{bmatrix} \lambda \cdot R_{[k-1]} \cdot V^T_{[k-1]} \\ a^T_{[k]} \end{bmatrix} &= Q_{[k]} \cdot \begin{bmatrix} \Theta_{[k]} \cdot R_{[k]} \\ 0 \end{bmatrix} \cdot V^T_{[k]} \cdots \\ &\quad \cdots + \underbrace{\delta E^1_{[k]} + \delta E^2_{[k]} + \delta E^3_{[k]} + \delta E^4_{[k]} + \delta E^5_{[k]} + \delta E^6_{[k]} + \delta E^7_{[k]}}_{\delta E^{1\to 7}_{[k]}}\end{aligned}$$

with

$$\|\delta E_{[k]}^{1\rightarrow 7}\|_2 \leq (k_1 \cdot n + k_2)\epsilon \cdot \|R\|_F.$$

This is worked into an error propagation formula for $\Delta_{[k]}$ as follows. Multiplying the left- and right-hand sides with their transpose results in

$$\begin{aligned}\lambda^2 \cdot (R_{[k-1]} \cdot V_{[k-1]}^T)^T \cdot (R_{[k-1]} \cdot V_{[k-1]}^T) + a_{[k]} \cdot a_{[k]}^T \\ = (R_{[k]} \cdot V_{[k]}^T)^T \cdot (R_{[k]} \cdot V_{[k]}^T) + \delta E_{[k]}\end{aligned}$$

where

$$\begin{aligned}\delta E_{[k]} &= (\delta E_{[k]}^{1\rightarrow 7})^T \cdot (Q_{[k]} \cdot \begin{bmatrix} \Theta_{[k]} \cdot R_{[k]} \\ 0 \end{bmatrix} \cdot V_{[k]}^T) \cdots \\ &\quad \cdots + (Q_{[k]} \cdot \begin{bmatrix} \Theta_{[k]} \cdot R_{[k]} \\ 0 \end{bmatrix} \cdot V_{[k]}^T)^T \cdot (\delta E_{[k]}^{1\rightarrow 7}) + O\{\epsilon^2\}\end{aligned}$$

with an upper bound

$$\begin{aligned}\|\delta E_{[k]}\|_F &\leq 2\|R_{[k]}\|_F \cdot \|\delta E_{[k]}^{1\rightarrow 7}\|_2 + O\{\epsilon^2\} \\ &\leq 2(k_1 \cdot n + k_2)\epsilon \cdot \|R\|_F^2 + O\{\epsilon^2\}.\end{aligned}$$

Substituting the definition of $\Delta_{[.]}$ results in

$$\begin{aligned}\lambda^2 \cdot A_{[k-1]}^T \cdot A_{[k-1]} - \lambda^2 \cdot \Delta_{[k-1]} + a_{[k]} \cdot a_{[k]}^T \\ = (R_{[k]} \cdot V_{[k]}^T)^T \cdot (R_{[k]} \cdot V_{[k]}^T) + \delta E_{[k]}\end{aligned}$$

$$\begin{aligned}A_{[k]}^T \cdot A_{[k]} - \lambda^2 \cdot \Delta_{[k-1]} \\ = (R_{[k]} \cdot V_{[k]}^T)^T \cdot (R_{[k]} \cdot V_{[k]}^T) + \delta E_{[k]}\end{aligned}$$

and finally

$$\boxed{\Delta_{[k]} = \lambda^2 \cdot \Delta_{[k-1]} + \delta E_{[k]}}$$

or if we use norms

$$\begin{aligned}\|\Delta_{[k]}\|_F &\leq \lambda^2 \cdot \|\Delta_{[k-1]}\|_F + \|\delta E_{[k]}\|_F \\ &\leq \lambda^2 \cdot \|\Delta_{[k-1]}\|_F + 2(k_1 \cdot n + k_2)\epsilon \cdot \|R\|_F^2 + O\{\epsilon^2\}.\end{aligned}$$

The first term, $\lambda^2 \cdot \Delta_{[k-1]}$ represents the error propagation, which is stable as $\lambda < 1$. The second term is an upper bound for local errors. If we assume that the weighting factor is constant, we finally obtain for all k a first order upper bound as follows

$$\boxed{\|\Delta_{[k]}\|_F \leq \frac{2(k_1 \cdot n + k_2)}{1-\lambda^2} \cdot \|R\|_F^2 \cdot \epsilon.}$$

In conclusion, the overall SVD updating scheme is found to be stable, if an exponential weighting is applied, with weighting factor $\lambda < 1$, and if a re-orthogonalization procedure is included, which keeps the stored V-matrix close to orthogonal. The obtained upper bound for the error $\|\Delta_{[k]}\|_F$ is clearly overly conservative, as it consists of an accumulation of several worst case upper bounds.

2.5 Conclusion

An SVD updating procedure has been constructed as a combination of i) matrix-vector multiplications, ii) QR updating and iii) a Jacobi-type SVD algorithm applied to the triangular factor. This procedure is fast –with only $O\{n^2\}$ operations per update instead of $O\{n^3\}$– easy to code, and furthermore perfectly stable when supplemented with a re-orthogonalization scheme, even when a potential infinite number of updates are needed.

As for subspace tracking problems, it has been shown how a fairly accurate approximation is restored after each updating, with an $O\{n\}$ tracking error. we have illustrated the theory with a few artificial examples, as well as a real-life example with data from the flexible one-link robot.

Finally, the updating procedure lends itself well to parallel implementation. In the next chapter, a systolic array for this updating algorithm is developed.

Chapter 3

Systolic arrays and square root free implementation

A systolic system is a network of processors which rhythmically compute and pass data through the system. Physiologists use the word "systole" to refer to the rhythmically recurrent contraction of the heart and arteries which pulses blood through the body. In a systolic computing system, the function of a processor is analogous to that of the heart. Every processor regularly pumps data in and out, each time performing some short computation, so that a regular flow of data is kept up in the network.

adapted from
H.T. Kung and C.E. Leiserson
systolic arrays for VLSI, 1979.

In Chapter 2, we have constructed an approximate SVD updating scheme, a combination of a Jacobi-type SVD procedure with QR updates and matrix-vector multiplications, possibly supplemented with a certain re-orthogonalization scheme. Each update takes $O\{n^2\}$ operations. For high throughput applications such as image and radar processing, this may still be excessive. Parallel computing then provides a proper way to meet the extreme throughput requirements.

For example, one can think of the well known Gentleman-Kung QR

updating array [Gentleman & Kung 1981]. Each QR update takes $O\{n^2\}$ operations. The array consists of $O\{n^2\}$ processors, and subsequent updates are pipelined such that a new update can start off in each time step. Briefly, we say that an $O\{n^0\}$ throughput is obtained with $O\{n^2\}$ parallelism for $O\{n^2\}$ computational complexity.

In this chapter, we develop a similar systolic array for SVD updating, also with $O\{n^2\}$ processors. Unfortunately, the different computational steps here[1] cannot be pipelined in an efficient manner if one uses conventional mapping techniques. This would then result in a throughput, which is only $O\{n^{-1}\}$, due to the existence of a critical path.

It is then shown how an $O\{n^0\}$ throughput is obtained, if one allows for a limited number of supplementary computations. In our array, different computational fronts run in different directions. Whenever two such fronts meet, intermediate results have to be updated, hence the additional computations. However, in this way we are able to fully exploit the available parallelism. The array is complicated at first sight, but extremely efficient and perfectly modular and repetitive. The idea of designing this type of arrays –with crossing computational fronts– is apparently new.

Furthermore, while Jacobi-type matrix algorithms are mostly mapped on orthogonally connected processor arrays, in this chapter, an alternative partitioning is described, resulting in a hexagonally connected processor grid. This partitioning is seen to be over four times more efficient, compared to the original configuration.

Finally, a main computational bottle-neck for such an array appears to be the computation of the rotation parameters in the boundary processors, with explicit computation of a number of square roots. It is shown how a square root free implementation is obtained by combining modified Givens rotations with approximate SVD schemes. This is an extension of the well known square root free QR updating.

In section 3.1, commonly known arrays for the QRD updating and the SVD are briefly reviewed. In section 3.2, a first SVD updating array is developed for the easy case, where corrective re-orthogonalizations are left out. In section 3.3, it is then shown how to incorporate these re-orthogonalizations. Finally a square root free implementation is derived in section 3.4.

[1] QR update, SVD and matrix-vector multiplications

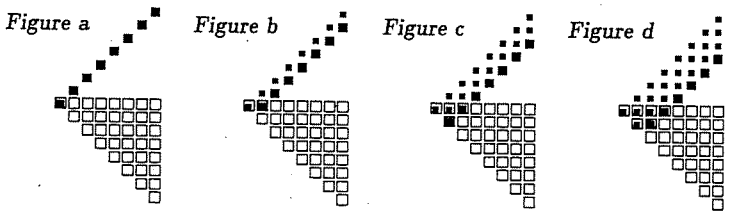


Figure 3.1: *Gentleman-Kung array.*

3.1 QRD and SVD arrays

In this section, we briefly review both the Gentleman-Kung QR updating array [Gentleman & Kung 1981] and the SVD array of [Luk 1986a], as later on these are combined into an SVD updating array. Furthermore we briefly comment on the specific processor configuration for the SVD array, and describe an alternative hexagonally connected configuration.

The Gentleman-Kung QR updating array

A QR updating is performed by applying a sequence of orthogonal (Givens) transformations –see Chapter 1. Gentleman and Kung have shown how subsequent updates can be pipelined on a triangular array of orthogonally connected processors [Gentleman & Kung 1981]. This array, which we refer to as the Gentleman-Kung array, is exhibited in **Figure 3.1.**

Each processor in the triarray contains one element of the R-factor. The data vectors are fed in in a skewed fashion as indicated, one at a time, and are propagated downwards while being changed by successive row transformations. The filled squares in Figure 3.1a correspond to the first input vector. The small ones in Figures 3.1b-c-d correspond to subsequent inputs. Processors on the main diagonal generate elementary orthogonal row transformations to annihilate a component of the input vector and propagate the transformation parameters to the right (Description 1). Off-diagonal processors apply the received row transformations. The resulting input vector component is passed on

downwards while the rotation parameters are passed on to the right (Description 2). The rotation parameters thus run across the array from left to right, one stage at a time. The data skew ensures that the same delay is imposed on the input vectors, so that the algorithm is executed correctly.

DESCRIPTION 1 :
BOUNDARY CELL R-FACTOR

Input x

$x \leftarrow x_{in}$

Compute Rotation Angle ψ

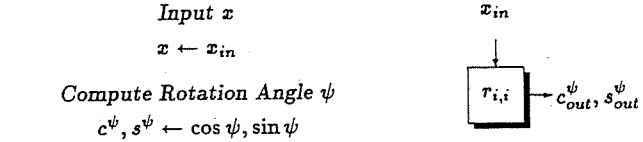
$c^{\psi}, s^{\psi} \leftarrow \cos\psi, \sin\psi$

Apply Transformation

$r_{i,i} \leftarrow \sqrt{r_{i,i}^2 + x^2}$

Propagate Transformation Parameters

$c^{\psi}_{out}, s^{\psi}_{out} \leftarrow c^{\psi}, s^{\psi}$



DESCRIPTION 2 :
INTERNAL CELL R-FACTOR

Input x

$x \leftarrow x_{in}$

Input Transformation Parameters

$c^{\psi}, s^{\psi} \leftarrow c^{\psi}_{in}, s^{\psi}_{in}$

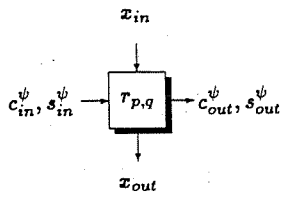
Apply Transformations

$$\begin{bmatrix} r_{p,q} \\ x \end{bmatrix} \leftarrow \begin{bmatrix} c^{\psi} & s^{\psi} \\ -s^{\psi} & c^{\psi} \end{bmatrix} \begin{bmatrix} r_{p,q} \\ x \end{bmatrix}$$

$c^{\psi}_{out}, s^{\psi}_{out} \leftarrow c^{\psi}, s^{\psi}$

Output x

$x_{out} \leftarrow x$

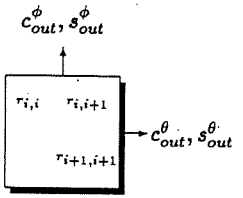


SVD array

Figure 3.2 *shows the SVD array of [Luk 1986a]. Let us for the time*

being assume that one processor is associated to each 2 × 2 block of contiguous elements. Later on, we return to this and comment on the actual processor configuration.

Processor elements on the main diagonal perform 2×2 SVD's –with implicit permutations– annihilating the available off-diagonal elements. In Figure 3.2a, all 2 × 2 SVD's corresponding to odd pivot indices are performed in parallel. Row transformation parameters are passed on to the right, while column transformation parameters are passed on upwards (Description 3). Off-diagonal processors only apply these transformations and propagate them to the blocks next outward (Description 4). In Figure 3.2c, this first set of rotations has moved far enough to allow the next set of rotations to be generated, this time corresponding to even pivot indices. In Figure 3.2e, the next odd sequence again starts off, etc. Note that column transformations are also propagated through the upper square part, containing the V-matrix [2] *(Description 5).*



DESCRIPTION 3 :

BOUNDARY CELL *R*-FACTOR

Compute Rotation Angles ϕ, θ

$$c^{\phi}, s^{\phi} \leftarrow \cos\phi, \sin\phi$$
$$c^{\theta}, s^{\theta} \leftarrow \cos\theta, \sin\theta$$

Apply Transformation

$$r_{i,i} \leftarrow r_{i+1,i+1}\frac{c^{\theta}}{c^{\phi}}$$
$$r_{i+1,i+1} \leftarrow r_{i,i}\frac{c^{\phi}}{c^{\theta}}$$
$$r_{i,i+1} \leftarrow 0$$

Propagate Transformation Parameters

$$c^{\phi}_{out}, s^{\phi}_{out} \leftarrow c^{\phi}, s^{\phi}$$
$$c^{\theta}_{out}, s^{\theta}_{out} \leftarrow c^{\theta}, s^{\theta}$$

[2] V's first row in the top row, etc.

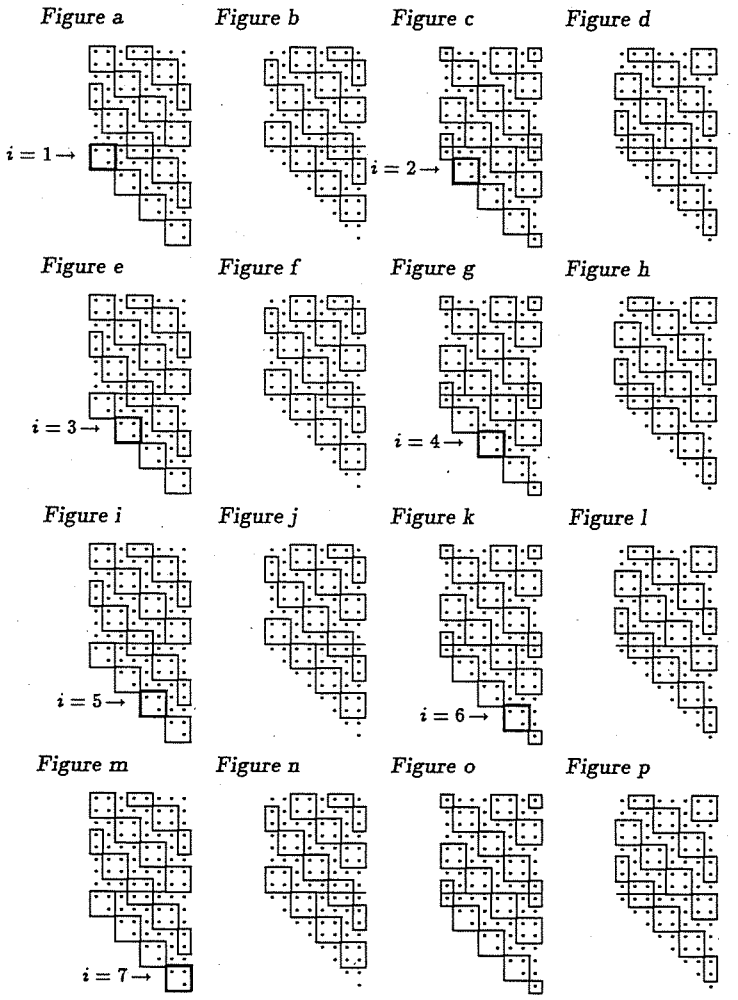
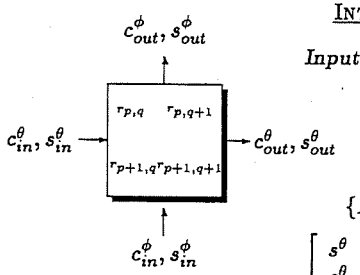


Figure 3.2: *SVD array.*

DESCRIPTION 4 :

INTERNAL CELL R-FACTOR



Input Transformation Parameters

$$c^{\phi}, s^{\phi} \leftarrow c^{\phi}_{in}, s^{\phi}_{in}$$
$$c^{\theta}, s^{\theta} \leftarrow c^{\theta}_{in}, s^{\theta}_{in}$$

Apply Transformation

$$\{R\}_{p:p+1,q:q+1} \leftarrow \begin{bmatrix} s^{\theta} & c^{\theta} \\ c^{\theta} & -s^{\theta} \end{bmatrix} \cdot \{R\}_{p:p+1,q:q+1} \cdot \begin{bmatrix} s^{\phi} & c^{\phi} \\ c^{\phi} & -s^{\phi} \end{bmatrix}$$

Propagate Transformation Parameters

$$c^{\phi}_{out}, s^{\phi}_{out} \leftarrow c^{\phi}, s^{\phi}$$
$$c^{\theta}_{out}, s^{\theta}_{out} \leftarrow c^{\theta}, s^{\theta}$$

DESCRIPTION 5 :

INTERNAL CELL V-MATRIX

Input Transformation Parameters

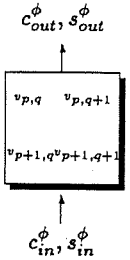
$$c^{\phi}, s^{\phi} \leftarrow c^{\phi}_{in}, s^{\phi}_{in}$$

Apply Transformation

$$\{V\}_{p:p+1,q:q+1} \leftarrow \{V\}_{p:p+1,q:q+1} \cdot \begin{bmatrix} s^{\phi} & c^{\phi} \\ c^{\phi} & -s^{\phi} \end{bmatrix}$$

Propagate Transformation Parameters

$$c^{\phi}_{out}, s^{\phi}_{out} \leftarrow c^{\phi}, s^{\phi}$$



In this parallel implementation, off-diagonal elements with odd and even row numbers are annihilated in an alternative fashion (odd-even ordering). However, it is readily verified that such an odd-even ordering corresponds to a cyclic by rows or columns ordering, apart from a different start-up phase [Luk & Park 1989, Stewart 1985]. The 2×2 SVD's that are performed in parallel on the main diagonal can indeed

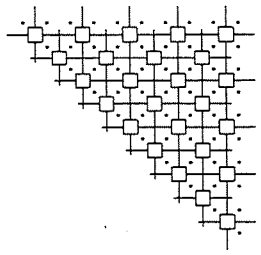


Figure 3.3: *Orthogonally connected processor array.*

be thought of as corresponding to different **pipelined sequences of $n-1$ rotations**, *where in each sequence the pivot index successively takes up the values $i = 1, \ldots, n-1$. Exactly n such sequences are known to correspond to a double sweep in a cyclic ordering, see Chapter 1. One such sequence starts off in Figure 3.2a and is indicated explicitly. The next one starts off in Figure 3.2e, etc. This simple observation is crucial for the further developments. In section 3.2, it is shown how to intertwine QR updates in between two such sequences.*

Processor configurations for the SVD array

From this description, it seems natural to think of a computational network that consists of an array of storage cells –dots in **Figure 3.3**– *each corresponding to one matrix element, together with a grid of processors –squares in Figure 3.3– corresponding to 2×2 subblocks [Stewart 1985, O'Leary & Stewart 1985]* [3]*. Each processor has access to four surrounding storage cells. The number of processors is roughly half the number of matrix elements. The processors are interconnected horizontally and vertically in order to allow the transformations to pass through the matrix. Each processor is active during only one of the phases a, b, c or d* [4]*, and idle for roughly 75% of the time.*

[3] We only show the triangular part of the array in Figures 3.3 through 3.5. The square part is similar of course, but left out for the sake of conciseness.

[4] Phase a corresponds to Figure 3.2a-e-i-m, phase b corresponds to Figure 3.2.b-f-j-n, etc.

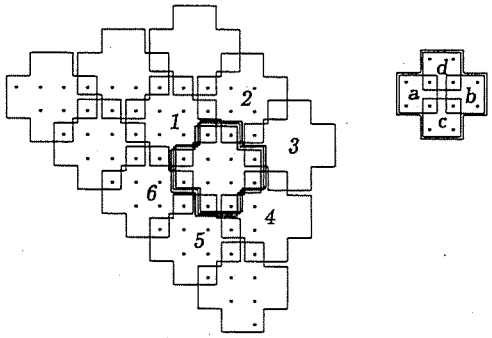


Figure 3.4: *Alternative partitioning.*

From this, it follows that alternatively four contiguous processors can be merged into one new processor. Remarkably enough, this merger completely changes the outlook of the array [Moonen & Vandewalle 1990a]. **Figure 3.4** *exhibits how processors can be merged, and indicates "activity regions" for the new processors. Each processor now has access to twelve storage cells. It alternately performs operations on the left-most, right-most, lower and upper 2×2 block in its activity region, corresponding to phases a-b-c-d respectively. As each processor has 4 matrix elements "of its own", plus 8 elements shared with one other processor, the number of processors equals one eighth times the number of matrix elements* [5].

From Figure 3.4, it follows that each processor has six neighbours, with which it exchanges either column and/or row transformations. The resulting grid is seen to be hexagonally connected, with a principal axis parallel to the main diagonal, **Figure 3.5**. *Apart from the boundary processors which associate with fewer storage cells, all processors are active throughout, so that the overall efficiency is roughly 100%. Furthermore, due to the fact that half of the matrix elements are associated with only one processor, the corresponding storage cells*

[5] which is indeed four times less than the number of processors in the original configuration.

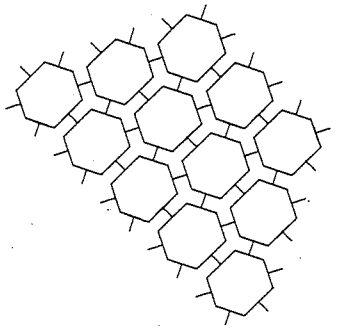


Figure 3.5: *Hexagonally connected processor array.*

actually represent internal memory. The number of external memory accesses is therefore roughly halved. In conclusion –and somewhat contradictory– an increased throughput is attained with roughly four times less hardware.

3.2 A systolic array for SVD updating

In this section, we show how the SVD updating algorithm –for the time being without re-orthogonalizations– can be mapped onto a systolic array. The SVD updating array is similar to the triangular SVD array in [Luk 1986a] where the SVD diagonalization and a preliminary QR factorization are performed on the same array, but clearly much more involved as the diagonalization process and the QR updating are now interlaced. Also, from the algorithmic description it follows that the V-matrix should be stored as well. Hence we have to provide for an additional square array, which furthermore performs the matrix-vector products $a^T \cdot V$. The pipelining of the different computational steps appears to be a major problem. Still, it is shown how a perfect pipelining can be obtained at the cost of a small computational overhead. Furthermore, it is shown how certain output vectors can be generated at each time step, with only very few additional computations.

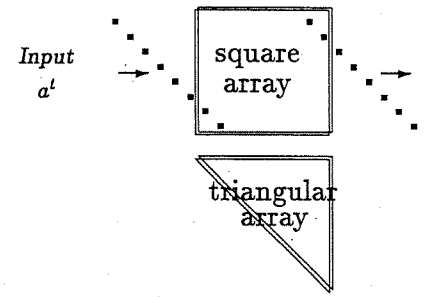


Figure 3.6: *Preview SVD updating array.*

Figure 3.6 *gives an overview of the array. New data vectors are continuously fed in, to the left-hand side of the array. The matrix-vector product is computed in the square part, and the resulting vector is passed on to the triangular array that performs the QR updating and the SVD diagonalization. Output vectors are flushed in the opposite direction, and become available at the right-hand side of the array. All these operations can be carried out simultaneously as is detailed next. The correctness of this array has been verified by software simulation.*

A modified Gentleman-Kung QR updating array

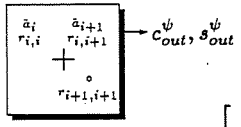
In view of combining things, the Gentleman-Kung array should first be matched to the SVD array. **Figure 3.7** *shows a modified QR updating array. While all operations remain unaltered, the pipelining is somewhat different, so that the data vectors are now propagated through the array in a slightly different manner, see Descriptions 6 and 7* [6]*. Note that each 2×2 block combines its first row with the available input vector components, and pushes the resulting data vector components one step downwards. The first update starts off in Figure 3.7a –large filled boxes– the second one in 3.7e –smaller filled boxes– etc. Furthermore, each update is seen to correspond to a sequence of rotations where the pivot index takes up the values $i = 1, \ldots, n$. Both the processor configuration and the pipelining thus turn out to be the same as for the SVD array.*

[6] In these descriptions, ○ and ◇ are memory cells which are filled by the updated elements of the $\tilde{a}$ -vector.

DESCRIPTION 6 :
BOUNDARY CELL R-FACTOR

Compute Rotation Angle ψ

$$c^\psi, s^\psi \leftarrow \cos\psi, \sin\psi$$



Apply Transformation

$$r_{i,i} \leftarrow \sqrt{r_{i,i}^2 + \tilde{a}_i^2}$$

$$\begin{bmatrix} r_{i,i+1} \\ \tilde{a}_{i+1} \end{bmatrix} \leftarrow \begin{bmatrix} c^\psi & s^\psi \\ -s^\psi & c^\psi \end{bmatrix} \begin{bmatrix} r_{i,i+1} \\ \tilde{a}_{i+1} \end{bmatrix}$$

Propagate $\tilde{a}_{i+1}$

$$\circ \leftarrow \tilde{a}_{i+1}$$

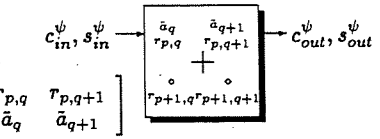
Propagate Transformation Parameters

$$c^\psi_{out}, s^\psi_{out} \leftarrow c^\psi, s^\psi$$

DESCRIPTION 7 :
INTERNAL CELL R-FACTOR

Input Transformation Parameters

$$c^\psi, s^\psi \leftarrow c^\psi_{in}, s^\psi_{in}$$



Apply Transformation

$$\begin{bmatrix} r_{p,q} & r_{p,q+1} \\ \tilde{a}_q & \tilde{a}_{q+1} \end{bmatrix} \leftarrow \begin{bmatrix} c^\psi & s^\psi \\ -s^\psi & c^\psi \end{bmatrix} \begin{bmatrix} r_{p,q} & r_{p,q+1} \\ \tilde{a}_q & \tilde{a}_{q+1} \end{bmatrix}$$

Propagate $\tilde{a}_q, \tilde{a}_{q+1}$

$$\circ, \diamond \leftarrow \tilde{a}_q, \tilde{a}_{q+1}$$

Propagate Transformation Parameters

$$c^\psi_{out}, s^\psi_{out} \leftarrow c^\psi, s^\psi$$

Matrix-vector product

The matrix-vector product $a^T \cdot V$ can be combined with the SVD steps as depicted in **Figure 3.8a-g.** *The data vectors a^T are fed in*

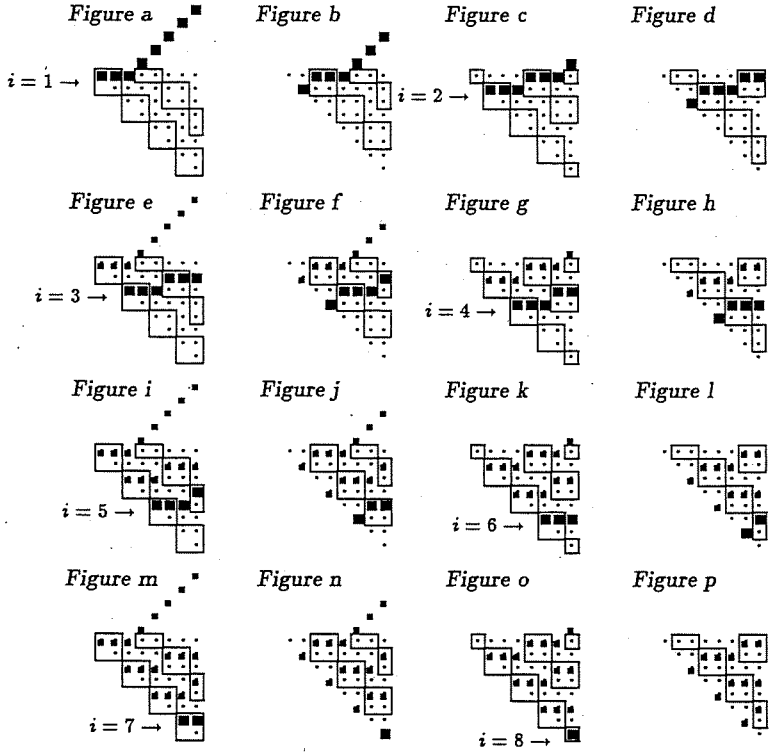


Figure 3.7: *Modified Gentleman-Kung array.*

in a skewed fashion as indicated, and are propagated to the right, in between two rotation fronts corresponding to the SVD diagonalization (frames). Each processor receives a-components from its left neighbour, and intermediate results from its lower neighbour. The intermediate results are then updated and passed on to the upper neighbour, while the a-component is passed on to the right. The resulting matrix-vector product becomes available at the top end of the square array.

It should be stressed that a consistent matrix-vector product $a \cdot V$ **can** *indeed* **only** *be formed in between two SVD rotation fronts. This restriction is imposed by the concurrent SVD operations. More particularly this results in the following :*

First *the propagation of the SVD rotation fronts dictates the direction in which a matrix-vector product can be formed. The resulting vector $\tilde{a}$ thus inevitably becomes available at the top end of the array, while it should be fed in into the triangular array for the subsequent QR update. The $\tilde{a}$-components are therefore reflected at the top end and propagated downwards, towards the triangular array, Figure 3.8e-p. The downward propagation of an $\tilde{a}$-vector is then carried out in exactly the same manner as the propagation in the modified Gentleman-Kung array, see also Figure 3.7.*

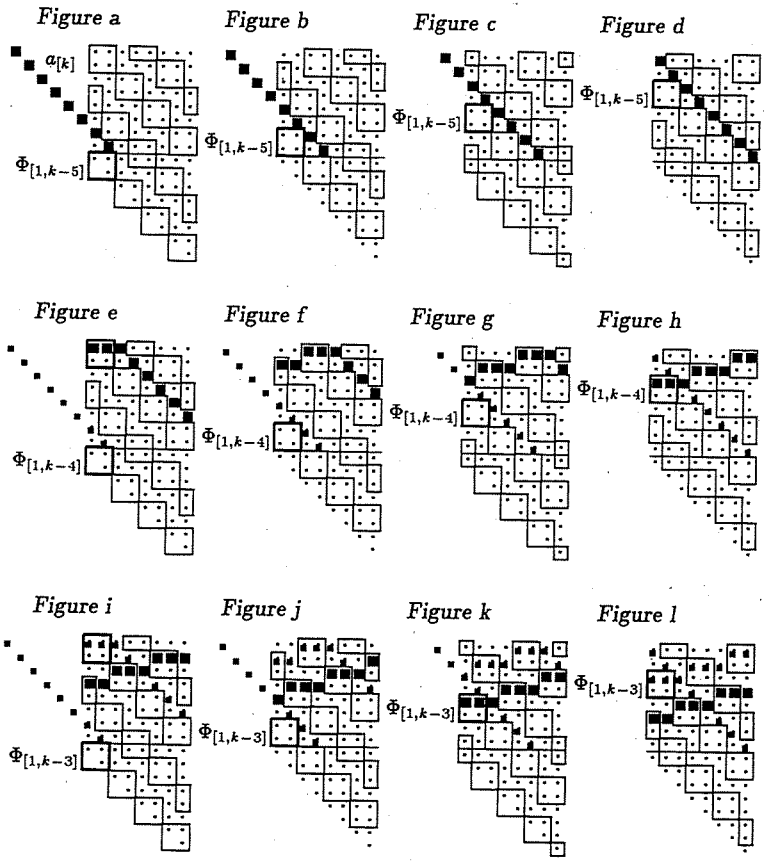
Second *the V-matrix that is used for computing $a^T \cdot V$ is in fact some older version of V, which we term $V_{[\mathfrak{k}]}$ [7]. For a specific input vector $a_{[k]}$, this $V_{[\mathfrak{k}]}$ equals $V_{[k-1]}$ up to a number of column transformations, such that*

$$V_{[k-1]} = V_{[\mathfrak{k}]} \cdot \Phi_{[\mathfrak{k}]}$$

where $\Phi_{[\mathfrak{k}]}$ denotes the accumulated lacking column transformations. In order to obtain $\tilde{a}_{[k]}$, it is necessary to apply $\Phi_{[\mathfrak{k}]}$ to the computed matrix-vector product

$$\begin{aligned} \tilde{a}_{[k]} &= a_{[k]}^T \cdot V_{[k-1]} \\ &= \underbrace{a_{[k]}^T \cdot V_{[\mathfrak{k}]}}_{\tilde{a}_{[\mathfrak{k}]}^T} \cdot \Phi_{[\mathfrak{k}]}. \end{aligned}$$

[7] One can check that it is not possible to substitute a specific time index for the "$\mathfrak{k}$", in other words that the $V_{[\mathfrak{k}]}$ does not correspond to any $V_{[k]}$ in the sequential description.



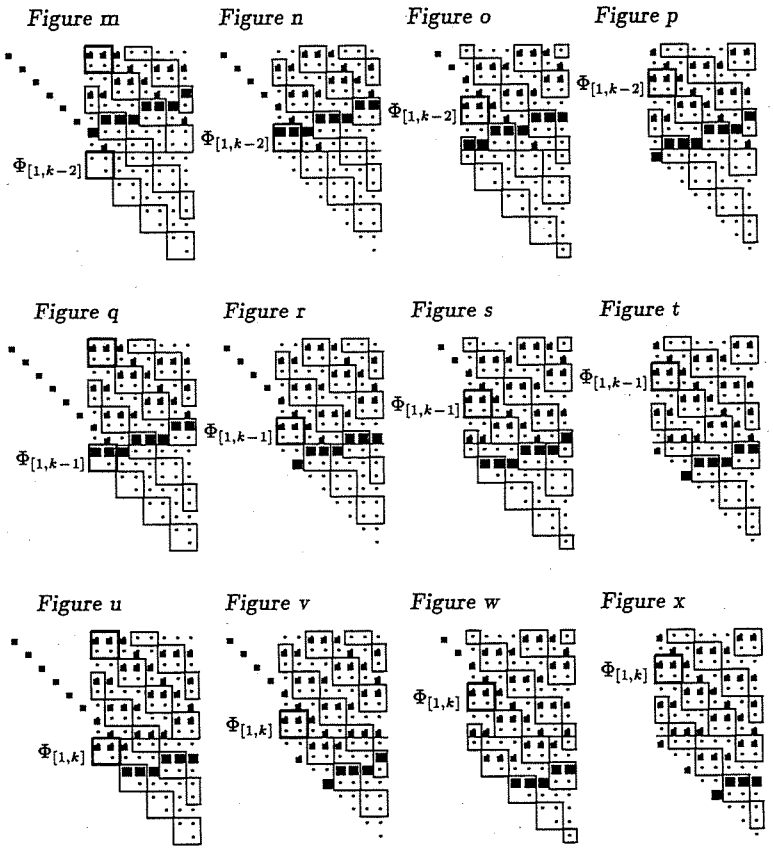


Figure 3.8: *SVD updating array.*

These additional transformations represent a computational overhead, the penalty for pipelining the matrix-vector products with the SVD steps on the same array. Notice however that waiting until $V_{[k-1]}$ is formed completely, before calculating the matrix-vector product, would induce $O\{n\}$ time lags and likewise result in an $O\{n^{-1}\}$ throughput. With the additional computations, the throughput is $O\{n^0\}$!

Let us now focus on the transformations in $\Phi_{[k]}$ and the way these can be processed. One of these transformations is e.g. $\Phi_{[1,k-5]}$ [8] which is computed on the main diagonal in Figure 3.8a –double frame. While being propagated downwards, the $\tilde{a}^T_{[k]}$ vector crosses the upgoing rotation $\Phi_{[1,k-5]}$ in Figure 3.8e. At this point this transformation can straightforwardly be applied to the available $\tilde{a}^T_{[k]}$-components. Similarly, one can verify that $\Phi_{[1,k-4]}$, $\Phi_{[1,k-3]}$, $\Phi_{[1,k-2]}$ and $\Phi_{[1,k-1]}$ are applied in Figure 3.8h, 3.8k, 3.8n and 3.8q respectively. The transformations in the other columns can be traced similarly. In conclusion, each frame in the square array now corresponds to a column transformation, that is applied to a 2×2 block of the V-matrix, and to the two available components of an $\tilde{a}_{[k]}$-vector. These components are propagated one step downwards next. A complete description for a 2×2 block in the V-matrix is then as follows. Notation is slightly modified for conciseness.

[8] see *Chapter 2* for notation

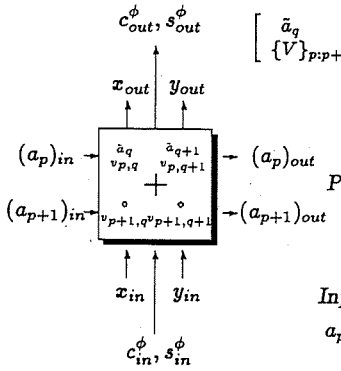
DESCRIPTION 8 :
INTERNAL CELL V-MATRIX

Input Transformation Parameters

$c^\phi, s^\phi \leftarrow c^\phi_{in}, s^\phi_{in}$

Apply Transformation

$$\begin{bmatrix} \tilde{a}_q & \tilde{a}_{q+1} \\ \{V\}_{p:p+1,q:q+1} \end{bmatrix} \leftarrow \begin{bmatrix} \tilde{a}_q & \tilde{a}_{q+1} \\ \{V\}_{p:p+1,q:q+1} \end{bmatrix} \begin{bmatrix} s^\phi & c^\phi \\ c^\phi & -s^\phi \end{bmatrix}$$



Propagate $\tilde{a}_q, \tilde{a}_{q+1}$

$\circ, \diamond \leftarrow \tilde{a}_q, \tilde{a}_{q+1}$

Propagate Transformation Parameters

$c^\phi_{out}, s^\phi_{out} \leftarrow c^\phi, s^\phi$

if q = even

Input a_p, a_{p+1} and Intermediate Results

$a_p, a_{p+1}, x, y \leftarrow (a_p)_{in}, (a_{p+1})_{in}, x_{in}, y_{in}$

Update Intermediate Result

$$\begin{aligned} x &\leftarrow x + v_{p,q} \cdot a_p + v_{p+1,q} \cdot a_{p+1} \\ y &\leftarrow y + v_{p,q+1} \cdot a_p + v_{p+1,q+1} \cdot a_{p+1} \end{aligned}$$

Propagate a_p, a_{p+1} and intermediate results

$(a_p)_{out}, (a_{p+1})_{out}, x_{out}, y_{out} \leftarrow a_p, a_{p+1}, x, y$

end

By the end of Figure 3.8p, the first $\tilde{a}$-vector leaves the square array in a form directly amenable to the (modified Gentleman-Kung) triangular array.

Interlaced QR updating and SVD diagonalization

Finally, the modified Gentleman-Kung array and the triangular SVD array are easily combined, Figure 3.8q etc. In each frame, column and row transformations corresponding to the SVD diagonalization

are performed first –see also Figure 3.2– while in a second step only row transformations are performed corresponding to the modified QR updating [9] *–see also Figure 3.7. Again, column transformations in the first step should be applied to the $\tilde{a}$-components as well. Complete descriptions for boundary cells and internal cells are as follows.*

DESCRIPTION 9 :

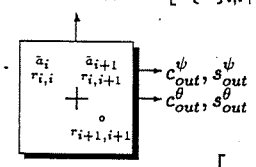
BOUNDARY CELL R-FACTOR

Compute Rotation Angles ϕ, θ

$$c^\phi, s^\phi, c^\theta, s^\theta \leftarrow \cos\phi, \sin\phi, \cos\theta, \sin\theta$$

Apply Transformation

$$\begin{bmatrix} \tilde{a}_i & \tilde{a}_{i+1} \\ \{R\}_{i,i+1} & \end{bmatrix} \leftarrow \begin{bmatrix} 1 & & \\ & s^\theta & c^\theta \\ & c^\theta & -s^\theta \end{bmatrix} \begin{bmatrix} \tilde{a}_i & \tilde{a}_{i+1} \\ \{R\}_{i,i+1} & \end{bmatrix} \begin{bmatrix} s^\phi & c^\phi \\ c^\phi & -s^\phi \end{bmatrix}$$



Compute Rotation Angle ψ

$$c^\psi, s^\psi \leftarrow \cos\psi, \sin\psi$$

Apply Transformation

$$\begin{bmatrix} r_{i,i} & r_{i,i+1} \\ 0 & \tilde{a}_{i+1} \end{bmatrix} \leftarrow \begin{bmatrix} c^\psi & s^\psi \\ -s^\psi & c^\psi \end{bmatrix} \begin{bmatrix} r_{i,i} & 0 \\ \tilde{a}_i & \tilde{a}_{i+1} \end{bmatrix}$$

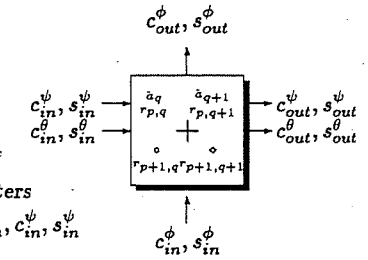
Propagate $\tilde{a}_{i+1}$

$$\circ \leftarrow \tilde{a}_{i+1}$$

Propagate Transformation Parameters

$$c^\phi_{out}, s^\phi_{out}, c^\theta_{out}, s^\theta_{out}, c^\psi_{out}, s^\psi_{out} \leftarrow c^\phi, s^\phi, c^\theta, s^\theta, c^\psi, s^\psi$$

[9] affecting the $\tilde{a}$-components and the upper part of the 2×2 -blocks.



DESCRIPTION 10 :

INTERNAL CELL R-FACTOR

Input Transformation Parameters

$$c^\phi, s^\phi, c^\theta, s^\theta, c^\psi, s^\psi \leftarrow c^\phi_{in}, s^\phi_{in}, c^\theta_{in}, s^\theta_{in}, c^\psi_{in}, s^\psi_{in}$$

Apply Transformation

$$\begin{bmatrix} \tilde{a}_q & \tilde{a}_{q+1} \\ \{R\}_{p:p+1,q:q+1} & \end{bmatrix} \leftarrow \begin{bmatrix} 1 & & \\ & s^\theta & c^\theta \\ & c^\theta & -s^\theta \end{bmatrix} \begin{bmatrix} \tilde{a}_q & \tilde{a}_{q+1} \\ \{R\}_{p:p+1,q:q+1} & \end{bmatrix} \begin{bmatrix} s^\phi & c^\phi \\ c^\phi & -s^\phi \end{bmatrix}$$

Apply Transformation

$$\begin{bmatrix} r_{p,q} & r_{p,q+1} \\ \tilde{a}_q & \tilde{a}_{q+1} \end{bmatrix} \leftarrow \begin{bmatrix} c^\psi & s^\psi \\ -s^\psi & c^\psi \end{bmatrix} \begin{bmatrix} r_{p,q} & r_{p,q+1} \\ \tilde{a}_q & \tilde{a}_{q+1} \end{bmatrix}$$

Propagate $\tilde{a}_q, \tilde{a}_{q+1}$

$$\circ, \diamond \leftarrow \tilde{a}_q, \tilde{a}_{q+1}$$

Propagate Transformation Parameters

$$c^\phi_{out}, s^\phi_{out}, c^\theta_{out}, s^\theta_{out}, c^\psi_{out}, s^\psi_{out} \leftarrow c^\phi, s^\phi, c^\theta, s^\theta, c^\psi, s^\psi$$

Output vector extraction

Without disturbing the array operations, it is possible to generate output vectors at regular time intervals. Let us consider an easy case where a total least squares solution is searched for, which can be obtained by scaling the right singular vector corresponding to the smallest singular value with its last component [Golub & Van Loan 1980].

Suppose that a vector t can be generated on the main diagonal, with all its components equal to zero, except for one component equal to "1". This "1" is generated at the position of the smallest singular

value, e.g. by applying a threshold function to the diagonal elements[10]

$$t_i = 1 \quad iff \quad r_{ii} < \gamma$$

$$t_i = 0 \quad iff \quad r_{ii} > \gamma$$

where γ *satisfies*

$$\sigma_{n-1} > \gamma > \sigma_n$$

and is assumed to be known a priori [11]. *The t-vector is propagated upwards to the square array, where the matrix-vector product* $V \cdot t$ *is performed –similar to the product* $a^T \cdot V$– *which singles out the appropriate right singular vector, see* **Figure 3.9**. *While t is propagated upwards, intermediate results are propagated to the right. The resulting vector becomes available at the right-hand side of the array. If necessary, a scaling of this vector with its last component can be incorporated, as this last component –bottom row in the square array– is encountered first by the nonzero t-component. Once again, it should be stressed that these output vectors can be generated at the same rate at which data vectors are fed in, and that both processes can run simultaneously without interference.*

3.3 Including re-orthogonalizations

In Chapter 2, it has been shown how additional re-orthogonalizations stabilize the overall round-off error propagation in the updating scheme. In the algorithmic description of section 2.2, $T_{[i,k]}$ *is an approximate re-orthogonalization and normalization of rows p and q in the V-matrix. The row indices p and q are chosen as a function of k and i, in a cyclic manner. In view of efficient parallel implementation, we first re-organize this re-orthogonalization scheme. The modified scheme is then mapped onto the systolic array. The computational overhead turns out to be negligible, as the square part of the array (V-matrix) so far remained under-loaded, compared to the triangular part, see below for figures.*

[10] the smallest diagonal element is $\frac{\epsilon^2}{\delta}$ -close to the smallest singular value, cfr. the subspace tracking analysis of *section 2.3*

[11] If such a γ would not be available, some additional control can be added to the diagonal processors in order to locate the smallest diagonal element.

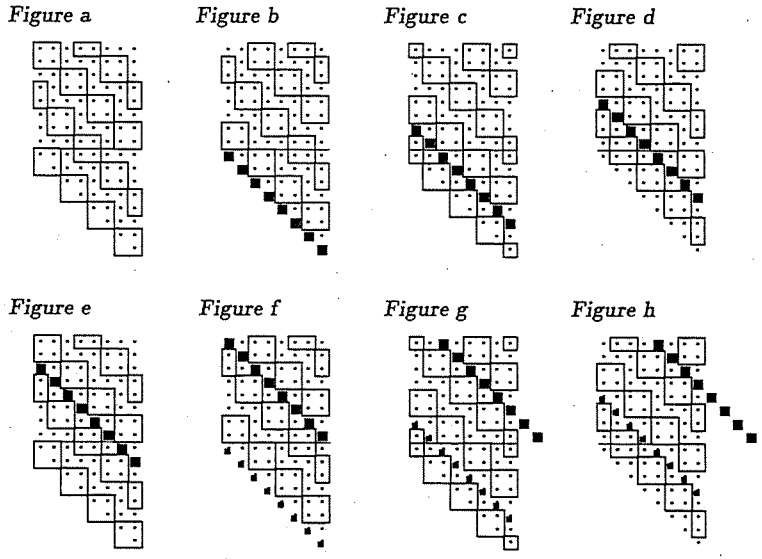


Figure 3.9: *Output vector extraction.*

First of all, as the re-orthogonalization scheme cyclicly adjusts the row vectors in the V-matrix, it is straightforward to introduce additional row permutations in the square part of the array. The 2×2 blocks in the square part then correspond to column transformations (SVD scheme) and row permutations (re-orthogonalization scheme). Orthogonal column transformations clearly do not affect the norms and inner products of the rows, except for local rounding errors assumed smaller than the accumulated errors. Hence, the column transformations are assumed not to interfere with the re-orthogonalization, and thus need not be considered explicitly. As for the row permutations, subsequent positions for the elements in the first column of V are indicated in **Figure 3.10**, *for a (fairly) arbitrary initial row numbering. As an example, the 2×2 block in the upper left corner in Figure 3.10a, interchanges elements 4 and 5, etc.*

Let us now focus on one single row (row 1), and see how it can (approximately) be normalized and orthogonalized with respect to all other rows. Later on we will use this in an overall procedure.

In a first step *the norm (squared) and inner products are computed as a matrix-vector product*

$$x_1 \stackrel{def}{=} V \cdot v_1 = \begin{bmatrix} \boxed{v_1^T} \\ \boxed{v_2^T} \\ \vdots \\ \boxed{v_n^T} \end{bmatrix} \cdot \left[\boxed{v_1} \right] \stackrel{def}{=} \begin{bmatrix} \xi_{11} \\ \xi_{12} \\ \vdots \\ \xi_{1n} \end{bmatrix}$$

where v_i^T is the i^{th} row in V. On the systolic array, where v_1 initially resides in the bottom row, it suffices to propagate the v_1-components upwards, and accumulate the inner products from the left to the right, see Figure 3.10a-h, where pq is shorthand for ξ_{pq}. The resulting x_1-vector components run out at the right-hand side.

In a second step *this x_1-vector is "back-propagated" to the left, Figure 3.10e-t. Due to the permutations along the way, the x_1-components reach the left-hand side of the array at the right time and in the right place, such that*

Figure a *Figure b* *Figure c* *Figure d*

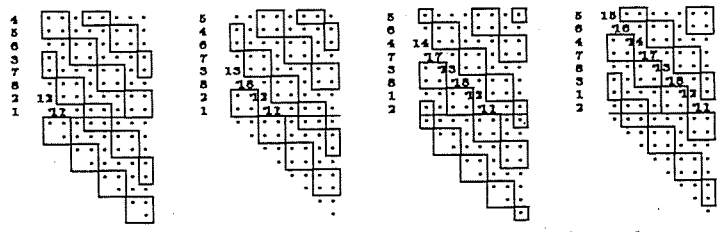


Figure e *Figure f* *Figure g* *Figure h*

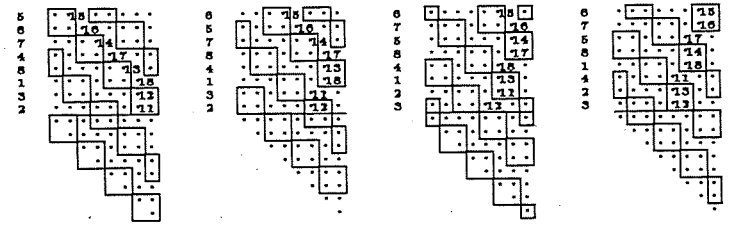


Figure i *Figure j* *Figure k* *Figure l*

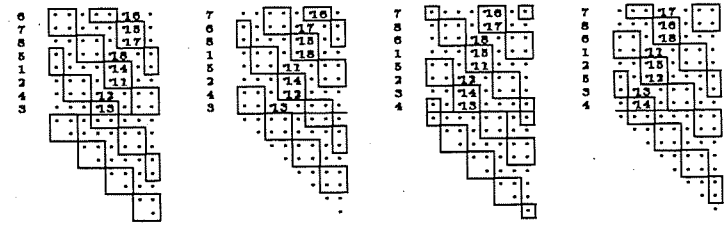
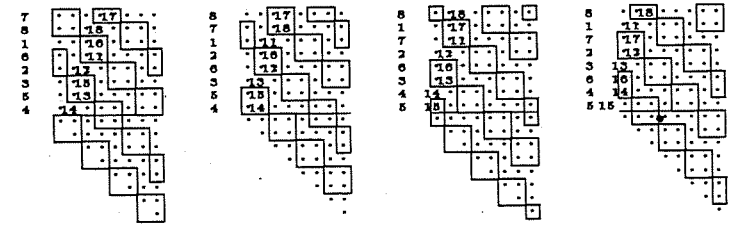


Figure m *Figure n* *Figure o* *Figure p*



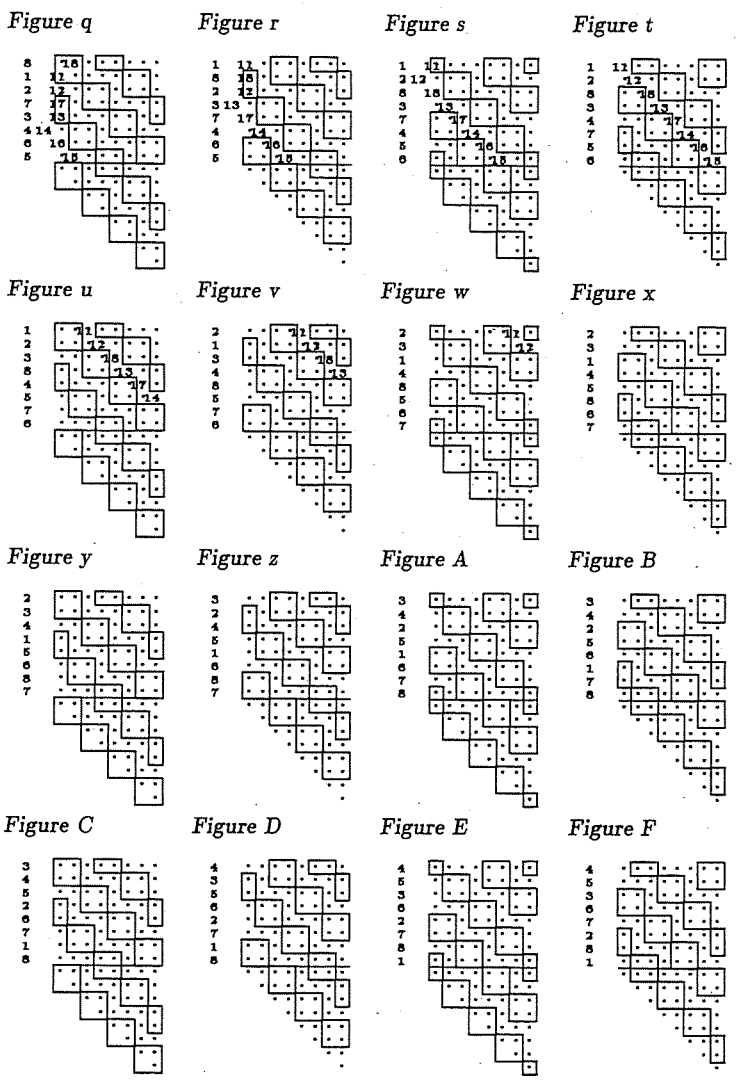


Figure 3.10: *Orthogonalization procedure.*

in a third step *a correction vector can be computed as a matrix-vector product*

$$y_1 \stackrel{def}{=} \left[\begin{array}{cccc} \frac{1}{2}(\xi_{11}-1) & \xi_{12} & \dots & \xi_{1n} \end{array} \right] \cdot \left[\begin{array}{c} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{array} \right]$$

where the term $\frac{1}{2}(\xi_{11}-1)$ corresponds to the first order term in the Taylor series expansion for the normalization of v_1. The y_1-vector components are accumulated from the bottom to the top, while x_1 is again being propagated to the right, Figure 3.10q-w. Finally,

in a fourth step *v_1 –which meanwhile moved on to the top row of the array– is adjusted with y_1*

$$\tilde{v}_1 \leftarrow v_1 - y_1.$$

These operations are performed in the top row of the array, Figure 3.10t-w.

One can check that if

$$\begin{aligned} v_1 \cdot v_1 &= 1 + O\{\epsilon\} \\ v_1 \cdot v_p &= O\{\epsilon\} \qquad p = 2, \dots, n \end{aligned}$$

for some small $\epsilon \ll 1$, then

$$\begin{aligned} \tilde{v}_1 \cdot \tilde{v}_1 &= 1 + O\{\epsilon^2\} \\ \tilde{v}_1 \cdot v_p &= O\{\epsilon^2\} \qquad p = 2, \dots, n. \end{aligned}$$

The above procedure for v_1 should now be repeated for rows 2, 3, etc., and furthermore everything should be pipelined. Obviously, one could start a similar procedure for v_2 in Figure 3.10e, for v_3 in Figure 3.10i, etc. The pipelining of such a scheme would be remarkbly simple, but unfortunately there is something wrong with it. A slight modification is needed to make things work properly.

As for the processing of v_2, one can check that the computed inner product ξ_{21} equals $v_2 \cdot v_1 = v_1 \cdot v_2 = \xi_{12}$, while it should equal $v_2 \cdot \tilde{v}_1$. A similar problem occurs with ξ_{31} and ξ_{32}, etc. In general, problems

occur when computing inner products with ascending rows, which still have to be adjusted in the top row of the array, before the relevant inner product can be computed. This problem is readily solved as follows. Instead of computing inner products with all other rows, we only take descending rows into account. This is easily done by assigning tags to the rows, where e.g. a 0-tag indicates an ascending row, a 1-tag indicates a descending row. Tags are reset at the top and the bottom of the array. Computing an x_i vector is then done as follows

$$x_i \stackrel{def}{=} V \cdot v_i = \begin{bmatrix} \boxed{TAG_1 v_1^T} \\ \boxed{TAG_2 v_2^T} \\ \vdots \\ \boxed{TAG_n v_n^T} \end{bmatrix} \cdot \left[\boxed{v_i} \right] \stackrel{def}{=} \begin{bmatrix} \xi_{i1} \\ \xi_{i2} \\ \vdots \\ \xi_{in} \end{bmatrix}.$$

For the 8×8 example of Figure 3.10, one can check that the result of this is that v_1 is orthogonalized onto v_2, v_3, v_4 and v_5 ($\xi_{16} = \xi_{17} = \xi_{18} = 0$). Similarly v_2 is orthogonalized onto v_3, v_4, v_5 and v_6 ($\xi_{27} = \xi_{28} = \xi_{21} = 0$), etc. The resulting ordering is recast as follows [12]

11	12	13	14	15						
	22	23	24	25	26					
		33	34	35	36	37				
			44	45	46	47	48			
				55	56	57	58	51		
					66	67	68	61	62	
						77	78	71	72	73
							88	81	82	83	84.

In such a sweep, each combination appears at least once [13]. In the general case, the rows are cyclicly normalized and orthogonalized onto the $\frac{n}{2}$ succeeding rows. One can prove that if $\|VV^T - I\|_F = O\{\epsilon\}$ before a particular sweep, then $\|VV^T - I\|_F = O\{\epsilon^2\}$ after the sweep. In other words, the quadratic convergence rate is maintained. On the other hand, it is seen that one single sweep takes twice the computation time for a sweep in a "normal" cyclic by rows or odd-even ordering.

[12] pp refers to the normalization of row p, whereas pq for $p \neq q$ refers to the orthogonalization of row p onto row q.

[13] Exactly $4 = \frac{n}{2}$ combination appear twice, e.g. 15 (51), etc.

This is hardly an objection, as the re-orthogonalization is only meant to keep V reasonably close to orthogonal. The error analysis of section 2.4 thus still applies –with slightly adjusted constant coefficients.

Complete processor descriptions are left out for the sake of brevity. Let it suffice to give an operation count for different kinds of processors. see **Table 3.1**, from which it follows that the re-orthogonalization does not increase the load of the critical cells.

	SVD updating	Re-orthog.	Total
Internal cell	16×	8×	24×
V-matrix	10+	8+	18+
Internal cell	28×		28×
R-matrix	14+		14+
Diagonal cell	25×		25×
R-matrix	13+		13+
	9÷		9÷
	4√·		4√·

Table 3.1: Operation count per cell

Finally, as the rows of V continuously interchange, each input vector a should be permuted accordingly, before multiplication [14]. One can straightforwardly design a kind of "preprocessor" for a, that outputs the right components of a at the right time, see **Figure 3.11**. For the sake of brevity, we will not go into the details of various possible realizations for this fairly elementary building block.

3.4 Square root free algorithms

The throughput in the parallel SVD updating array is essentially determined by the computation times for the processors on the main diagonal to calculate the rotation angles both for the QR updatings and the SVD steps. In general these computations require respectively one and three square roots, and this appears to be the main computational bottle-neck. Gentleman developed a square root free (SQF) procedure for QR updating [Gentleman 1973]. where

[14] $a^t \cdot V = (a^t \cdot \Pi^t) \cdot (\Pi \cdot V)$, where Π is a permutation matrix.

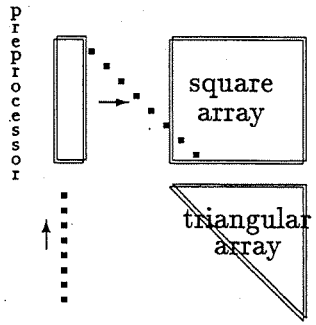


Figure 3.11: *Overview SVD updating array.*

use is made of a one-sided factorization of the R-matrix. The SVD schemes however as such do not lend themselves to SQF implementation. Still, in [Charlier et al. 1988] a few alternative SVD schemes have been investigated, based on approximate formulas for the computation of either $\tan\theta$ *or* $\tan\phi$*. When combined with a generalized Gentleman procedure with a two-sided factorization of the R-factor, these schemes eventually yield SQF SVD updating algorithms. Implementation on a systolic array hardly imposes any changes, when compared to the conventional algorithm. Furthermore, as the approximate formulas for the rotation angles are in fact at least first order approximations, the (first order) performance analysis in Chapter 2 still applies. In other words, the same upper bounds for the tracking error are valid, even when approximate formulas are used.*

Gentleman's procedure for QR updating

A QR updating amounts to applying a sequence of orthogonal row transformations –Givens rotations– see Chapter 1. Each Givens transformation combines a row vector r_i^T of R with the current version of the input vector x^T (partially zeroed)

$$r_i^T = \begin{bmatrix} 0 & \dots & 0 & r_{i,i} & r_{i,i+1} & \dots & r_{i,n} \end{bmatrix}$$

$$x^T = \begin{bmatrix} 0 & \dots & 0 & x_i & x_{i+1} & \dots & x_n \end{bmatrix}$$

and transforms them to two new vectors

$$r_i^{*T} = \begin{bmatrix} 0 & \dots & 0 & r_{i,i}^* & r_{i,i+1}^* & \dots & r_{i,n}^* \end{bmatrix}$$
$$x^{*T} = \begin{bmatrix} 0 & \dots & 0 & 0 & x_{i+1}^* & \dots & x_n^* \end{bmatrix}$$

where

$$\begin{bmatrix} r_i^{*T} \\ x^{*T} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} r_i^T \\ x^T \end{bmatrix}.$$

The rotation angle θ is chosen to eliminate the leading element x_i, such that $x_i^* = 0$

$$\tan\theta = \frac{x_i}{r_{i,i}}$$
$$\cos\theta = \frac{1}{\sqrt{1+\tan^2\theta}}$$
$$\sin\theta = \cos\theta \cdot \tan\theta.$$

Now the aim is to avoid the square root in the computation of $\cos\theta$*. In [Gentleman 1973] it is shown how to perform plane rotations without square roots and with an additional saving in the number of multiplications –three multiplication formulas. Hammarling extended this idea to the general modified Givens rotations [Hammarling 1974], also further reducing the number of multiplications –two multiplication formulas– although clearly with some danger of underflow when a sequence of rotations is involved. Scaled Givens rotations [Barlow & Ipsen 1987] in essence represent a further refinement to the basic idea, in order to prevent problems of underflow and pivoting. Amongst many different implementations, at this stage we prefer to choose the simplest one for the sake of an easy exposition. All possible modifications to the main idea thereby largely carry over to our case in a straightforward manner.*

For the QR updating problem, one computes a diagonal matrix D and a upper triangular matrix $\bar{R}$ [15]*, such that*

$$R = D^{\frac{1}{2}} \cdot \bar{R}.$$

[15] in the sequel, an overbar always refers to a factorization.

The i^{th} row in R is thus scaled with a certain factor, that is stored in squared form as a diagonal element d_i in the D-factor. Consider now a Givens transformation applied to two row vectors

$$\begin{aligned} r_i^T &= \left[\begin{array}{ccccccc} 0 & \dots & 0 & d_i^{\frac{1}{2}}\bar{r}_{i,i} & d_i^{\frac{1}{2}}\bar{r}_{i,i+1} & \dots & d_i^{\frac{1}{2}}\bar{r}_{i,n} \end{array}\right] \\ x^T &= \left[\begin{array}{ccccccc} 0 & \dots & 0 & \delta^{\frac{1}{2}}\bar{x}_i & \delta^{\frac{1}{2}}\bar{x}_{i+1} & \dots & \delta^{\frac{1}{2}}\bar{x}_n \end{array}\right] \end{aligned}$$

which transforms them to two new vectors

$$\begin{aligned} r_i^{*T} &= \left[\begin{array}{ccccccc} 0 & \dots & 0 & d_i^{*\frac{1}{2}}\bar{r}_{i,i}^* & d_i^{*\frac{1}{2}}\bar{r}_{i,i+1}^* & \dots & d_i^{*\frac{1}{2}}\bar{r}_{i,n}^* \end{array}\right] \\ x^{*T} &= \left[\begin{array}{ccccccc} 0 & \dots & 0 & 0 & \delta^{*\frac{1}{2}}\bar{x}_{i+1}^* & \dots & \delta^{*\frac{1}{2}}\bar{x}_n^* \end{array}\right]. \end{aligned}$$

A two multiplication formula is derived as follows

$$\begin{aligned} &\left[\begin{array}{cc} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{array}\right] \cdot \left[\begin{array}{cc} d_i^{\frac{1}{2}} & 0 \\ 0 & \delta^{\frac{1}{2}} \end{array}\right] \cdot \left[\begin{array}{c} \bar{r}_{i,k} \\ \bar{x}_k \end{array}\right] \\ &= \underbrace{\left[\begin{array}{cc} d_i^{\frac{1}{2}}\cos\theta & 0 \\ 0 & \delta^{\frac{1}{2}}\cos\theta \end{array}\right]}_{diag\{d_i^{*\frac{1}{2}},\delta^{*\frac{1}{2}}\}} \cdot \underbrace{\left[\begin{array}{cc} 1 & (\frac{\delta}{d_i})^{\frac{1}{2}}\tan\theta \\ -(\frac{d_i}{\delta})^{\frac{1}{2}}\tan\theta & 1 \end{array}\right] \cdot \left[\begin{array}{c} \bar{r}_{i,k} \\ \bar{x}_k \end{array}\right]}_{\left[\begin{array}{c} \bar{r}_{i,k}^* \\ \bar{x}_k^* \end{array}\right]}. \end{aligned}$$

Again the rotation angle θ is chosen to eliminate the leading element x_i

$$\tan\theta = \frac{\delta^{\frac{1}{2}}\bar{x}_i}{d_i^{\frac{1}{2}}\bar{r}_{i,i}}.$$

The row transformation formula then reads

$$\left[\begin{array}{c} \bar{r}_{i,k}^* \\ \bar{x}_k^* \end{array}\right] = \underbrace{\left[\begin{array}{cc} 1 & (\frac{\delta}{d_i})^{\frac{1}{2}}\tan\theta \\ -(\frac{d_i}{\delta})^{\frac{1}{2}}\tan\theta & 1 \end{array}\right]}_{\left[\begin{array}{cc} 1 & \frac{\bar{x}_i\delta}{\bar{r}_{i,i}d_i} \\ -\frac{\bar{x}_i}{\bar{r}_{i,i}} & 1 \end{array}\right]} \cdot \left[\begin{array}{c} \bar{r}_{i,k} \\ \bar{x}_k \end{array}\right]$$

with scale factor updating

$$\begin{aligned} d_i^* &= d_i\cos^2\theta \\ \delta^* &= \delta\cos^2\theta \end{aligned}$$

$$\cos^2\theta = \frac{1}{1+\frac{\bar{x}_i^2\delta}{\bar{r}_{i,i}^2 d_i}}.$$

The above formulas describe how the transformed row vectors can be expressed as a row of an updated $D^{*\frac{1}{2}}\cdot\bar{R}^*$ and a new scaled row x^*. The updating formulas not only avoid the square root as mentioned, but take only two multiplications instead of four for the computation of $\bar{r}_{i,k}^*$ and $\bar{x}_k^*$ $(k=i+1,\dots,n)$. In a systolic array, instead of propagating $\cos\theta$ and $\sin\theta$, the rotation parameters $\frac{\bar{x}_i\delta}{\bar{r}_{i,i}d_i}$ and $\frac{\bar{x}_i}{\bar{r}_{i,i}}$ have to be propagated. Another advantage of modified Givens rotations over standard rotations occurs when downdating is to be performed. It then suffices to perform an updating with the scaling factor of the input row set equal to $\sqrt{-1}$ $(\delta=-1)$, orthogonal rotations thereby being transformed into hyperbolic rotations [Barlow & Ipsen 1987].

Square root free SVD computations

The SVD procedure is seen to reduce to solving elementary 2×2 SVD's on the main diagonal, Chapter 1. For an approximate SVD computation, the relevant transformation formula becomes

$$\underbrace{\left[\begin{array}{cc} r_{i,i}^* & r_{i,i+1}^* \\ 0 & r_{i+1,i+1}^* \end{array}\right]}_{\{R^*\}_{i,i+1}} \leftarrow \underbrace{\left[\begin{array}{cc} -\sin\theta & \cos\theta \\ \cos\theta & \sin\theta \end{array}\right]}_{\{\Pi\{i,i+1\}G\{i,i+1,\theta\}\}_{i,i+1}} \cdot \underbrace{\left[\begin{array}{cc} r_{ii} & r_{i,i+1} \\ 0 & r_{i+1,i+1} \end{array}\right]}_{\{R\}_{i,i+1}} \cdot \underbrace{\left[\begin{array}{cc} -\sin\phi & \cos\phi \\ \cos\phi & \sin\phi \end{array}\right]}_{\{G\{i,i+1,\phi\}\Pi\{i,i+1\}\}_{i,i+1}}$$

where $r_{i,i+1}$ is only approximately being annihilated ($|r_{i,i+1}^*| \le |r_{i,i+1}|$). In particular, the following approximate schemes from [Charlier et al. 1988] turn out to be very useful for our purpose. For details, we refer to [Charlier et al. 1988].

if $|r_{i,i}| \ge |r_{i+1,i+1}|$

$$\sigma = \frac{r_{i+1,i+1}r_{i,i+1}}{r_{i,i}^2 - r_{i+1,i+1}^2 + r_{i,i+1}^2}$$

approximation 1 : $\tan\theta = \sigma$

approximation 2 : $\tan\theta = \frac{\sigma}{1+\sigma^2}$

$$\tan\phi = \frac{r_{i+1,i+1}\tan\theta + r_{i,i+1}}{r_{i,i}}$$

if $|r_{i,i}| \leq |r_{i+1,i+1}|$

$$\sigma = \frac{r_{i,i} r_{i,i+1}}{r_{i+1,i+1}^2 - r_{i,i}^2 + r_{i,i+1}^2}$$

approximation 1 : $\tan\phi = \sigma$

approximation 2 : $\tan\phi = \frac{\sigma}{1+\sigma^2}$

$$\tan\theta = \frac{r_{i,i}\tan\phi - r_{i,i+1}}{r_{i+1,i+1}}.$$

In the sequel, we consider only $|r_{i,i}| \geq |r_{i+1,i+1}|$, *as the derived formulas can straightforwardly be adapted for the other case. These approximate schemes still require two square roots for the computation of* $\cos\phi$ *and* $\cos\theta$.

The above approximate formulas can however be combined with a **generalized Gentleman procedure**, *where use is made of a two-sided factorization of the R-matrix*

$$R = D_{row}^{\frac{1}{2}} \cdot \bar{R} \cdot D_{col}^{\frac{1}{2}}$$

and where only $\bar{R}$, D_{row} *and* D_{col} *are stored*

Let us first rewrite the first approximate formula for $\tan\theta$

$$\tan\theta = \underbrace{\left(\frac{r_{i+1,i+1}^2}{r_{i,i}^2 - r_{i+1,i+1}^2 + r_{i,i+1}^2}\right)}_{\rho} \cdot \frac{r_{i,i+1}}{r_{i+1,i+1}}.$$

As ρ *contains only squared values, it can be computed from the factorization of R as well.*

$$\rho = \frac{d_{i+1}^{row} d_{i+1}^{col} \bar{r}_{i+1,i+1}^2}{d_i^{row} d_i^{col} \bar{r}_{i,i}^2 - d_{i+1}^{row} d_{i+1}^{col} \bar{r}_{i+1,i+1}^2 + d_i^{row} d_{i+1}^{col} \bar{r}_{i,i+1}^2}.$$

Obviously, a similar formula for ρ *can be derived from the second approximate formula for* $\tan\theta$ [16]. *Applying Gentleman's procedure to the row transformation then gives*

$$\begin{bmatrix} -\sin\theta & \cos\theta \\ \cos\theta & \sin\theta \end{bmatrix} \cdot \begin{bmatrix} d_i^{row\frac{1}{2}} & 0 \\ 0 & d_{i+1}^{row\frac{1}{2}} \end{bmatrix} \cdot \begin{bmatrix} \bar{r}_{i,i} & \bar{r}_{i,i+1} \\ 0 & \bar{r}_{i+1,i+1} \end{bmatrix} \cdot \begin{bmatrix} d_i^{col\frac{1}{2}} & 0 \\ 0 & d_{i+1}^{col\frac{1}{2}} \end{bmatrix}$$

[16] that has better convergence properties, see [Charlier *et al.* 1988].

$$= \underbrace{\begin{bmatrix} d_{i+1}^{row\frac{1}{2}}\cos\theta & 0 \\ 0 & d_i^{row\frac{1}{2}}\cos\theta \end{bmatrix}}_{diag\{d_i^{row^*\frac{1}{2}}, d_{i+1}^{row^*\frac{1}{2}}\}} \cdots$$

$$\cdots \underbrace{\begin{bmatrix} -(\frac{d_i^{row}}{d_{i+1}^{row}})^{\frac{1}{2}}\tan\theta & 1 \\ 1 & (\frac{d_{i+1}^{row}}{d_i^{row}})^{\frac{1}{2}}\tan\theta \end{bmatrix} \cdot \begin{bmatrix} \bar{r}_{i,i} & \bar{r}_{i,i+1} \\ 0 & \bar{r}_{i+1,i+1} \end{bmatrix}}_{\begin{bmatrix} \bar{r}_{i,i}^\circ & \bar{r}_{i,i+1}^\circ \\ \bar{r}_{i+1,i}^\circ & \bar{r}_{i+1,i+1}^\circ \end{bmatrix}} \cdot \begin{bmatrix} d_i^{col\frac{1}{2}} & 0 \\ 0 & d_{i+1}^{col\frac{1}{2}} \end{bmatrix}.$$

With

$$\tan\theta = \rho \cdot \frac{\bar{r}_{i,i+1} d_i^{row\frac{1}{2}}}{\bar{r}_{i+1,i+1} d_{i+1}^{row\frac{1}{2}}}$$

this leads to **row transformation formulas**

$$\begin{bmatrix} \bar{r}_{i,i}^\circ & \bar{r}_{i,i+1}^\circ \\ \bar{r}_{i+1,i}^\circ & \bar{r}_{i+1,i+1}^\circ \end{bmatrix} = \begin{bmatrix} -\rho \cdot \frac{\bar{r}_{i,i+1} d_i^{row}}{\bar{r}_{i+1,i+1} d_{i+1}^{row}} & 1 \\ 1 & \rho \cdot \frac{\bar{r}_{i,i+1}}{\bar{r}_{i+1,i+1}} \end{bmatrix} \cdot \begin{bmatrix} \bar{r}_{i,i} & \bar{r}_{i,i+1} \\ 0 & \bar{r}_{i+1,i+1} \end{bmatrix}$$

with **scale factor updating** *–note the implicit row permutation–*

$$d_i^{row^*} = d_{i+1}^{row}\cos^2\theta$$
$$d_{i+1}^{row^*} = d_i^{row}\cos^2\theta$$

$$\cos^2\theta = \frac{1}{1 + \rho^2 \cdot \frac{\bar{r}_{i,i+1}^2 d_i^{row}}{\bar{r}_{i+1,i+1}^2 d_{i+1}^{row}}}.$$

The column transformation should then annihilate $r_{i+1,i}^\circ$ *in order to preserve the triangular structure. With*

$$\tan\phi = \frac{\bar{r}_{i+1,i+1}^\circ d_{i+1}^{col\frac{1}{2}}}{\bar{r}_{i+1,i}^\circ d_i^{col\frac{1}{2}}}$$

we can again apply Gentleman's procedure as follows

$$\begin{bmatrix} {d_i^{row}}^{*\frac{1}{2}} & 0 \\ 0 & {d_{i+1}^{row}}^{*\frac{1}{2}} \end{bmatrix} \cdot \begin{bmatrix} \bar{r}^{\circ}_{i,i} & \bar{r}^{\circ}_{i,i+1} \\ \bar{r}^{\circ}_{i+1,i} & \bar{r}^{\circ}_{i+1,i+1} \end{bmatrix} \cdot \begin{bmatrix} {d_i^{col}}^{\frac{1}{2}} & 0 \\ 0 & {d_{i+1}^{col}}^{\frac{1}{2}} \end{bmatrix} \cdot \begin{bmatrix} -\sin\phi & \cos\phi \\ \cos\phi & \sin\phi \end{bmatrix}$$

$$= \begin{bmatrix} {d_i^{row}}^{*\frac{1}{2}} & 0 \\ 0 & {d_{i+1}^{row}}^{*\frac{1}{2}} \end{bmatrix} \cdot \underbrace{\begin{bmatrix} \bar{r}^{\circ}_{i,i} & \bar{r}^{\circ}_{i,i+1} \\ \bar{r}^{\circ}_{i+1,i} & \bar{r}^{\circ}_{i+1,i+1} \end{bmatrix} \cdot \begin{bmatrix} -(\frac{d_i^{col}}{d_{i+1}^{col}})^{\frac{1}{2}} \tan\phi & 1 \\ 1 & (\frac{d_{i+1}^{col}}{d_i^{col}})^{\frac{1}{2}} \tan\phi \end{bmatrix}}_{\begin{bmatrix} \bar{r}^{*}_{i,i} & \bar{r}^{*}_{i,i+1} \\ 0 & \bar{r}^{*}_{i+1,i+1} \end{bmatrix}} \cdots$$

$$\cdots \underbrace{\begin{bmatrix} {d_{i+1}^{col}}^{\frac{1}{2}} \cos\phi & 0 \\ 0 & {d_i^{col}}^{\frac{1}{2}} \cos\phi \end{bmatrix}}_{diag\{{d_i^{col}}^{*\frac{1}{2}}, {d_{i+1}^{col}}^{*\frac{1}{2}}\}}$$

which then leads to **column transformation formulas**

$$\begin{bmatrix} \bar{r}^{*}_{i,i} & \bar{r}^{*}_{i,i+1} \\ 0 & \bar{r}^{*}_{i+1,i+1} \end{bmatrix} = \begin{bmatrix} \bar{r}^{\circ}_{i,i} & \bar{r}^{\circ}_{i,i+1} \\ \bar{r}^{\circ}_{i+1,i} & \bar{r}^{\circ}_{i+1,i+1} \end{bmatrix} \cdot \begin{bmatrix} -\frac{\bar{r}^{\circ}_{i+1,i+1}}{\bar{r}^{\circ}_{i+1,i}} & 1 \\ 1 & \frac{\bar{r}^{\circ}_{i+1,i+1} d_{i+1}^{col}}{\bar{r}^{\circ}_{i+1,i} d_i^{col}} \end{bmatrix}$$

with **scale factor updating**

$$\begin{aligned} d_i^{col*} &= d_{i+1}^{col} \cos^2\phi \\ d_{i+1}^{col*} &= d_i^{col} \cos^2\phi \end{aligned}$$

$$\cos^2\phi = \frac{1}{1 + \frac{\bar{r}^{\circ 2}_{i+1,i+1} d_{i+1}^{col}}{\bar{r}^{\circ 2}_{i+1,i} d_i^{col}}}.$$

Square root free SVD updating

The above approximate SVD schemes straightforwardly combine with the square root free QR updating procedure into a square root free SVD updating procedure. At a certain time step, the data matrix is reduced to R, triangular and stored in factorized form

$$R = D_{row}^{\frac{1}{2}} \cdot \bar{R} \cdot D_{col}^{\frac{1}{2}}.$$

Furthermore, the same column scaling is applied to the V-matrix

$$V = \bar{V} \cdot D_{col}^{\frac{1}{2}}$$

where $\bar{V}$ is stored instead of V. The reason for this is twofold. First, the column rotations to be applied to the V-matrix are computed as modified Givens rotations. Explicitly applying these transformations to an unfactorized V would then necessarily require square roots. Second, a new row vector a^T to be updated, immediately gets the correct column scaling from the matrix-vector product $a^T \cdot \bar{V}$, so that the QR updating can then be carried out as if there were no column scaling at all. The updating can indeed be described as follows.

$$\begin{aligned} \begin{bmatrix} A \\ a^T \end{bmatrix} &= \begin{bmatrix} U & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} D_{row}^{\frac{1}{2}} \cdot \bar{R} \cdot D_{col}^{\frac{1}{2}} \\ a^T \cdot (\bar{V} \cdot D_{col}^{\frac{1}{2}}) \end{bmatrix} \cdot V^T \\ &= \begin{bmatrix} U & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} D_{row}^{\frac{1}{2}} & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \bar{R} \\ a^T \cdot (\bar{V}) \end{bmatrix} \cdot D_{col}^{\frac{1}{2}} \cdot V^T \\ &= \textit{etc.} \end{aligned}$$

The factor in the midst of this expression can then be reduced to a triangular factor by QR updating, making use of modified Givens rotations. The further reduction of the resulting triangular factor can be carried out next as detailed in the previous section.

From the above explanation, it follows that on a systolic array, a square root free updating algorithm imposes hardly any changes. The diagonal matrices D_{row} and D_{col} are obviously stored in the processor elements on the main diagonal, and $\bar{R}$ and $\bar{V}$ are stored instead of R and V. The matrix-vector product $a^T \cdot \bar{V}$ is computed in the square part, and the $\bar{R}$-factor is updated with

$$a^T \cdot \bar{V}$$

next, much like the R-factor was updated with $a^T \cdot V$ in the original algorithm. All other operations are carried out much the same way, albeit that modified rotations are used throughout. When re-orthogonalizations are included, it is necessary to propagate the scale factors to the square array –along with the column transformation– such that the norms and inner products can be computed consistently. The rest is straightforward.

An operation count for a square root free implementation is exhibited in **Table 3.2.** *The operation count for V-processors remains unchanged (as compared to Table 3.1), while the computational load for the diagonal processors is reduced to roughly the same level, apart from the divisions. Note that the operation count for the diagonal processors heavily depends on the specific implementation. We refer to the literature for various –more efficient– implementations [Barlow & Ipsen 1987, Hammarling 1974]. The internal R-processors are seen to be under-loaded this time, due to the reduction in the number of multiplications.*

	SVD updating	*Re-orthog.*	*Total*
Internal cell	*10×*	*12×*	*22×*
V-matrix	*10+*	*8+*	*18+*
Internal cell	*14×*		*14×*
R-matrix	*14+*		*14+*
Diagonal cell	*35×*		*35×*
R-matrix	*17+*		*17+*
	9:		*9:*

Table 3.2: Operation count per cell

Finally, to get an idea of the computational burden in practical situations, we make a crude operation count as follows. Both the matrix (problem) size n and the sampling frequency, i.e. the number of updates per second f_s, are important here:

- *In an hexogonally connected array, each node corresponds to four of the 2 × 2 cells for which an operation count is given in Table 3.2. On the average, each node thus performs something like 160 operations (+ or ×) per update. The total number of operations per node per second thus equals*

$$\sharp ops.\ per\ node\ per\ second \simeq 160 \cdot f_s.$$

If each such node is e.g. fitted onto one (off-the-shelf) DSP, which can perform say 10k floating point operations per second (like a TMS 32010), the attainable sampling rate would still only be 60Hz. If fixed point operations can be used[17]*, the attainable*

[17] this is a suggestion for further research

sampling rate could be more than 100 times higher, up to 10kHz say. With dedicated (floating point) processors, an increase of performance of 10^3 seems to be within sight –see e.g. [McCanny & McWhirter 1987]–, up to 100kHz say.

- *For a fixed size processor array, the total number of operations per update is important. As the number of elementary nodes roughly equals $\frac{1}{8} \cdot \frac{3}{2} n^2$ (eight matrix elements per node, roughly $\frac{3}{2} n^2$ matrix elements in R and V), the total operation count is as follows*

$$\begin{aligned} \sharp ops.\ per\ second &\simeq 160 \cdot f_s \cdot \frac{3}{16} n^2 \\ &\simeq 30 \cdot f_s \cdot n^2. \end{aligned}$$

A few characteristic figures are given in **Table 3.3.** *For a small problem, where $n = 10$ and $f_s = 1Hz$, for instance one TMS32010 will do, even when floating point operations are used. For $n = 10$ and $f_s = 1kHz$, the problem can be fitted onto a few -three say– TMS32010 processors, with fixed point arithmetic. In our –apparently simple– example with the* **flexible one-link robot,** *where $f_s = 200Hz$ and we chose $n = 40$, the operation count is 10^7, so that roughly 10(!) TMS processors are needed, fixed point arithmetic. The large value for n –inherent in the identification procedure– is particularly hurtful here.*

	$n = 10$	$n = 100$
$f_s = 1Hz$	$3 \cdot 10^3$	$3 \cdot 10^5$
$f_s = 1kHz$	$3 \cdot 10^6$	$3 \cdot 10^8$

Table 3.3: Overall operation count per second

3.5 Conclusion

The approximate SVD updating procedure of the previous chapter has been mapped onto a systolic array with $O\{n^2\}$ parallelism for $O\{n^2\}$ complexity. Different operations (matrix-vector multiplications, QR updates and SVD steps) thereby have to be pipelined on the same array. It has been shown how the throughput could be increased from $O\{n^{-1}\}$ up to $O\{n^0\}$ at the expense of a small computational overhead. This approach is apparently new. In Chapters 4 and 5, similar systolic arrays will be developed for various other combined algorithms.

As for the implementational details, it has finally been shown how all square roots can be avoided, by combining modified Givens rotations with approximate schemes for the computation of rotation angles in the SVD steps. In this way, a main computational bottleneck for the array implementation can be overcome.

Chapter 4

Updating generalized SVD's

In this chapter, we extend the SVD updating procedure to generalized SVD's (GSVD's), viz. the **quotient SVD** *(QSVD) [Van Loan 1976, Paige & Saunders 1981] and the* **product SVD** *(PSVD) [Fernando & Hammarling 1987, De Moor 1989]. The emphasis is on algorithms and their parallel implementation. Performance and error analyses are left out for the sake of brevity.*

In order to demonstrate the usefulness of such techniques, we first dwell on one particular application of QSVD updating, systems identification once again. The **state space identification** *procedure of Chapter 1 is reformulated with the QSVD, where now an input-output error covariance matrix can be taken into account explicitly. The applicability of the identification scheme is thereby extended to the case where the input and output data are corrupted by additive coloured noise. It turns out that in practice, due to the use of various pre-filtering techniques (anti-aliasing etc.), this latter case is mostly encountered. The extended identification scheme then explicitly compensates for the filter characteristics, as will be demonstrated. An adaptive version for model updating is devised as well, which boils down to QSVD updating*[1].

We then focus on algorithms for such generalized SVD updating problems where new rows are appended to either one or both of the matrices involved. As far as we know, these problems had not been looked at so far. For the QSVD updating problem, use is made of a Jacobi-type QSVD algorithm

[1]The present development is heavily inspired by recent progress in total least squares techniques [Van Huffel 1989] for identifying *static linear relations*. The generalized identification scheme can therefore be viewed as the analogous counterpart for identifying *dynamic linear relations*.

due to Paige [Paige 1986]. For the PSVD updating on the other hand, we make use of a similar algorithm that essentially reduces to one in [Heath et al. 1986], except that the context is somewhat different here. Systolic arrays can again be obtained as combined implementations of Jacobi-type SVD algorithms with matrix-vector multiplications and QR updates. As for the QSVD it is shown how additional backsubstitutions are required for computing output vectors. The backsubstitutions are intertwined with the other operations at the cost of a few additional computations, such that again the throughput remains $O\{n^0\}$. It should be stressed that while QSVD updating definitely has practical relevance, the PSVD updating problem is apparently more of an academic nature. PSVD updating seems to be of little use in practice, but the algorithm is very similar to the QSVD updating algorithm, so it's all in the day's work. On the other hand, we obtain a fairly complete set of updating algorithms for problems which had not been tackled before.

In Section 4.1 the QSVD is introduced and it is shown how it can be used for generalizing the identification procedure. QSVD updating is discussed in section 4.2. Finally, the PSVD and PSVD updating are dealt with in section 4.3. Systolic arrays as well as square root free implementations are developed.

4.1 QSVD and QSVD-based identification

The identification procedure in Chapter 1 was shown to provide consistent results for the white noise case as depicted in **Figure 1.5**, *and when a tall thin H-matrix is used, in principle with an infinite number of rows. Let us now proceed to the case where the I/O-data are corrupted by coloured noise. For instance, in* **Figure 4.1** *the use of prefiltering $F(q)$* [2] *does not change the dynamic relation between in- and output, but it does introduce a colouring of the additive (measurement) noise.*

Assume that the entries in the Hankel matrix H are subject to additive error, such that the available matrix $\bar{H}$ differs from the original H

$$\bar{H} = H + \Delta H.$$

In the sequel, we will make use of the error covariance matrix $E\{\Delta H^T \cdot \Delta H\}$ [3]*, more specifically of its Cholesky factor R_Δ (square root) which is assumed*

[2]e.g. anti-aliasing, or bandpass if a model in a specific frequency range is searched for

[3]$E\{\cdot\}$ is the expectation operator.

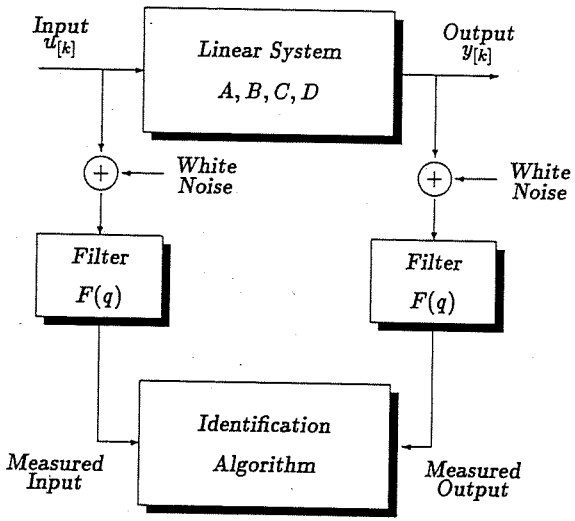


Figure 4.1: *Identification experiment set-up*

known up to a factor

$$\Delta \stackrel{def}{=} E\{\Delta H^T \cdot \Delta H\} \stackrel{def}{=} \sigma^2 \cdot R_\Delta^T \cdot R_\Delta$$

(R_Δ upper triangular).

One easily verifies that the transformed matrix $H \cdot R_\Delta^{-1}$ has an error covariance matrix equal to the identity up to a factor of proportionality. Therefore we could use the 'pre-whitened' matrix $H \cdot R_\Delta^{-1}$ instead of H for the identification –with a consistent computation of the right singular vectors for tall thin matrices, cfr. Chapter 1– and include a re-transformation with R_Δ in order to compensate for the first transformation with R_Δ^{-1}. The overall identification scheme would then deliver a consistent estimate. However, if R_Δ is singular or ill-conditioned, the matrix inverse R_Δ^{-1} should not be computed explicitly. Instead, one should use the **quotient singular value decomposition (QSVD) of the matrix pair** $\{H, R_\Delta\}$*, which in the non-singular case indeed reduces to the SVD of $H \cdot R_\Delta^{-1}$. A double QSVD identification scheme is then designed, analogously to the double SVD scheme for the white noise case*[4].

QSVD

The QSVD of a matrix pair $\{A, B\}$ was first introduced by Van Loan in [Van Loan 1976], who named it BSVD after the additional B-matrix in the decomposition. Paige and Saunders extended this to handle all possible cases [Paige & Saunders 1981]. They called it the generalized SVD, as at that time it was apparently the one and only SVD generalization. With the advent of the Product SVD [Fernando & Hammarling 1987, De Moor 1989], the Restricted SVD [Zha 1989, De Moor & Golub 1989b], and even the Structured SV, Takagi SVD, Unsigned-Unordered SVD [De Moor & Boyd 1989], the child was given a new name Quotient SVD, after the associated "matrix quotient" $A \cdot B^{-1}$.

The QSVD is a simultaneous diagonalization of any two matrices A and B with the same number of columns, by orthogonal transformations to the left and a common non-singular transformation to the right.

Theorem 1 *Quotient SVD [Van Loan 1976]*

[4] the latter being a special case of the former where every single QSVD reduces to an SVD as can easily be verified

Every pair of matrices $A_{m\times n}$ and $B_{p\times n}$, can be factored as follows

$$\begin{aligned}\underbrace{A}_{m\times n} &= \underbrace{U_A}_{m\times m}\cdot\underbrace{\Sigma_A}_{m\times n}\cdot\underbrace{X^{-1}}_{n\times n}\\ \underbrace{B}_{p\times n} &= \underbrace{U_B}_{p\times p}\cdot\underbrace{\Sigma_B}_{p\times n}\cdot\underbrace{X^{-1}}_{n\times n}\end{aligned}$$

where U_A and U_B are orthogonal, X is non-singular and Σ_A and Σ_B have the following structure

$$\Sigma_A = \begin{array}{c} \\ \rho_{ab}-\rho_b \\ \rho_a+\rho_b-\rho_{ab} \\ m-\rho_a \end{array}\begin{array}{c} \begin{array}{cccc} \rho_{ab}-\rho_b & \rho_a+\rho_b-\rho_{ab} & \rho_{ab}-\rho_a & \end{array} \\ \begin{pmatrix} I & 0 & 0 & 0 \\ 0 & \Sigma_1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\end{array}$$

$$\Sigma_B = \begin{array}{c} \\ p-\rho_b \\ \rho_a+\rho_b-\rho_{ab} \\ \rho_{ab}-\rho_a \end{array}\begin{array}{c} \begin{array}{cccc} \rho_{ab}-\rho_b & \rho_a+\rho_b-\rho_{ab} & \rho_{ab}-\rho_a & \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \Sigma_2 & 0 & 0 \\ 0 & 0 & I & 0 \end{pmatrix}\end{array}$$

$$\begin{aligned}\rho_a &= \text{rank}\{A\}\\ \rho_b &= \text{rank}\{B\}\\ \rho_{ab} &= \text{rank}\left\{\begin{bmatrix} A \\ B \end{bmatrix}\right\}\end{aligned}$$

and Σ_1 and Σ_2 are square diagonal with positive diagonal elements.

As further on we consider the updating problem when new rows are appended sequentially, we can assume that the number of rows in A and B is larger than the number of columns. For that case, one can verify that an equivalent reduction is given by

$$\begin{aligned}\underbrace{A}_{m\times n} &= \underbrace{U_A}_{m\times n}\cdot\underbrace{\Sigma_A}_{n\times n}\cdot\underbrace{X^{-1}}_{n\times n}\\ \underbrace{B}_{p\times n} &= \underbrace{U_B}_{p\times n}\cdot\underbrace{\Sigma_B}_{n\times n}\cdot\underbrace{X^{-1}}_{n\times n}\end{aligned}$$

$$\Sigma_A = \begin{array}{c} \\ \rho_{ab}-\rho_b \\ \rho_a+\rho_b-\rho_{ab} \\ \rho_{ab}-\rho_a \end{array}\begin{array}{c} \begin{array}{cccc} \rho_{ab}-\rho_b & \rho_a+\rho_b-\rho_{ab} & \rho_{ab}-\rho_a & \end{array} \\ \begin{pmatrix} I & 0 & 0 & 0 \\ 0 & \Sigma_1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\end{array} = \text{diag}\{\alpha_1,\ldots,\alpha_n\}$$

$$\Sigma_B = \begin{array}{c} \\ \rho_{ab}-\rho_b \\ \rho_a+\rho_b-\rho_{ab} \\ \rho ab-\rho_a \end{array}\begin{array}{c} \begin{array}{cccc} \rho_{ab}-\rho_b & \rho_a+\rho_b-\rho_{ab} & \rho_{ab}-\rho_a & \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \Sigma_2 & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\end{array} = \text{diag}\{\beta_1,\ldots,\beta_n\}$$

where U_A and U_B have orthonormal columns.
The pairs $\{\alpha_i,\beta_i\}$ are called the quotient singular value pairs of $\{A,B\}$, while $\frac{\alpha_i}{\beta_i}$ are the quotient singular values.

In view of both numerical stability and efficient parallel implementation, a more desirable form for the QSVD was given in [Paige & Saunders 1981]. Only orthogonal transformations are used, and A and B are reduced to triangular matrices, the corresponding rows of which are parallel:

$$\begin{aligned}\underbrace{A}_{m\times n} &= \underbrace{U_A}_{m\times n}\cdot\underbrace{\Sigma_A\cdot R}_{n\times n}\cdot\underbrace{Q^T}_{n\times n}\\ \underbrace{B}_{p\times n} &= \underbrace{U_B}_{p\times n}\cdot\underbrace{\Sigma_B\cdot R}_{n\times n}\cdot\underbrace{Q^T}_{n\times n}\end{aligned}$$

where $R_{n\times n}$ is a triangular matrix, the matrices U_A and U_B have orthonormal columns and Q is an orthogonal matrix [5] *It is readily seen that $X = Q\cdot R^{-1}$ is a QR decomposition of X.*

QSVD applications include generalized least squares problems associated with the general Gauss-Markov linear model [Paige 1985], generalized total least squares problems [Van Huffel 1989], generalized state space identification –see below– etc. Most of the time A is a data matrix –H in our identification example– while B corresponds

[5] $Q^T\cdot Q = I_{n\times n}$

to a known error covariance matrix $-R_\Delta$ in our case. The generalized methods then extend the white noise case, where $B = I_{n\times n}$ such that the QSVD reduces to an ordinary SVD.

QSVD-based identification

Although the first QSVD form (theorem 1) might not be the most desirable one, it is quite expository and instructive for our purpose. We therefore stick to this representation, keeping in mind that when it comes to actually computing things –see e.g.section 4.2– one should preferably make use of the second QSVD form [Paige 1986]. For $\{H, R_\Delta\}$ one can then also write

$$\begin{aligned} H &= U_H \cdot \Sigma_H \cdot X^{-1} \\ R_\Delta &= U_\Delta \cdot \Sigma_\Delta \cdot X^{-1}. \end{aligned}$$

It is instructive to first consider the noise free case (error covariance proportional to Δ, but with a zero factor of proportionality), and then demonstrate that the derivations still hold if there is a non-zero error contribution. If the data are noise free, then from the above QSVD definition, it follows that

$$\begin{aligned} H &= \begin{bmatrix} H_1 & H_2 \end{bmatrix} \\ &= U_H \cdot \Sigma_H \cdot X^{-1} \\ &= U_H \cdot \begin{bmatrix} \Sigma_{11} & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}^{-1}. \end{aligned}$$

$$\begin{aligned} X_{11} &\in \Re^{(mi+li)\times(2mi+n)} \\ X_{12} &\in \Re^{(mi+li)\times(2li-n)} \\ X_{21} &\in \Re^{(mi+li)\times(2mi+n)} \\ X_{22} &\in \Re^{(mi+li)\times(2li-n)} \\ \Sigma_{11} &\in \Re^{(2mi+n)\times(2mi+n)}. \end{aligned}$$

Again, from

$$H_1 \cdot X_{12} = -H_2 \cdot X_{22}$$

it follows that the column space of $H_1 \cdot X_{12}$ equals the required intersection. As $H_1 \cdot X_{12}$ contains $2li - n$ columns, only n of which are

linearly independent (dimension of the intersection), it remains to select n suitable combinations of these row vectors.
By making use of a QSVD, it is easily shown that

$$X_{12} = \begin{bmatrix} \underbrace{X_{12}^{(1)}}_{li-n} & \underbrace{X_{12}^{(2)}}_{n} & \underbrace{X_{12}^{(3)}}_{li-n} \end{bmatrix} \cdot \begin{bmatrix} I & & \\ & C & \\ & & 0 \end{bmatrix} \cdot X_\star^T$$

$$X_{22} = \begin{bmatrix} \underbrace{X_{22}^{(1)}}_{li-n} & \underbrace{X_{22}^{(2)}}_{n} & \underbrace{X_{22}^{(3)}}_{li-n} \end{bmatrix} \cdot \begin{bmatrix} 0 & & \\ & S & \\ & & I \end{bmatrix} \cdot X_\star^T$$

$$\begin{aligned} C &= diag\{c_1, \ldots, c_n\} \\ S &= diag\{s_1, \ldots, s_n\} \\ C^2 + S^2 &= I_{n\times n}. \end{aligned}$$

Clearly, only $X_{12}^{(2)}$ delivers useful combinations for the computation of the intersection, and we can take

$$\boxed{\mathcal{X} = H_1 \cdot X_{12}^{(2)}.}$$

Note that in the white noise case, this last QSVD reduces to a CS-decomposition and could then be computed from a single SVD, resulting in an overall **double SVD scheme** *for the computation of the intersection. In the general case, the computation of this intersection is carried out in a* **double QSVD scheme.**

In the second step, the system matrices can be computed from the following reduced set of equations, obtained after discarding the common orthogonal factor U_H

$$\begin{bmatrix} X_{12}^{(2)^T} \cdot \{X^{-t}\}_{m+l+1:(i+1)(m+l),1:2mi+n} \cdot \Sigma_{11} \\ \{X^{-t}\}_{mi+li+m+1:(m+l)(i+1),1:2mi+n} \cdot \Sigma_{11} \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} X_{12}^{(2)^T} \cdot \{X^{-t}\}_{1:mi+li,1:2mi+n} \cdot \Sigma_{11} \\ \{X^{-t}\}_{mi+li+1:mi+li+m,1:2mi+n} \cdot \Sigma_{11} \end{bmatrix}.$$

It remains to show that the above identification scheme delivers consistent results if the error contribution is non-zero, and if the number

of rows in H tends to infinity. For that case, it is shown that the matrix X in the QSVD is computed consistently [De Moor 1988]. As the system matrices are next computed essentially from X only[6] the model estimate is clearly consistent.

A final remark concerns trivial quotient singular values $\frac{\alpha_i}{\beta_i} = \frac{0}{0}$. As these correspond to vectors in the null space of H ($\alpha_i = 0$) [7], they should be treated as such. In other words, the column vectors in X corresponding to trivial quotient singular values should be reckoned in X_{12} and X_{22}.

The following small examples illustrate the usefulness of the generalized identification procedure.

Example 1 : *As a first example, we consider a first order single input single output system, with state space equations*

$$\begin{aligned} x_{[k+1]} &= .5x_{[k]} + u_{[k]} \\ y_{[k]} &= x_{[k]}. \end{aligned}$$

The general set-up is depicted in **Figure 4.1**. *An I/O-sequence was generated using a random input sequence, with additive white noise superimposed on both the input and the output sequence next. Finally, the noise corrupted I/O-sequence was passed through a linear filter $F(q)$, with filter characteristics as shown in* **Figure 4.2**. *The noise colouring, or equivalently the noise error covariance matrix can be determined from the filter impulse response $\delta_{[0]}, \delta_{[1]}, \delta_{[2]}, \ldots$ as follows (symmetric Toeplitz matrix)*

$$\underbrace{\Delta}_{(2li+2mi)\times(2li+2mi)} =$$

$$\begin{bmatrix} E\{f^{out}_{[k]} f^{out}_{[k]}\} & 0 & E\{f^{out}_{[k]} f^{out}_{[k+1]}\} & 0 & \cdots \\ 0 & E\{f^{out}_{[k]} f^{out}_{[k]}\} & 0 & E\{f^{out}_{[k]} f^{out}_{[k+1]}\} & \cdots \\ E\{f^{out}_{[k]} f^{out}_{[k+1]}\} & 0 & E\{f^{out}_{[k]} f^{out}_{[k]}\} & 0 & \cdots \\ 0 & E\{f^{out}_{[k]} f^{out}_{[k+1]}\} & 0 & E\{f^{out}_{[k]} f^{out}_{[k]}\} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$

[6] the matrix Σ_H in the above set of equations again imposes weights that do not influence the solution in the considered case, see also *Chapter 1*

[7] and additionally also correspond to noise free directions as $\beta_i = 0$

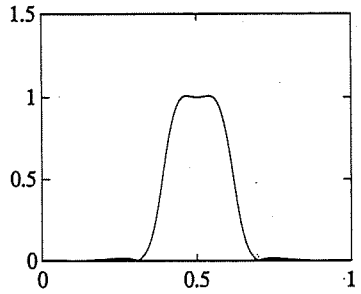


Figure 4.2: *Filter characteristic: gain vs. normalized frequency*

where $E\{.\}$ is the expectation operator, and $f^{out}_{[k]}$ is the output of the filter for a white noise input sequence $f^{in}_{[k]}$. The zero entries are due to the fact that the additive noise sequences on the in- and output ($u_{[k]}$ and $y_{[k]}$) are assumed uncorrelated. Finally $E\{f^{out}_{[k]} f^{out}_{[k+l]}\}$ can be computed as follows

$$E\{f^{out}_{[k]} f^{out}_{[k+l]}\} = \sigma^2_{noise} \sum_{r=0}^{\infty} \delta_{[r]}\delta_{[r+l]}$$

where σ^2_{noise} is a factor of proportionality that need not be known. By making use of the Cholesky factor of Δ, one can then apply the QSVD based identification procedure. **Figure 4.3** *shows the identified pole as a function of the number of rows in H, both for the QSVD based identification scheme and for the SVD based scheme, the latter not being able to compensate for the noise colouring. For small noise contributions (1%) the difference between both results is seen to be relatively small. For larger values of the signal-to-noise ratio –10% in the right-hand figure– only the QSVD scheme delivers unbiased results. The estimate clearly gets better as the number of rows in H increases (consistency).*

Example 2 : *As a second example, we again consider the same first order SISO system. If only the output is corrupted by additive white noise*

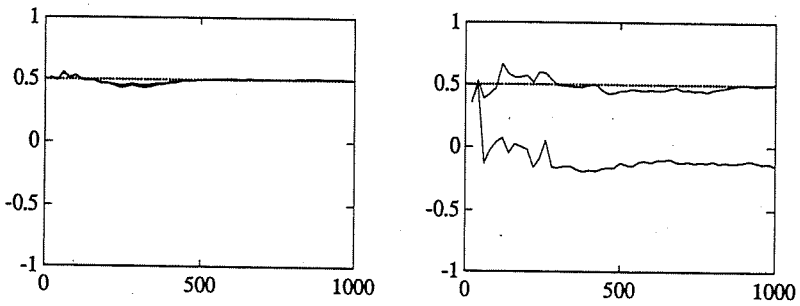


Figure 4.3: *Identified pole vs. matrix size. Upper curve: qsvd scheme. Lower curve: svd scheme. Left: 1% noise, Right: 10% noise.*

–**Figure 4.4**– *the error covariance matrix clearly equals*

$$\underbrace{\Delta}_{(2li+2mi)\times(2li+2mi)} = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}.$$

which is a rank deficient matrix. **Figure 4.5** *shows the identified pole as a function of the number of rows in H, both for the QSVD based identification scheme and for the SVD based scheme. Again, only the QSVD scheme delivers consistent results.*

As was done for the SVD-based identification scheme, the QSVD-based scheme can be worked into an adaptive scheme, where use is made of QSVD updating. This is the subject of the next section.

4.2 QSVD updating

The updating problem considered here, consists in computing the QSVD of the modified matrices, after applying exponential weighting and appending

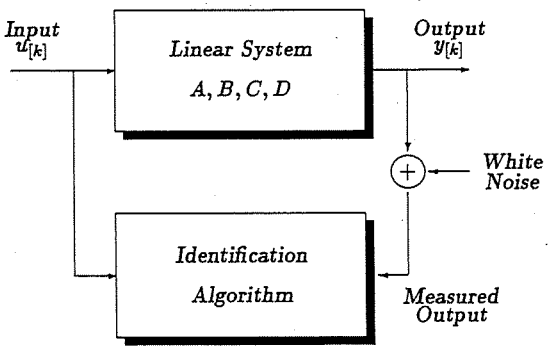


Figure 4.4: *Identication experiment set-up*

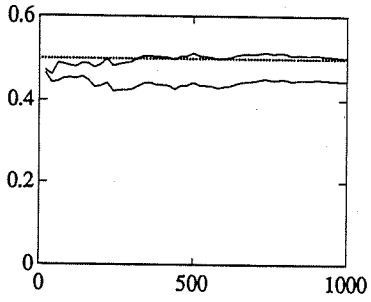


Figure 4.5: *Identified pole vs. matrix size. Upper curve: qsvd scheme. Lower curve: svd scheme.*

new rows to A and/or B:

$$\underbrace{\underline{A}}_{(m+1)\times n} = \begin{bmatrix} \lambda \cdot A \\ a^T \end{bmatrix} = \underbrace{\underline{U}_A}_{(m+1)\times n} \cdot \underline{\Sigma}_A \cdot \underline{R} \cdot \underline{Q}^T$$

$$\underbrace{\underline{B}}_{(m+1)\times n} = \begin{bmatrix} \lambda \cdot B \\ b^T \end{bmatrix} = \underbrace{\underline{U}_B}_{(m+1)\times n} \cdot \underline{\Sigma}_B \cdot \underline{R} \cdot \underline{Q}^T$$

[89]. *In on-line applications (e.g adaptive generalized state space identification), a new updating is often to be performed after each sampling, and $A_{[k]}$ and $B_{[k]}$ are defined in a recursive manner*

$$A_{[k]} = \begin{bmatrix} \lambda \cdot A_{[k-1]} \\ a^T \end{bmatrix}$$

$$B_{[k]} = \begin{bmatrix} \lambda \cdot B_{[k-1]} \\ b^T \end{bmatrix}$$

[8] See below for a note on the dual updating problem, where columns are appended instead of rows.

[9] We consider constant weighting factors, for the sake of brevity.

In most cases, the U-matrices –with growing matrix dimensions!– need not be computed explicitly, as only R, Q and Σ_A, Σ_B are of interest.

We first briefly review a Jacobi-type QSVD algorithm due to Paige [Paige 1986], and then turn it into an adaptive algorithm, by making use of interlaced QR updatings whenever new input vectors have to be worked in.

Paige's QSVD method

Briefly, the QSVD can be computed as follows. First, the matrices A and B have to be reduced to triangular form by preliminary QRD's

$$A = Q_A \cdot \begin{bmatrix} R_A \\ 0 \end{bmatrix}$$

$$B = Q_B \cdot \begin{bmatrix} R_B \\ 0 \end{bmatrix}$$

where R_A and R_B are square upper triangular, and Q_A and Q_B are orthogonal. The QSVD then obviously follows from the QSVD of R_A and R_B

$$A = Q_A \cdot \begin{bmatrix} \Theta_A \cdot \Sigma_A \cdot R \cdot Q^T \\ 0 \end{bmatrix} = \underbrace{Q_A \cdot \begin{bmatrix} \Theta_A \\ 0 \end{bmatrix}}_{U_A} \cdot \Sigma_A \cdot R \cdot Q^T$$

$$B = Q_B \cdot \begin{bmatrix} \Theta_B \cdot \Sigma_B \cdot R \cdot Q^T \\ 0 \end{bmatrix} = \underbrace{Q_B \cdot \begin{bmatrix} \Theta_B \\ 0 \end{bmatrix}}_{U_B} \cdot \Sigma_B \cdot R \cdot Q^T.$$

The QSVD of $\{R_A, R_B\}$ is computed by carrying out an iterative procedure, where a series of Givens transformations is applied to R_A and R_B in order to yield triangular factors with parallel rows $\Sigma_A \cdot R$ and $\Sigma_B \cdot R$ [10]. *In other words, a sequence of triangular matrices $R_{A_{[k]}}, R_{B_{[k]}}$ is generated via the recurrence*

$$R_{A_{[0]}} \Leftarrow R_A$$
$$R_{B_{[0]}} \Leftarrow R_B$$

[10] the i^{th} row in the first matrix is parallel to the i^{th} row in the second matrix, for all i.

for $k = 1, \ldots, k_{\max}$

$$R_{A_{[k]}} \Leftarrow \Theta_{A_{[k]}}^T \cdot R_{A_{[k-1]}} \cdot Q_{[k]}$$
$$R_{B_{[k]}} \Leftarrow \Theta_{B_{[k]}}^T \cdot R_{B_{[k-1]}} \cdot Q_{[k]}$$

end

with $\Theta_{A_{[k]}}$, $\Theta_{B_{[k]}}$ and $Q_{[k]}$ orthogonal transformations to be defined next. After a certain number of iterations $k_{\max}$ the triangular factors have parallel rows –up to the relative precision ϵ–

$$\Sigma_A \cdot R \overset{\epsilon}{\simeq} R_{A_{[k_{\max}]}}$$
$$\Sigma_B \cdot R \overset{\epsilon}{\simeq} R_{B_{[k_{\max}]}}.$$

The orthogonal transformations are accumulated to give

$$Q \simeq Q_{[1]} \cdots Q_{[k_{\max}]}$$
$$\Theta_A \simeq \Theta_{A_{[1]}} \cdots \Theta_{A_{[k_{\max}]}}$$
$$\Theta_B \simeq \Theta_{B_{[1]}} \cdots \Theta_{B_{[k_{\max}]}}.$$

$\Theta_{A_{[k]}}$, $\Theta_{B_{[k]}}$ and $Q_{[k]}$ are embeddings of plane rotations. To every k, there correspond rotation angles $\theta_{A_{[k]}}$, $\theta_{B_{[k]}}$ and $\phi_{[k]}$ and a pivot index i such that

$$\{\Theta_{A_{[k]}}\}_{i,i+1} = \begin{bmatrix} \cos\theta_{A_{[k]}} & \sin\theta_{A_{[k]}} \\ -\sin\theta_{A_{[k]}} & \cos\theta_{A_{[k]}} \end{bmatrix}$$
$$\{\Theta_{B_{[k]}}\}_{i,i+1} = \begin{bmatrix} \cos\theta_{B_{[k]}} & \sin\theta_{B_{[k]}} \\ -\sin\theta_{B_{[k]}} & \cos\theta_{B_{[k]}} \end{bmatrix}$$
$$\{Q_{[k]}\}_{i,i+1} = \begin{bmatrix} \cos\phi_{[k]} & \sin\phi_{[k]} \\ -\sin\phi_{[k]} & \cos\phi_{[k]} \end{bmatrix}$$

where $\{M\}_{i,i+1}$ again denotes the 2×2 matrix on the intersection of rows $i, i+1$ and columns $i, i+1$ of M. The rotation angles should be chosen as follows. Let us first consider the simple case where R_B is nonsingular [11]*. The QSVD of R_A and R_B then corresponds to the SVD of $R_A \cdot R_B^{-1}$*

$$R_A \cdot R_B^{-1} = \Theta_A \cdot \{\Sigma_A \cdot \Sigma_B^{-1}\} \cdot \Theta_B^T.$$

[11] We sometimes omit time indices for short notation.

As both R_A and R_B are upper triangular, so are R_B^{-1} and $R_A \cdot R_B^{-1}$. As a consequence, one has

$$\{R_A \cdot R_B^{-1}\}_{i,i+1} = \{R_A\}_{i,i+1} \cdot \{R_B\}_{i,i+1}^{-1}.$$

This equation allows for the implicit application of the parallel SVD algorithm to the upper triangular matrix $R_A \cdot R_B^{-1}$, without explicitly forming this matrix. The rotations $\Theta_{A_{[k-1]}}$ and $\Theta_{B_{[k-1]}}$ that constitute Θ_A and Θ_B respectively, can thus be computed from a 2×2 SVD

$$\begin{bmatrix} s_{i,i} & 0 \\ 0 & s_{i+1,i+1} \end{bmatrix} \Leftarrow \underbrace{\begin{bmatrix} \cos\theta_A & \sin\theta_A \\ -\sin\theta_A & \cos\theta_A \end{bmatrix}}_{\{\Theta_{A_{[k]}}\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} r_{i,i}^A & r_{i,i+1}^A \\ 0 & r_{i+1,i+1}^A \end{bmatrix}}_{\{R_{A_{[k-1]}}\}_{i,i+1}} \cdots$$
$$\cdots \underbrace{\begin{bmatrix} r_{i,i}^B & r_{i,i+1}^B \\ 0 & r_{i+1,i+1}^B \end{bmatrix}^{-1}}_{\{R_{B_{[k-1]}}\}_{i,i+1}^{-1}} \cdot \underbrace{\begin{bmatrix} \cos\theta_B & \sin\theta_B \\ -\sin\theta_B & \cos\theta_B \end{bmatrix}}_{\{\Theta_{B_{[k]}}\}_{i,i+1}}$$

where the elements $s_{i,i}$ and $s_{i+1,i+1}$ need not be computed explicitly. From

$$\{\Theta_{A_{[k]}}^T \cdot R_{A_{[k-1]}}\}_{i,i+1} = diag\{s_{i,i}, s_{i+1,i+1}\} \cdot \{\Theta_{B_{[k]}}^T \cdot R_{B_{[k-1]}}\}_{i,i+1}$$

it follows that the corresponding rows in the 2×2 matrices $\{\Theta_{A_{[k]}}^T \cdot R_{A_{[k-1]}}\}_{i,i+1}$ and $\{\Theta_{B_{[k]}}^T \cdot R_{B_{[k-1]}}\}_{i,i+1}$ are parallel, so that one Givens rotation $Q_{[k]}$ can upper-triangularize both matrices simultaneously.

$$\begin{bmatrix} \tilde{r}_{i,i}^A & \tilde{r}_{i,i+1}^A \\ 0 & \tilde{r}_{i+1,i+1}^A \end{bmatrix} = \{\Theta_{A_{[k]}}^T \cdot R_{A_{[k-1]}} \cdot Q_{[k]}\}_{i,i+1}$$

$$\begin{bmatrix} \tilde{r}_{i,i}^B & \tilde{r}_{i,i+1}^B \\ 0 & \tilde{r}_{i+1,i+1}^B \end{bmatrix} = \{\Theta_{B_{[k]}}^T \cdot R_{B_{[k-1]}} \cdot Q_{[k]}\}_{i,i+1}.$$

$R_{A_{[k]}}$ and $R_{B_{[k]}}$ thus remain upper triangular at any stage, while $R_{A_{[k]}} \cdot R_{B_{[k]}}^{-1}$ is being reduced to a diagonal matrix, without explicitly being formed.

As soon as $R_{A_{[k]}} \cdot R_{B_{[k]}}^{-1}$ *is diagonal,* $R_{A_{[k]}}$ *and* $R_{B_{[k]}}$ *obviously have parallel rows as requested*

$$R_{A_{[k]}} = \Sigma_A \cdot R$$

$$R_{B_{[k]}} = \Sigma_B \cdot R.$$

The general case where R_B *can be singular requires slight modifications for the computation of the rotation angles. Instead of using* $\{R_B\}_{i,i+1}^{-1}$, *one should then make use of* $adj\{R_B\}_{i,i+1}$ *and proceed along much the same lines. In [Paige 1986] it is shown that such a procedure remarkably enough yields consistent results.*

Each iteration thus essentially reduces to a 2×2 *QSVD on the main diagonal, parallelizing the rows of* $\{R_A\}_{i,i+1}$ *and* $\{R_B\}_{i,i+1}$. *By making use of either inner rotations + permutations or outer rotations on the one hand, and an elegant pipelining of forward and backward sweeps on the other hand, an efficient implementation can again be obtained [Stewart 1985, Luk 1985], wherewith the pivot index successively takes up all possible values*

$$i = 1, 2, \ldots, n-1$$

in a cyclic manner and 2×2 *QSVD's on the main diagonal are performed accordingly. After a certain number of such sequences, the R-factors will be reduced to triangular factors with parallel rows.*

QSVD updating

In an on-line environment, the rows of A and/or B become sequentially available. As new rows become available, these should be included in the R-factors, by making use of QR updating. After each QR updating, the implicit iterative diagonalization process of $R_A \cdot R_B^{-1}$ *is resumed.*

Briefly, new data vectors can be worked in as follows. Suppose that at a certain time step $k-1$, *the triangular factors are reduced to* $R_{A_{[k-1]}}$ *and* $R_{B_{[k-1]}}$ *(with approximately parallel rows)* [12] *and the corresponding Q-matrix is denoted as* $Q_{[k-1]}$:

$$A_{[k-1]} = U_{A_{[k-1]}} \cdot R_{A_{[k-1]}} \cdot Q_{[k-1]}^T$$

[12] Note that k is now a time index, instead of an iteration index.

$$B_{[k-1]} = U_{B_{[k-1]}} \cdot R_{B_{[k-1]}} \cdot Q_{[k-1]}^T$$

[13]. *After appending new rows* $a_{[k]}^T$ *and/or* $b_{[k]}^T$, *one has a decomposition of the type*

$$A_{[k]} = \begin{bmatrix} \lambda \cdot A_{[k-1]} \\ a_{[k]}^T \end{bmatrix} = \begin{bmatrix} \boxed{U_{A_{[k-1]}}} & \boxed{\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}} \\ \boxed{0 \ \ldots \ 0} & \boxed{1} \end{bmatrix} \cdot \begin{bmatrix} \lambda \cdot R_{A_{[k-1]}} \\ a_{[k]}^T \cdot Q_{[k-1]} \end{bmatrix} \cdot Q_{[k-1]}^T$$

$$B_{[k]} = \begin{bmatrix} \lambda \cdot B_{[k-1]} \\ b_{[k]}^T \end{bmatrix} = \begin{bmatrix} \boxed{U_{B_{[k-1]}}} & \boxed{\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}} \\ \boxed{0 \ \ldots \ 0} & \boxed{1} \end{bmatrix} \cdot \begin{bmatrix} \lambda \cdot R_{B_{[k-1]}} \\ b_{[k]}^T \cdot Q_{[k-1]} \end{bmatrix} \cdot Q_{[k-1]}^T$$

so that clearly, the R-factors have to be updated with the transformed input vectors $\tilde{a}_{[k]}^T = a_{[k]}^T \cdot Q_{[k-1]}$ *and* $\tilde{b}_{[k]}^T = b_{[k]}^T \cdot Q_{[k-1]}$, *resulting in*

$$A_{[k]} = \underbrace{\begin{bmatrix} \boxed{U_{A_{[k-1]}}} & \boxed{\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}} \\ \boxed{0 \ \ldots \ 0} & \boxed{1} \end{bmatrix} \cdot Q_{A_{[k]}}}_{\tilde{U}_{A_{[k]}}} \cdot \begin{bmatrix} \tilde{R}_{A_{[k]}} \\ 0 \end{bmatrix} \cdot Q_{[k-1]}^T$$

$$B_{[k]} = \underbrace{\begin{bmatrix} \boxed{U_{B_{[k-1]}}} & \boxed{\begin{matrix} 0 \\ \vdots \\ 0 \end{matrix}} \\ \boxed{0 \ \ldots \ 0} & \boxed{1} \end{bmatrix} \cdot Q_{B_{[k]}}}_{\tilde{U}_{B_{[k]}}} \cdot \begin{bmatrix} \tilde{R}_{B_{[k]}} \\ 0 \end{bmatrix} \cdot Q^{\{k\}^T}.$$

Note that these QR updatings do not alter the matrix $Q_{[k-1]}$. *New rows can thus be worked in by performing matrix-vector multiplications and QR updates. After each update the QSVD process is resumed. Similar to the SVD updating procedure, a QSVD updating procedure can be devised as follows, with* $n-1$ *QSVD steps after each*

[13] only $R_{A_{[k-1]}}$, $R_{B_{[k-1]}}$ and $Q_{[k-1]}$ are stored.

QR update as an arbitrary choice :

Initialization

$$\begin{aligned} Q &\Leftarrow I_{n\times n} \\ R_A &\Leftarrow O_{n\times n} \\ R_B &\Leftarrow O_{n\times n} \end{aligned}$$

Loop
for $k = 1, \ldots, \infty$
input new measurement vectors $a_{[k]}, b_{[k]}$
1. matrix-vector multiplications

$$\tilde{a}_{[k]} = a_{[k]}^T \cdot Q$$

$$\tilde{b}_{[k]} = b_{[k]}^T \cdot Q$$

2. QR updatings

$$\begin{bmatrix} R_A \\ 0 \end{bmatrix} \Leftarrow Q_{A_{[k]}}^T \cdot \begin{bmatrix} \lambda \cdot R_A \\ \tilde{a}_{[k]}^T \end{bmatrix}$$

$$\begin{bmatrix} R_B \\ 0 \end{bmatrix} \Leftarrow Q_{B_{[k]}}^T \cdot \begin{bmatrix} \lambda \cdot R_B \\ \tilde{b}_{[k]}^T \end{bmatrix}$$

3. QSVD steps
for $i = 1, \ldots, n-1$

$$\begin{aligned} R_A &\Leftarrow \Theta_{A_{[k,i]}}^T \cdot R_A \cdot Q_{[k,i]} \\ R_B &\Leftarrow \Theta_{B_{[k,i]}}^T \cdot R_B \cdot Q_{[k,i]} \\ Q &\Leftarrow Q \cdot Q_{[k,i]} \end{aligned}$$

end
end

Adaptive systems identification

Much like it was the case for the SVD-based scheme, the QSVD-based identification algorithm is readily converted into an adaptive one for time-varying systems, by making use of QSVD updating and exponential weighting. If the error covariance matrix is time-invariant, only one matrix ($A = H$) is updated with new rows. Again, H is never constructed explicitly, but only the factors Σ_H, Σ_Δ and R, Q are stored and updated. A complete algorithmic description is given in [Moonen and Vandewalle 1990b].

Systolic QSVD updating

The QSVD array in [Luk 1985] of quite similar to the SVD array of [Luk 1986a]. The triangular part now contains two triangular factors instead of one, while the square part contains the matrix Q instead of V –what's in a name. The processors on the main diagonal perform 2×2 QSVD's instead of SVD's, and the off-diagonal processors apply transformations to R_A-elements and R_B-elements [14]. In principle, nothing is changed.

Similar to the SVD updating array in Chapter 3, one can therefore straightforwardly design a QSVD updating array –**Figure 4.6.** *The propagation of column and row transformations remains unchanged with respect to the SVD case, see Figure 3.8. If both R_A and R_B are to be updated with new rows, the filled boxes in Figure 3.8 actually represent two data vectors, viz. a and b. These data vectors are fed in to the left of the square array, where the matrix-vector products $\tilde{a}^T = a^T \cdot Q$ and $\tilde{b}^T = b^T \cdot Q$ are performed. The resulting vectors $\tilde{a}^T$, $\tilde{b}^T$ become available at the top end, and are bounced back into the array, towards the triangular array, which performs the QSVD reduction with interlaced QR updatings. Also if only one matrix is to be updated with new rows, the systolic implementation is straightforward. Cell descriptions are similar to the SVD case and are omitted for the sake of brevity. See Chapter 3 for further details.*

The QSVD is an efficient tool for computing e.g. the generalized total least squares (GTLS) solution for systems of linear equations, when the covariance matrix of errors in the rows is known up to a scale factor [Van Huffel 1989]. For that case, only one triangular factor, R_A say, is to be updated with new data vectors. The second one

[14] Note that the row transformations applied to R_A and R_B are different.

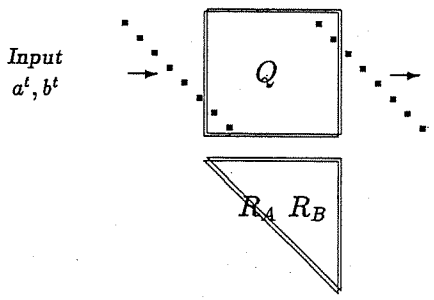


Figure 4.6: *QSVD updating array*

corresponds to the Cholesky factor of the error covariance matrix $\Delta = R_\Delta^T \cdot R_\Delta$, *which is assumed fixed. Although the QSVD updating itself is straightforward, generating e.g. GTLS solution vectors is somewhat more involved. We consider only the easy case where* $R_B = R_\Delta$ *is non-singular. An approximate GTLS solution can then be obtained as the column of* $X = Q \cdot R_\Delta^{-1}$ *corresponding to the smallest quotient singular value* $\frac{\alpha_i}{\beta_i}$. *Much like it was done in the SVD case, we can generate a vector* t *on the main diagonal with all its components equal to zero, except for one component equal to "1" at the position of the smallest quotient singular value. The ratios of diagonal elements in* R_A *and* R_Δ *are "reasonable" estimates for the quotient singular values, so that we can make use of a threshold function:*

$$t_i = 1 \quad \textit{iff} \quad r_{ii} < \gamma$$

$$t_i = 0 \quad \textit{iff} \quad r_{ii} > \gamma$$

$$r_{ii} = \frac{r_{ii}^A}{r_{ii}^\Delta}$$

where γ *satisfies*

$$\frac{\alpha_{n-1}}{\beta_{n-1}} > \gamma > \frac{\alpha_n}{\beta_n}$$

and is assumed to be known a priori. The output vector then equals $x = Q \cdot R_\Delta^{-1} \cdot t$, *the computation of which thus requires a triangular backsolve, followed by a matrix-vector multiplication. The backsubstitution necessarily starts off at the bottom of the array, with intermediate results being propagated upwards, unlike e.g. the* QR *updates which run in precisely the opposite direction. This particularly seems to rule out an elegant implementation.*

Much the same problem occurred with the implementation of interlaced QR *updating and* $(Q)SVD$ *reduction : the* QR *updates run from the upper-left corner to the lower-right corner, while the odd-evens run from the diagonal to upper-right. Interlacing could be carried out if the odd-evens were interpreted as pipelined sequences of* $n-1$ *rotations, starting from the upper-left corner and propagating downwards along the diagonal. The same strategy can now be applied to the* R_Δ*-factor. This matrix is only affected by odd-evens, as no input vectors have to be worked in. These odd-evens can then be interpreted as pipelined sequences of* $n-1$ *rotations, starting at the lower right corner and propagating upwards along the diagonal. In other words, the original sequence of transformations acting upon* R_Δ, *viz.*

for $k = 1, 2, \ldots, \infty$
  for $i = 1, 2, \ldots, n-1$

$$R_\Delta \Leftarrow \Theta_{\Delta_{[k,i]}} \cdot R_\Delta \cdot Q_{[k,i]}$$

  end
end

can implicitly be rearranged into

for $k = 1, 2, \ldots, \infty$
  for $i = n-1, n-2, \ldots, 1$

$$R_\Delta \Leftarrow \Theta_{\Delta_{[k,i]}} \cdot R_\Delta \cdot Q_{[k,i]}$$

  end
end

which is an equivalent ordering, except for a different start-up phase. Backsubstitutions can then readily be inserted, as detailed next. Note that with a sequential algorithm, one would obtain a column of $Q_{[k]} \cdot$

$R^{-1}_{\Delta_{[k]}}$ as a solution at time step k. For the parallel implementation the obtained solution corresponds to an intermediate version $Q_{[k]} \cdot R^{-1}_{\Delta_{[k]}}$. This latter equals $Q_{[k]} \cdot R^{-1}_{\Delta_{[k]}}$ up to a limited number of transformations. As we consider slowly time-varying systems, the error is assumed reasonably small, as was also verified by software simulation.

Figure 4.7 shows how successive output vectors can be generated. On the main diagonal, components of $t^* = R^{-1}_{\Delta} \cdot t$ are generated –see below– which are propagated upwards. Off-diagonal processors in both the triangular part and the square part receive such components from their lower neighbours, and intermediate results from their right neighbours. The products $t^*_j \cdot q_{ij}$ or $t^*_j \cdot r^{\Delta}_{ij}$ are added to the intermediate results. The adjusted intermediate results are passed on to the left neighbour, while the t^*-components are passed on upwards. In this way, the diagonal processors receive the product

$$\sum_{j=i+1}^{n} t^*_j \cdot r^{\Delta}_{ij}$$

that is used to compute the next t^*-component

$$t^*_i = \{t_i - \sum_{j=i+1}^{n} t^*_j \cdot r^{\Delta}_{ij}\} \cdot \{r^{\Delta}_{ii}\}^{-1}$$

where t_i is generated by applying the aforementioned threshold function, see descriptions 1, 2 and 3. Notice that row transformations applied to R_{Δ} in the QSVD steps, should be applied to intermediate results as well, much like column transformations on R_A are to be applied to the intermediate QR updating results. Again this is a computational overhead, introduced whenever different computational fronts cross.

The output vectors run out at the left-hand side of the array in a skewed fashion as indicated. If necessary, they can be bounced back into the array, and propagated to the right in order to make them available at the right-hand side –as was indicated in Figure 4.6. Again, it should be stressed that all operations –viz. the QSVD updating and the output vector generation including the triangular backsolve– are carried out continuously and simultaneously.

Figure a

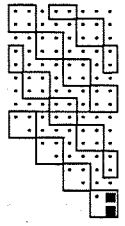


Figure b

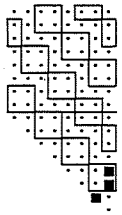


Figure c

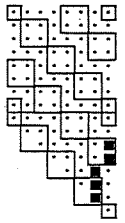


Figure d

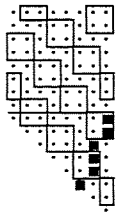


Figure e

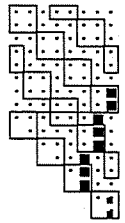


Figure f

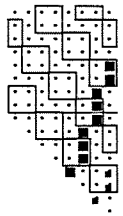


Figure g

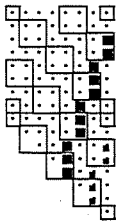


Figure h

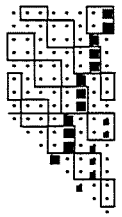


Figure i

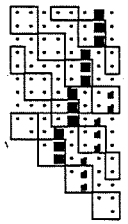


Figure j

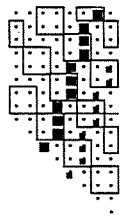


Figure k

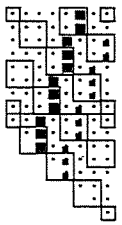
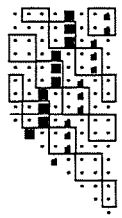


Figure l



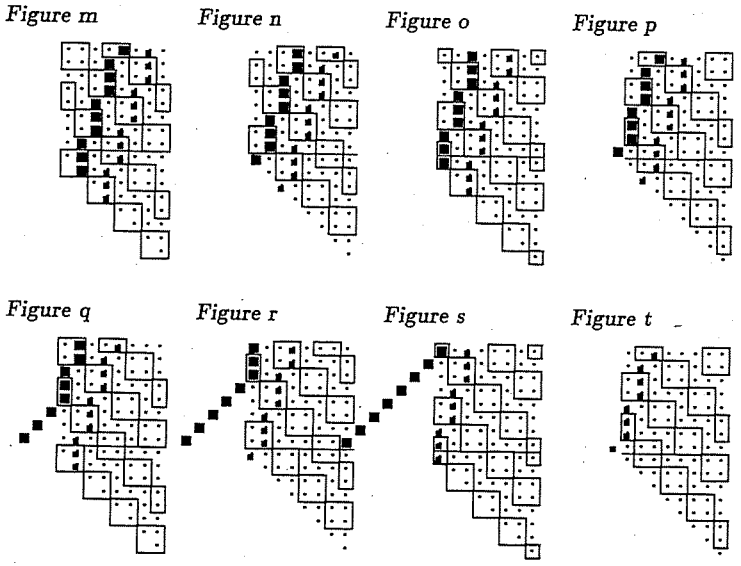
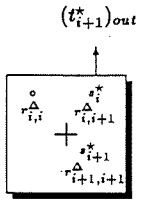


Figure 4.7: *Output vector generation*



DESCRIPTION 1 :
BOUNDARY CELL R-FACTOR

(QSVD & QR steps omitted)

Generate $t^\star_{i+1}$

$$\textit{if} \quad \frac{r^\triangle_{i+1,i+1}}{r^\triangle_{i+1,i+1}} \leq \gamma \quad \textit{then} \quad t_{i+1} = 1$$
$$\textit{else} \quad t_{i+1} = 0$$

$$t^\star_{i+1} = (t_{i+1} - s^\star_{i+1}) \cdot (r^\triangle_{i+1,i+1})^{-1}$$

Update Intermediate Result

$$s^\star_i \leftarrow s^\star_i + r^\triangle_{i,i+1} \cdot t^\star_{i+1}$$

Propagate Intermediate Result

$$\circ \leftarrow s^\star_i$$

Propagate $t^\star_{i+1}$

$$(t^\star_{i+1})_{out} \leftarrow t^\star_{i+1}$$

DESCRIPTION 2 :
INTERNAL CELL R-FACTOR

(QSVD & QR steps omitted)

Input $t^\star_{q+1}$

$$t^\star_{q+1} \leftarrow (t^\star_{q+1})_{in}$$

Update Intermediate Results

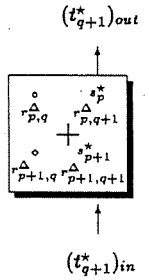
$$s^\star_p \leftarrow s^\star_p + r^\triangle_{p,q+1} \cdot t^\star_{q+1}$$
$$s^\star_{p+1} \leftarrow s^\star_{p+1} + r^\triangle_{p+1,q+1} \cdot t^\star_{q+1}$$

Propagate Intermediate Results

$$\circ \leftarrow s^\star_p$$
$$\diamond \leftarrow s^\star_{p+1}$$

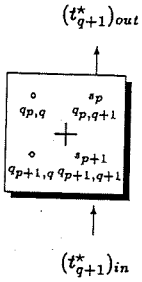
Propagate $t^\star_{q+1}$

$$(t^\star_{q+1})_{out} \leftarrow t^\star_{q+1}$$



DESCRIPTION 3 :
INTERNAL CELL Q-MATRIX

(QSVD & QR steps omitted)



Input t^*_{q+1}

$t^*_{q+1} \leftarrow (t^*_{q+1})_{in}$

Update Intermediate Results

$$s_p \leftarrow s_p + q_{p,q+1} \cdot t^*_{q+1}$$
$$s_{p+1} \leftarrow s_{p+1} + q_{p+1,q+1} \cdot t^*_{q+1}$$

Propagate Intermediate Results

$$\circ \leftarrow s_p$$
$$\diamond \leftarrow s_{p+1}$$

Propagate t^*_{q+1}

$(t^*_{q+1})_{out} \leftarrow t^*_{q+1}$

Square root free QSVD updating

In Chapter 3, it was shown how a generalized Gentleman procedure with a two-sided factorization of the R-factor, can be combined with approximate SVD schemes in order to obtain a square root free SVD updating algorithm, with an additional saving in the number of multiplications. The derivation of a square root free QSVD updating algorithm proceeds along the same lines.

Briefly, a square root free procedure can be obtained as follows. At a certain time step, the triangular factors are reduced to R_A *and* R_B*, with almost parallel rows, and both are stored in factorized form*

$$R_A = D^{\frac{1}{2}}_{row_A} \cdot \bar{R}_A \cdot D^{\frac{1}{2}}_{col}$$

$$R_B = D^{\frac{1}{2}}_{row_B} \cdot \bar{R}_B \cdot D^{\frac{1}{2}}_{col}$$

[15]. *Note that the column scaling is the same for* R_A *and* R_B*. In a systolic array, the diagonal matrices* D_{row_A}*,* D_{row_B} *and* D_{col} *are*

[15] $\bar{R}_A$, $\bar{R}_B$ and D_{row_A}, D_{row_B}, D_{col} are stored.

obviously stored in the processor elements on the main diagonal. The square part of the array contains a scaled version $\bar{Q}$ *of the orthogonal matrix* Q

$$Q = \bar{Q} \cdot D^{\frac{1}{2}}_{col}.$$

The reason for this is the same as for the SVD case –see Chapter 3.

The only real difference concerns the 2×2 *QSVD's on the main diagonal. As was explained earlier, in a first step an SVD of* $\{R_A\}_{i,i+1} \cdot \{R_B\}^{-1}_{i,i+1}$ *is computed, while the second step consists in performing one single Givens rotation. As for the SVD one immediately sees that*

$$\begin{aligned}
&\{R_A\}_{i,i+1} \cdot \{R_B\}^{-1}_{i,i+1} \\
&= \{D^{\frac{1}{2}}_{row_A}\}_{i,i+1} \cdot \{\bar{R}_A\}_{i,i+1} \cdot \overbrace{\{D^{\frac{1}{2}}_{col}\}_{i,i+1} \cdot}^{I_{2\times 2}} \cdot \\
&\qquad \cdots \{D^{-\frac{1}{2}}_{col}\}_{i,i+1} \cdot \{\bar{R}_B\}^{-1}_{i,i+1} \cdot \{D^{-\frac{1}{2}}_{row_B}\}_{i,i+1} \\
&= \{D^{\frac{1}{2}}_{row_A}\}_{i,i+1} \cdot \{\bar{R}_A\}_{i,i+1} \cdot \{\bar{R}_B\}^{-1}_{i,i+1} \{D^{-\frac{1}{2}}_{row_B}\}_{i,i+1}
\end{aligned}$$

boils down to an SVD with a 2-sided factorization, which can be computed approximately without square roots, as was detailed in Chapter 3. If $\mathrm{adj}\{R_B\}_{i,i+1}$ *is used instead of* $\{R_B\}^{-1}_{i,i+1}$*, the above formula becomes*

$$\begin{aligned}
&\{R_A\}_{i,i+1} \cdot \mathrm{adj}\{R_B\}_{i,i+1} \\
&= \{D^{\frac{1}{2}}_{row_A}\}_{i,i+1} \cdot \{\bar{R}_A\}_{i,i+1} \cdot \overbrace{\{D^{\frac{1}{2}}_{col}\}_{i,i+1} \cdot}^{d_i^{col\,\frac{1}{2}} d_{i+1}^{col\,\frac{1}{2}} \cdot I_{2\times 2}} \cdot \\
&\qquad \cdots \mathrm{adj}\{D^{\frac{1}{2}}_{col}\}_{i,i+1} \cdot \mathrm{adj}\{\bar{R}_B\}_{i,i+1} \cdot \mathrm{adj}\{D^{\frac{1}{2}}_{row_B}\}_{i,i+1} \\
&= \{D^{\frac{1}{2}}_{row_A}\}_{i,i+1} \cdot \{\bar{R}_A\}_{i,i+1} \cdot \mathrm{adj}\{\bar{R}_B\}_{i,i+1} \cdot \{d_i^{col\,\frac{1}{2}} d_{i+1}^{col\,\frac{1}{2}} \cdot \mathrm{adj}\{D^{\frac{1}{2}}_{row_B}\}_{i,i+1}\}.
\end{aligned}$$

Again, this boils down to an SVD with a 2-sided factorization that can be computed approximately without square roots.

In an unfactored form, this approximate SVD reads as follows:

$$\begin{bmatrix} s_{i,i} & s_{i,i+1} \\ 0 & s_{i+1,i+1} \end{bmatrix} \Leftarrow \underbrace{\begin{bmatrix} \cos\theta_A & \sin\theta_A \\ -\sin\theta_A & \cos\theta_A \end{bmatrix}}_{\{\Theta_{A_{[k]}}\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} r^A_{i,i} & r^A_{i,i+1} \\ 0 & r^A_{i+1,i+1} \end{bmatrix}}_{\{R_{A_{[k-1]}}\}_{i,i+1}} \cdots$$

$$\cdots \underbrace{\begin{bmatrix} r^B_{i,i} & r^B_{i,i+1} \\ 0 & r^B_{i+1,i+1} \end{bmatrix}^{-1}}_{\{R_{B_{[k-1]}}\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} \cos\theta_B & \sin\theta_B \\ -\sin\theta_B & \cos\theta_B \end{bmatrix}}_{\{\Theta_{B_{[k]}}\}_{i,i+1}}$$

–or else with "adj" instead of the inverse– from which it then follows that the second rows [16] *in* $\{\Theta^T_{A_{[k]}} \cdot R_{A_{[k-1]}}\}_{i,i+1}$ *and* $\{\Theta^T_{B_{[k]}} \cdot R_{B_{[k-1]}}\}_{i,i+1}$ *are parallel, so that again one rotation* $Q_{[k]}$ *can upper-triangularize both matrices at the same time. For this Givens transformation, it suffices to simply apply Gentleman's procedure.*

Again, it follows that on a systolic array, a square root free QSVD updating algorithm imposes hardly any changes compared to the original procedure.

A note on the dual QSVD updating problem

So far, we considered the updating problem where new rows are appended to the matrices A and/or B. Alternatively, one could also append columns to A and B, and try to devise a similar updating algorithm. However, this dual QSVD updating problem seems to be much more difficult to solve. The updating problem now consists in computing the modified QSVD

$$\underline{A} = \begin{bmatrix} \lambda \cdot A & a \end{bmatrix} = \underline{U}_A \cdot \underline{\Sigma}_A \cdot \underline{R} \cdot \underline{Q}^T$$

$$\underline{B} = \begin{bmatrix} \lambda \cdot B & b \end{bmatrix} = \underline{U}_B \cdot \underline{\Sigma}_B \cdot \underline{R} \cdot \underline{Q}^T$$

by making use of the original QSVD of A and B. This time U_A, U_B, Σ_A, Σ_B *and R are important. The matrix Q has growing dimensions, and is not computed explicitly. Suppose that A and B are stored in*

[16] Note that in general the first rows are not parallel, as $s_{i,i+1}$ is nonzero for the approximate SVD computations.

triangular form. The main difficulty appears to be the QR updating of the triangular factors R_A *and* R_B *after appending new columns, as the orthogonal updating transformation to the right necessarily has to be the same for both* R_A *and* R_B *(Q-matrix!). On the other hand, if use is made of both left- and right transformations for these updates, the available approximations for* U_A *and* U_B *can be changed significantly, so that many more QSVD steps are needed afterwards in order to restore the approximation.*

Fortunately, any updating algorithm of this kind, would apparently be of little use in practice. In general, the QSVD for two matrices $A_{m\times n}$ *and* $B_{p\times n}$, *with more columns than rows* $(n > p, m)$ *looks as follows:*

$$\underbrace{A}_{m\times n} = \underbrace{U_A}_{m\times m} \cdot \underbrace{\Sigma_A}_{m\times k} \cdot \begin{bmatrix} \underbrace{R}_{k\times k} & \underbrace{0}_{k\times n-k} \end{bmatrix} \cdot Q^T$$

$$\underbrace{B}_{p\times n} = \underbrace{U_B}_{p\times p} \cdot \underbrace{\Sigma_B}_{p\times k} \cdot \begin{bmatrix} \underbrace{R}_{k\times k} & \underbrace{0}_{k\times n-k} \end{bmatrix} \cdot Q^T$$

$$\Sigma_A = \begin{bmatrix} \underbrace{I}_{r\times r} & & \\ & \underbrace{S_A}_{s\times s} & \\ & & \underbrace{0}_{m-r-s\times k-r-s} \end{bmatrix}$$

$$\Sigma_B = \begin{bmatrix} \underbrace{0}_{p-k+r\times r} & & \\ & \underbrace{S_B}_{s\times s} & \\ & & \underbrace{I}_{k-r-s\times k-r-s} \end{bmatrix}$$

$$S_A^2 + S_B^2 = I$$

where $k = \text{rank}\{A^T, B^T\}$ *and s is the dimension of the intersection of the row spaces of A and B. If A and B contain measured data, with a signal-to-noise ratio of e.g. SN≃ 100, then –due to the noise– A and B as well as* $\{A^T, B^T\}$ *will have full row rank* $(k = m + p)$, *with a fairly small condition number of approximately 100. The QSVD then*

follows somewhat trivially from an RQ factorization:

$$\begin{bmatrix} A \\ B \end{bmatrix} = \underbrace{R}_{m+p \times m+p} \cdot Q^T$$

which can be written in a QSVD form

$$\begin{aligned} A &= \underbrace{I}_{m \times m} \cdot \begin{bmatrix} I & 0 \end{bmatrix} \cdot R \cdot Q^T \\ B &= \underbrace{I}_{p \times p} \cdot \begin{bmatrix} 0 & I \end{bmatrix} \cdot R \cdot Q^T. \end{aligned}$$

For practical applications –e.g. systems identification from measured I/O-data– the most valuable information that can be drawn from such a QSVD, would be the intersection of the row spaces of A and B. Due to the noise, this intersection is immediately lost (generically $s = 0$). The QSVD is clearly not able to point out an "approximate intersection", but becomes trivial instead. A one stage QSVD procedure thus turns out to be useless for these types of problems. Alternatively, one can make use of a two stage QSVD algorithm, like in [Paige & Saunders 1981]. In a first step, the SVD of $\{A^T, B^T\}^T$ is computed, and in a second step, the QSVD follows from the CS-decomposition of certain submatrices computed in the first step –see [Paige & Saunders 1981] for details. The difference between these methods turns out to be the intermediate rank decision after the first SVD in the latter approach, that e.g. fixes the dimension of the approximate intersection, by implicitly setting the small singular values of $\{A^T, B^T\}$ equal to zero. Furthermore, once this rank decision can be made in a reliable manner, e.g. the generalized singular values are well conditioned, as it was shown in [Sun 1983a, Sun 1983b, Paige 1984]. Although this –possibly difficult– intermediate rank decision has been a main motive for developing a one stage QSVD algorithm [Paige 1986], in practice it is somehow inevitable.

Finally, let us mention that in a two stage QSVD algorithm, the first step –which is an ordinary SVD - can be computed adaptively, by making use of SVD updating, see Chapter 2.

4.3 PSVD Updating

Similar to the above development for the QSVD, one can consider the updating of the PSVD and devise an adaptive algorithm accordingly. The Product Singular Value Decomposition (PSVD) is a generalized singular value decomposition, at the same level of generality as the QSVD [Fernando & Hammarling 1987, De Moor & Zha 1990]. It is a simultaneous reduction of two matrices A and B having the same number of columns, by two orthogonal matrices U_A and U_B and a non-singular matrix X.

Theorem 2 *Product SVD [Fernando & Hammarling 1987, De Moor 1989]*

Every pair of matrices $A_{m \times n}$ and $B_{p \times n}$, can be factored as follows

$$\begin{aligned} \underbrace{A}_{m \times n} &= \underbrace{U_A}_{m \times m} \cdot \underbrace{\Sigma_A}_{m \times n} \cdot \underbrace{X^T}_{n \times n} \\ \underbrace{B}_{p \times n} &= \underbrace{U_B}_{p \times p} \cdot \underbrace{\Sigma_B}_{p \times n} \cdot \underbrace{X^{-1}}_{n \times n} \end{aligned}$$

where U_A and U_B are orthogonal, and Σ_A and Σ_B have the following structure

$$\Sigma_A = \begin{array}{c} \\ \rho_1 \\ \rho_a - \rho_1 \\ m - \rho_a \end{array} \begin{array}{c} \begin{array}{cccc} \rho_1 & \rho_a - \rho_1 & \rho_b - \rho_1 & n - \rho_a - \rho_b + \rho_1 \end{array} \\ \begin{pmatrix} \Sigma_1^{1/2} & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$\Sigma_B = \begin{array}{c} \\ \rho_1 \\ \rho_b - \rho_1 \\ p - \rho_b \end{array} \begin{array}{c} \begin{array}{cccc} \rho_1 & \rho_a - \rho_1 & \rho_b - \rho_1 & n - \rho_a - \rho_b + \rho_1 \end{array} \\ \begin{pmatrix} \Sigma_1^{1/2} & 0 & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$\begin{aligned} \rho_a &= \text{rank}\{A\} \\ \rho_b &= \text{rank}\{B\} \\ \rho_1 &= \text{rank}\{A \cdot B^T\} \end{aligned}$$

and Σ_1 is square diagonal with positive diagonal elements equal to the non-zero singular values of $A \cdot B^T$.

The PSVD obviously reveals the SVD of the product $A \cdot B^T$, hence the name. The PSVD appears in a number of problems in systems theory and signal processing, e.g. via the orthogonal Procrustes problems in harmonic retrieval and direction finding [Arun & Rao 1988, Arun 1989, Zoltowski & Stavrinides 1989], in computing balancing transformations [Laub et al. 1987], in computing the Kalman decomposition [Fernando 1987], etc.

As further on we consider the updating problem when new rows are appended sequentially, we can assume that the number of rows in A and B is larger than the number of columns. For that case, one can verify that an equivalent reduction is given by

$$\begin{aligned} \underbrace{A}_{m\times n} &= \underbrace{U_A}_{m\times n} \cdot \underbrace{\Sigma_A}_{n\times n} \cdot \underbrace{X^T}_{n\times n} \\ \underbrace{B}_{p\times n} &= \underbrace{U_B}_{p\times n} \cdot \underbrace{\Sigma_B}_{n\times n} \cdot \underbrace{X^{-1}}_{n\times n} \end{aligned}$$

$$\Sigma_A = \begin{array}{c} \\ \rho_1 \\ \rho_a - \rho_1 \\ \rho_b - \rho_1 \end{array} \begin{array}{c} \begin{array}{cccc} \rho_1 & \rho_a - \rho_1 & \rho_b - \rho_1 & \end{array} \\ \begin{pmatrix} \Sigma_1^{1/2} & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} = diag\{\alpha_1, \ldots, \alpha_n\}$$

$$\Sigma_B = \begin{array}{c} \\ \rho_1 \\ \rho_a - \rho_1 \\ \rho_b - \rho_1 \end{array} \begin{array}{c} \begin{array}{cccc} \rho_1 & \rho_a - \rho_1 & \rho_b - \rho_1 & \end{array} \\ \begin{pmatrix} \Sigma_1^{1/2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} = diag\{\beta_1, \ldots, \beta_n\}$$

where U_A and U_B have orthonormal columns.
The pairs $\{\alpha_i, \beta_i\}$ are called the product singular value pairs of $\{A, B\}$, while $\alpha_i \cdot \beta_i$ is a product singular value. In view of numerical stability, a more desirable form for the PSVD can be obtained as follows –similar to the QSVD case. By making use of a factorization of X,

$$X = Q \cdot R^T$$

where Q is orthogonal and R upper triangular, we obtain

$$\begin{aligned} A &= U_A \cdot \Sigma_A \cdot R \cdot Q^t \\ B &= U_B \cdot \Sigma_B \cdot R^{-t} \cdot Q^t \end{aligned}$$

where A and B are reduced by orthogonal transformations to triangular matrices –upper resp. lower– with orthogonal rows, in a sense that

$$\{\Sigma_A \cdot R\} \cdot \{\Sigma_B \cdot R^{-t}\}^T = \Sigma_A \cdot \Sigma_B.$$

Analogous to the QSVD updating, we can consider the following updating problem

$$\begin{aligned} \underline{A} = \begin{bmatrix} \lambda \cdot A \\ a^T \end{bmatrix} &= \underbrace{\underline{U_A}}_{(m+1)\times n} \cdot \underline{\Sigma_A} \cdot \underline{R} \cdot \underline{Q}^T \\ \underline{B} = \begin{bmatrix} \lambda \cdot B \\ b^T \end{bmatrix} &= \underbrace{\underline{U_B}}_{(p+1)\times n} \cdot \underline{\Sigma_B} \cdot \underline{R}^{-t} \cdot \underline{Q}^T. \end{aligned}$$

We consider a continuous updating after each sampling, where only Q, R (and/or R^{-t}) and Σ_A, Σ_B are of interest.

Similar to the QSVD case, we develop an updating algorithm by combining QR and QL-updating (!) with a PSVD algorithm [Heath et al. 1986].

A Jacobi-type PSVD algorithm

The following Jacobi-type PSVD algorithm is adapted from [Heath et al. 1986]. Again, preliminary decompositions of A and B are performed

$$A = Q_A \cdot \begin{bmatrix} R_A \\ 0 \end{bmatrix}$$

$$B = Q_B \cdot \begin{bmatrix} L_B \\ 0 \end{bmatrix}$$

where R_A is upper triangular, while L_B is lower triangular this time [17]. The PSVD can then obviously be computed from the PSVD of R_A and L_B

$$A = Q_A \cdot \begin{bmatrix} \Theta_A \cdot \Sigma_A \cdot R \cdot Q^T \\ 0 \end{bmatrix} = \underbrace{Q_A \cdot \begin{bmatrix} \Theta_A \\ 0 \end{bmatrix}}_{U_A} \cdot \Sigma_A \cdot R \cdot Q^T$$

[17] Computing a QL decomposition is similar to computing a QR decomposition.

$$B = Q_B \cdot \begin{bmatrix} \Theta_B \cdot \Sigma_B \cdot R^{-t} \cdot Q^T \\ 0 \end{bmatrix} = \underbrace{Q_B \cdot \begin{bmatrix} \Theta_b \\ 0 \end{bmatrix}}_{U_B} \cdot \Sigma_B \cdot R^{-t} \cdot Q^T.$$

An iterative procedure is then applied, that implicitly reduces $R_A \cdot L_B^T$ to a diagonal matrix. A sequence of matrices $R_{A_{[k-1]}}, R_{B_{[k-1]}}$ is generated via the recurrence

$$\begin{aligned} R_{A_{[0]}} &\Leftarrow R_A \\ L_{B_{[0]}} &\Leftarrow L_B \end{aligned}$$

for $k = 1, \ldots, k_{\max}$

$$\begin{aligned} R_{A_{[k-1]}} &\Leftarrow \Theta_{A_{[k]}}^T \cdot R_{A_{[k-1]}}^T \cdot Q_{[k]} \\ L_{B_{[k-1]}} &\Leftarrow \Theta_{B_{[k]}}^T \cdot L_{B_{[k-1]}}^T \cdot Q_{[k]} \end{aligned}$$

end

with $\Theta_{A_{[k-1]}}, \Theta_{B_{[k-1]}}$ and $Q_{[k]}$ orthogonal transformations to be defined next. After a certain number of iterations $k_{\max}$ the triangular factors have orthogonal rows –up to the relative precision ϵ–

$$\begin{aligned} \Sigma_A \cdot R &\overset{\epsilon}{\simeq} R_{A_{[k_{\max}]}} \\ \Sigma_B \cdot R^{-T} &\overset{\epsilon}{\simeq} L_{B_{[k_{\max}]}}. \end{aligned}$$

The orthogonal transformations are accumulated to give

$$\begin{aligned} Q &\simeq Q_{[1]} \ldots Q_{[k_{\max}]} \\ \Theta_A &\simeq \Theta_{A_{[1]}} \ldots \Theta_{A_{[k_{\max}]}} \\ \Theta_B &\simeq \Theta_{B_{[1]}} \ldots \Theta_{B_{[k_{\max}]}}. \end{aligned}$$

As both R_A and L_B^T are upper triangular, one has

$$\{R_A \cdot L_B^T\}_{i,i+1} = \{R_A\}_{i,i+1} \cdot \{L_B\}_{i,i+1}^T.$$

This equation allows for the implicit application of the parallel SVD algorithm to the upper triangular matrix $R_A \cdot L_B^T$. Similar to he QSVD case, the rotations $\Theta_{A_{[k-1]}}$ and $\Theta_{B_{[k-1]}}$ that constitute Θ_A and Θ_B

respectively, can then be computed from a 2×2 SVD

$$\begin{bmatrix} s_{i,i} & 0 \\ 0 & s_{i+1,i+1} \end{bmatrix} \Leftarrow \underbrace{\begin{bmatrix} \cos\theta_A & \sin\theta_A \\ -\sin\theta_A & \cos\theta_A \end{bmatrix}}_{\{\Theta_{A_{[k]}}\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} r_{i,i}^A & r_{i,i+1}^A \\ 0 & r_{i+1,i+1}^A \end{bmatrix}}_{\{R_{A_{[k-1]}}\}_{i,i+1}} \ldots$$

$$\ldots \underbrace{\begin{bmatrix} l_{i,i}^B & l_{i,i+1}^B \\ 0 & l_{i+1,i+1}^B \end{bmatrix}}_{\{L_{B_{[k-1]}}\}_{i,i+1}^T} \cdot \underbrace{\begin{bmatrix} \cos\theta_B & \sin\theta_B \\ -\sin\theta_B & \cos\theta_B \end{bmatrix}}_{\{\Theta_{B_{[k]}}\}_{i,i+1}}$$

where the elements $s_{i,i}$ and $s_{i+1,i+1}$ need not be computed explicitly. From

$$\{\Theta_{A_{[k]}}^T \cdot R_{A_{[k-1]}}\}_{i,i+1} \cdot \{\Theta_{B_{[k]}}^T \cdot L_{B_{[k-1]}}\}_{i,i+1}^T = diag\{s_{i,i}, s_{i+1,i+1}\}$$

it then follows that one rotation can triangularize both 2×2 matrices at the same time

$$\begin{bmatrix} \tilde{r}_{i,i}^A & \tilde{r}_{i,i+1}^A \\ 0 & \tilde{r}_{i+1,i+1}^A \end{bmatrix} = \{\Theta_{A_{[k]}}^T \cdot R_{A_{[k-1]}} \cdot Q_{[k]}\}_{i,i+1}$$

$$\begin{bmatrix} \tilde{l}_{i,i}^B & \tilde{l}_{i,i+1}^B \\ 0 & \tilde{l}_{i+1,i+1}^B \end{bmatrix}^T = \{\Theta_{B_{[k]}}^T \cdot L_{B_{[k-1]}} \cdot Q_{[k]}\}_{i,i+1}.$$

$R_{A_{[k]}}$ and $L_{B_{[k]}}$ thus remain upper, resp. lower triangular at any stage, while $R_{A_{[k]}} \cdot L_{B_{[k]}}^T$ is being reduced to a diagonal matrix –without explicitly being formed. The rest is similar to the QSVD case, and left out for the sake of brevity.

PSVD updating

Again, a PSVD updating algorithm can straightforwardly be devised similar to the QSVD updating algorithm. The algorithm is summarized as follows (with $n-1$ rotations after each update). Note that QL-updating is used instead of QR updating for the lower triangular L_B-factor [18].

[18] Updating a QLD is similar to updating a QRD.

Initialization

$$\begin{aligned} Q &\Leftarrow I_{n\times n} \\ R_A &\Leftarrow O_{n\times n} \\ L_B &\Leftarrow O_{n\times n} \end{aligned}$$

Loop

for $k = 1, \ldots, \infty$

input new measurement vectors $a_{[k]}, b_{[k]}$

1. matrix-vector multiplication

$$\tilde{a}_{[k]} = a_{[k]}^T \cdot Q$$

$$\tilde{b}_{[k]} = b_{[k]}^T \cdot Q$$

2. QR and/or QL-updatings

$$\begin{bmatrix} R_A \\ 0 \end{bmatrix} \Leftarrow Q_{A_{[k]}}^T \cdot \begin{bmatrix} \lambda \cdot R_A \\ \tilde{a}_{[k]}^T \end{bmatrix}$$

$$\begin{bmatrix} L_B \\ o \end{bmatrix} \Leftarrow Q_{B_{[k]}}^T \cdot \begin{bmatrix} \lambda \cdot L_B \\ \tilde{b}_{[k]}^T \end{bmatrix}$$

3. PSVD steps

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

$$\begin{aligned} R_A &\Leftarrow \Theta_{A_{[k,i]}}^T \cdot R_A \cdot Q_{[k,i]} \\ L_B &\Leftarrow \Theta_{B_{[k,i]}}^T \cdot L_B \cdot Q_{[k,i]} \\ Q &\Leftarrow Q \cdot Q_{[k,i]} \end{aligned}$$

end

end

Systolic PSVD updating array

A PSVD updating array performing the above algorithm, is somewhat more involved, compared to the corresponding QSVD array, due to

the fact that L_B is lower triangular instead of upper triangular. An easy solution consists in providing two triangular arrays, containing R_A, resp. L_B, and two square arrays, both containing the orthogonal Q-matrix, as depicted in **Figure 4.8**. *The topmost array performs matrix-vector multiplications $\tilde{a}_{[k]}^T = a_{[k]}^T \cdot Q_{[k]}$, while the lowermost performs the products $\tilde{b}_{[k]}^T = b_{[k]}^T \cdot Q_{[k]}$. The triangular arrays perform the PSVD steps, interlaced with QR and QL-updatings. Note that in the upper triangular factor R_A, the QR updates run from the upper-left corner to the lower-right corner. The odd-even PSVD waves therefore have to be interpreted as pipelined sequences of $n-1$ rotations, starting from the upper-left corner and running downwards along the diagonal. On the other hand, in the lower triangular factor L_B, the QL-updates run from the lower-right corner to the upper-left corner. The odd-even PSVD waves then have to be interpreted as pipelined sequences of $n-1$ rotations, starting from the bottom and running upwards along the diagonal. As a result of this, the implemented* **parallel algorithm** *differs from the* **sequential algorithm** *in that the computed transformations Θ_A, etc. do not correspond to e.g. a specific $\Theta_{A_{[k,i]}}$ –so we should use notations like $\Theta_{A_{[k]}}$. In principle however, nothing has changed : The triangular factors are iteratively being corrected such that $R_A \cdot L_B^T$ is always close to a diagonal matrix, while at the same time new rows are worked in, both in R_A and in L_B.*

Finally, without disturbing the array operations, it is again possible to generate output vectors at regular time intervals (columns of X and/or X^{-t}, corresponding to the smallest product singular values). For instance, columns of $X = Q \cdot R^T \simeq Q \cdot R_A^T$ [19] *(up to a scale factor) can be obtained as follows. As in the QSVD case, a vector t can be generated on the main diagonal, making use of the same threshold function as in the previous section, where now*

$$\tau_{ii} = \tau_{ii}^A \cdot l_{ii}^B$$

and γ satisfies

$$\alpha_{n-1} \cdot \beta_{n-1} > \gamma > \alpha_n \cdot \beta_n$$

and is assumed to be known a priori. The output vector can then be singled out, by performing the matrix-vector product $x = Q \cdot R_A^T \cdot t$, see

[19] $Q_{[k]} \cdot R_{A_{[k]}}^T$ to be more "specific".

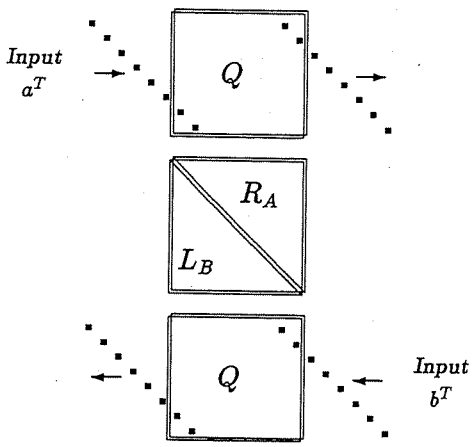


Figure 4.8: *PSVD updating array*

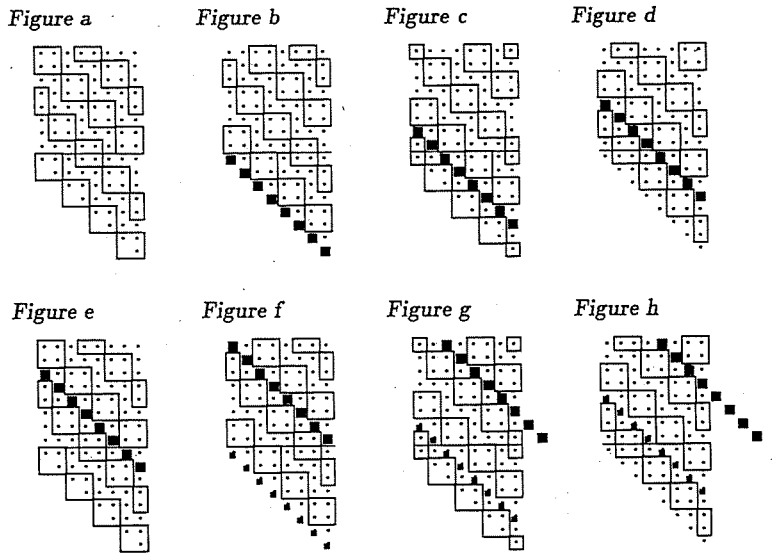


Figure 4.9: *Output vector generation*

Figure 4.9. *The t-vector is generated in Figure 4.9a, and propagated to the right in order to perform $t^* = R_A^T \cdot t$. Intermediate results are passed on upwards. The resulting t^* vector then enters the topmost square array in a skewed fashion, as indicated, where the matrix-vector product $Q \cdot t^*$ is performed, with t^* being propagated further on upwards and intermediate results being propagated to the right. The output vector becomes available at the right-hand side of the array. Cell descriptions can be obtained straightforwardly and are omitted for the sake of brevity. Similarly, column vectors of $X^{-t} = Q \cdot R^{-1} \simeq Q \cdot L_B^T$ can be generated, making use of L_B and the lowermost square array. The output vectors then become available at the right-hand side of the array, Figure 4.8.*

With respect to numerical reliability, it should be noted that duplicating the Q-matrix into a second square array, will not introduce any additional source of error build-up. Although the pipelining is different –which seems to rule out the possibility of devising a PSVD updating array with one single square part– both square arrays perform exactly the same operations, introducing exactly the same round-off errors. Therefore the stored matrices in these arrays remain in perfect mutual agreement.

Square root free PSVD updating

Briefly, a square root free procedure can be obtained as follows. The triangular factors are stored in factored form with the same column scalings:

$$R_A = D^{\frac{1}{2}}_{row_A} \cdot \bar{R}_A \cdot D^{\frac{1}{2}}_{col}$$

$$L_B = D^{\frac{1}{2}}_{row_B} \cdot \bar{L}_B \cdot D^{\frac{1}{2}}_{col}.$$

The orthogonal matrix Q is factored accordingly:

$$Q = \bar{Q} \cdot D^{\frac{1}{2}}_{col}.$$

The only real difference concerns the 2×2 PSVD's on the main diagonal. As was explained earlier, in a first step an SVD of $\{R_A\}_{i,i+1} \cdot \{L_B\}^T_{i,i+1}$ is computed, while the second step consists in performing one single Givens rotation. As for the SVD, one immediately sees that

$$\{R_A\}_{i,i+1} \cdot \{L_B\}^T_{i,i+1} = \{D^{\frac{1}{2}}_{row_A}\}_{i,i+1} \cdot \{\bar{R}_A\}_{i,i+1} \cdot \underbrace{\{D^{\frac{1}{2}}_{col}\}_{i,i+1} \cdot \{D^{\frac{1}{2}}_{col}\}_{i,i+1}}_{\{D_{col}\}_{i,i+1}} \cdot \{\bar{L}_B\}^T_{i,i+1} \cdot \{D^{\frac{1}{2}}_{row_B}\}_{i,i+1}$$

–where D_{col} is available– boils down to an SVD with a 2-sided factorization, which can be computed approximately without square roots, as was detailed in Chapter 3.

Similar to the QSVD case, one can prove that even if an approximate SVD is performed –with an upper triangular result- one single Givens rotation $Q_{[k]}$ can upper resp. lower triangularize both matrices at the same time. For this Givens transformation, it again suffices to simply apply Gentleman's procedure.

Note on the dual PSVD updating problem

Similar to the QSVD case, the dual PSVD updating problem –where new column are appended in each time step– turns out to be much more difficult, but of little use in practice. Let us only mention that e.g. due to noise on the data, the Σ_1-blocks in the PSVD definition, will immediataly have the highest possible dimension $\max(p, m)$, as $A \cdot B^T$ will generically be of full row or column rank, depending on which one is smaller, p or m. In other words, the PSVD then does not contain any more essential information than the SVD of the product $A \cdot B^T$.

4.4 Conclusions

The SVD row updating technique has been extended to generalized SVD updating problems, viz. QSVD and PSVD updating when new rows are appended to either one or both of the matrices involved. All of these algorithms deliver approximate decompositions after each time step, with an operation count which is always $O\{n^2\}$ –similar to the SVD case. The algorithms can be stabilized by including appropriate re-orthogonalization procedures, see Chapter 3.

Systolic arrays have been devised as well. A perfect pipelining is obtained at the cost of additional operations, and sometimes –for PSVD row updating for instance– by re-ordering the transformations. The implemented parallel algorithm then slightly differs from the corresponding sequential algorithm. For slowly time-vaying systems, the difference is irrelevant.

Chapter 5

Jacobi-type algorithms for signal processing, systems and control

In the previous chapters, we have derived several updating algorithms, where each time a Jacobi-type algorithm is combined with an updating strategy (QR/QL updating) and a few other key matrix operations (matrix-vector multiplications, triangular backsolves). It appears that with these basic ingredients one can cook up a whole set of extremely useful algorithms for various related problems. Basically these algorithms always consist of one or more Jacobi-type algorithms[1]*. In general, elementary Jacobi-type algorithms are pieced together to give* **combined Jacobi-type algorithms.**

In this chapter, we first derive a Jacobi-type algorithm for inverting triangular matrices. As such this algorithm is rather inefficient and thus nothing to rave about. However it can be combined with QR updating to result in an elegant iterative scheme for solving **Riccati and Lyapunov equations,** *which is then readily implemented on a systolic array.*

A similar inversion algorithm can be incorporated in **recursive least squares schemes** *that exploit* **inverse factorization.** *The inversion algorithm then continuously annihilates accumulated errors and thus stabilizes the overall QR updating scheme. The resulting systolic array is very similar to the SVD updating array.*

A second –and related– systolic array for RLS is derived, based on the **orthogonal-inverse updating scheme** *of [Pan & Plemmons 1989]. This systolic implementation shows the same Jacobi-type partitioning, but it is over five times more efficient then the first array. As far as we know it is the most (only) efficient systolic implementation for RLS.*

Finally, we also consider the **Kalman filtering** *problem. It is well known that square root information filtering procedures can be recast as structured QR updating problems. The trick is then to split up this QR updating problem into a smaller QR updating problem plus a QR factorization, and use a Jacobi-type algorithm for the latter. Once again, the basic ingredients are there and an efficient systolic implementations is straightforwardly derived.*

Section 5.1 deals with solving Riccati and Lyapunov equations. Recursive least squares schemes with inverse factorization are dealt with in section 5.2 and section 5.3. Finally the Kalman filtering problem is addressed in section 5.4.

[1] Note that the QR updating algorithm can be viewed as a Jacobi-type algorithm as well

5.1 Riccati and Lyapunov equations

Riccati and Lyapunov equations arise in a number of problems in control and filtering, and their numerical solution has been a topic of considerable interest, see e.g [Bartels & Stewart 1972, Byers 1987, Gardiner & Laub 1986, Golub et al. 1979, Hammarling 1982, Laub 1979, Van Dooren 1981]. The recursive matrix sign function algorithm is known to provide a suitable iterative method for solving these equations. The primary appeal of this approach is its simplicity. In a recent paper a factored version of this algorithm is given, which is particularly amenable to parallel implementation [Charlier & Van Dooren 1989b]. The basic operations involved are triangular matrix inversions and triangular updates. It is furthermore shown how these can be pipelined on a triangular processor array.

We first briefly recall this factored algorithm for the matrix sign recursions, wherewith we closely follow the development in [Charlier & Van Dooren 1989b]. Next a Jacobi-type algorithm is described for inverting triangular matrices, which is then combined with the (re-organized) triangular updates to give a Jacobi-type algorithm for the matrix sign recursions. The resulting algorithm is finally implemented on a systolic array, which is roughly three times more efficient than the array in [Charlier & Van Dooren 1989b].

Factored matrix sign recursions

Let A, B, V and W be given $\frac{n}{2} \times \frac{n}{2}$ matrices, where V and W are

symmetric. Then the equation

$$XA + A^T X + XBW^{-1}B^T X + V = 0$$

is called the algebraic Riccati equation, which is of central interest in control theory. In the special case where $B = 0$, the (quadratic) Riccati equation reduces to the (linear) Lyapunov equation

$$XA + A^T X + V = 0.$$

A key technique for solving the above equations cosists in converting the problem to an eigenvalue problem involving an $n \times n$ Hamiltonian matrix

$$H = \begin{bmatrix} A & -BW^{-1}B^T \\ -V & -A^T \end{bmatrix}$$

which thus satisfies

$$J \cdot H^T \cdot J = H$$

for

$$J = \begin{bmatrix} 0 & I_{\frac{n}{2}} \\ -I_{\frac{n}{2}} & 0 \end{bmatrix}.$$

The stable subspace[2] $\mathcal{H}_{stable}$ *of the matrix H is known to yield the solution to the above equations as follows* [3]

$$X = X_2 \cdot X_1^{-1}$$

if X_1 is invertible and for any basis $\begin{bmatrix} X_1^T & X_2^T \end{bmatrix}^T$ such that

$$\mathcal{H}_{stable} = range\{\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}\}.$$

[2] spanned by eigenvectors corresponding to stable eigenvalues

[3] Standard assumptions in this respect are $V > 0$, $W > 0$, and that H has no purely imaginary eigenvalues.

The matrix sign function algorithm is a simple way to compute this subspace. Consider the following recursion

$$H_{k+1} = \frac{1}{2c_k} \cdot (H_k + c_k^2 H_k^{-1}) \qquad c_k = |det(H_k)|^{\frac{1}{n}}$$

with $H_0 = H$. This recursion is known to converge to the matrix sign function of H

$$H_\infty = sign\{H\}$$

which has the same eigenvectors as H. The corresponding eigenvalues are ± 1 and satisfy

$$\lambda_i\{H_\infty\} = sign\{\lambda_i\{H\}\}.$$

The requested stable subspace $\mathcal{H}_{stable}$ then coincides with the null space of $H_\infty + I_n$. Let

$$H_\infty + I_n = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}$$

then it is shown that

$$X = -X_{12}^{-1} \cdot X_{11} = -X_{22}^{-1} \cdot X_{21}$$

when the relevant matrices are invertible.

The main computational work is thus in the matrix sign recursions, for which efficient algorithms have been designed. The systolic algorithm in [Charlier & Van Dooren 1989b] proceeds as follows. First of all, it is readily seen that the matrices H_k are Hamiltonian throughout, i.e. $JH_k^T J = H_k$. From this it follows that the matrices $Z_k \stackrel{def}{=} JH_k$ are symmetric, a property much easier to exploit algorithmically. The recursions in H_k are then worked into recursions in Z_k as follows

$$\begin{aligned} JH_{k+1} &= \frac{1}{2c_k} \cdot (JH_k + c_k^2 \underbrace{JH_k^{-1}}_{J(JH_k)^{-1}J}) \\ Z_{k+1} &= \frac{1}{2c_k} \cdot (Z_k + c_k^2 J Z_k^{-1} J) \qquad c_k = |det(Z_k)|^{\frac{1}{n}}. \end{aligned}$$

Next, use is made of two symmetric factorizations [4]

$$\begin{aligned} Z_k &= L_k \cdot \Sigma \cdot L_k^T \\ J \cdot Z_k \cdot J &= U_k \cdot \Sigma \cdot U_k^T \end{aligned}$$

where U_k and L_k are upper, resp. lower triangular matrices and where

$$\Sigma = \begin{bmatrix} -I_{\frac{n}{2}} & 0 \\ 0 & I_{\frac{n}{2}} \end{bmatrix}.$$

This then leads to coupled recursions

$$L_{k+1} \cdot \Sigma \cdot L_{k+1}^T = \frac{1}{2c_k} \begin{bmatrix} L_k & c_k U_k^{-T} \end{bmatrix} \cdot \begin{bmatrix} \Sigma & 0 \\ 0 & \Sigma \end{bmatrix} \cdot \begin{bmatrix} L_k^T \\ c_k U_k^{-1} \end{bmatrix}$$

$$U_{k+1} \cdot \Sigma \cdot U_{k+1}^T = \frac{1}{2c_k} \begin{bmatrix} U_k & c_k L_k^{-T} \end{bmatrix} \cdot \begin{bmatrix} \Sigma & 0 \\ 0 & \Sigma \end{bmatrix} \cdot \begin{bmatrix} U_k^T \\ c_k L_k^{-1} \end{bmatrix}.$$

Obtaining L_{k+1} and U_{k+1} from L_k and U_k then finally amounts to determining diag$\{\Sigma, \Sigma\}$-unitary matrices Q_L and Q_U such that

$$\begin{bmatrix} L_{k+1}^T \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2c_k}} Q_L \cdot \begin{bmatrix} L_k^T \\ c_k U_k^{-1} \end{bmatrix}$$

$$\begin{bmatrix} U_{k+1}^T \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2c_k}} Q_U \cdot \begin{bmatrix} U_k^T \\ c_k L_k^{-1} \end{bmatrix}.$$

The only operations involved are thus seen to be triangular inversions and triangular updates [5]. *In [Charlier & Van Dooren 1989b], it is shown how the inversions and the updates can be pipelined on a triangular processor array. Note that in the formula for L_{k+1}^T, only upper triangular matrices are involved, while in the formula for U_{k+1}^T, only lower triangular matrices are involved. Below we derive updating formulas that exclusively work with upper triangular matrices. This turns out to be a main advantage in view of efficient parallel implementation. Furthermore, Jacobi-type algorithms are used for each of the two steps –inversion and triangularization– which can then elegantly be combined into a systolic implementation that is far more efficient.*

[4] the existence of these is proved in [Charlier & Van Dooren 1989b].

[5] where diag$\{\Sigma, \Sigma\}$-unitary instead of unitary matrices are used, which however basically amounts to the same operation

A Jacobi-type algorithm for inverting triangular matrices

First of all let us focus on the required matrix inversions L_k^{-1} and U_k^{-1}. Our aim is to derive a Jacobi-type algorithm for these, based on locally computed plane transformations.

More generally, let R be a given $n \times n$ upper triangular matrix. Suppose that this matrix can be reduced to the identity by applying a sequence of row transformations T_l, where $l = 1, \ldots, \frac{n(n-1)}{2}$, together with column permutations Π_l [6]

$$\underbrace{T_{\frac{n(n-1)}{2}} \ldots T_1}_{T} \cdot R \cdot \underbrace{\Pi_1 \ldots \Pi_{\frac{n(n-1)}{2}}}_{\Pi} = I.$$

Hence

$$T = \Pi^{-1} \cdot R^{-1}$$

If we now at the same time apply a similar sequence of transformations onto the identity [7] *we obtain*

$$\underbrace{\Pi_{\frac{n(n-1)}{2}}^T \ldots \Pi_1^T}_{\Pi^T} \cdot I \cdot \underbrace{T_1^T \ldots T_{\frac{n(n-1)}{2}}^T}_{T^T} = \Pi^T \cdot R^{-T} \cdot \Pi^{-T}.$$

The sequence of permutations will be such that

$$\Pi = \begin{bmatrix} 0 & & & 1 \\ & & 1 & \\ & \cdot & & \\ & 1 & & \\ 1 & & & 0 \end{bmatrix}$$

with $\Pi^T = \Pi^{-1} = \Pi$. Hence, the obtained matrix equals $\Pi R^{-T} \Pi$, from which R^{-1} can readily be recovered. It will turn out that for the matrix sign recursions, $\Pi R^{-T} \Pi$ is even more useful than R^{-1} itself. Finally, note that both $\Pi R^{-T} \Pi$ and R^{-1} are upper triangular.

[6] alternatively one could also apply column transformations together with row permutations.

[7] this is an old trick

It thus remains to define the transformation matrices. First, Π_l is a permutation in the $\{i, i+1\}$ plane, for a certain pivot index i. In other words, Π_l differs from the identity only in

$$\{\Pi_l\}_{i,i+1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Next, T_l is an embedding of a plane transformation, which differs from the identity only in the entries

$$\{T_l\}_{i,i+1} = \{\Pi_l\}_{i,i+1} \cdot \{R_{l-1}\}^{-1}_{i,i+1}$$

where

$$R_{l-1} \stackrel{def}{=} T_{l-1} \ldots T_1 \cdot R \cdot \Pi_1 \ldots \Pi_{l-1}.$$

From this it follows that

$$\begin{aligned} \{R_l\}_{i,i+1} &= \{T_l\}_{i,i+1} \cdot \{R_{l-1}\}_{i,i+1} \cdot \{\Pi_l\}_{i,i+1} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \end{aligned}$$

In other words, each iteration is seen to set a 2×2 block on the diagonal equal to I_2. It is readily seen that the complete R-matrix is reduced to the identity if we let the pivot index i successively take up the following values (forward sweep)

$$\begin{array}{lll} i & = & 1 \;\; 2 \;\; \ldots \;\; n-2 \;\; n-1 \\ & & 1 \;\; 2 \;\; \ldots \;\; n-2 \\ & & \vdots \\ & & 1. \end{array}$$

Accumulating the permutations Π_l is then seen to give the previously defined Π-matrix. We give a complete algorithmic description of this, followed by a small example to make things clear.

Algorithm triangular inversion :

Initialization

$$\begin{aligned} R &\leftarrow R_0 \\ X &\leftarrow I_{n \times n} \end{aligned}$$

Loop
  for $j = 1, \ldots, n-1$
    for $i = 1, \ldots, n-j$
      compute transformations

$$\{\Pi_{i_j}\}_{i,i+1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\{T_{i_j}\}_{i,i+1} = \{\Pi_{i_j}\}_{i,i+1} \cdot \{R\}^{-1}_{i,i+1}$$

      apply transformations

$$\begin{aligned} R &\leftarrow T_{i_j} \cdot R \cdot \Pi_{i_j} \\ X &\leftarrow \Pi_{i_j} \cdot X \cdot T^T_{i_j} \end{aligned}$$

    end
  end

After the algorithm has terminated, X equals $\Pi R_0^{-T} \Pi$.

Example :

	R				X			
	−1.15	−1.34	−.38	−1.27	1.00	0	0	0
	0	.37	.01	.40	0	1.00	0	0
	0	0	.42	.22	0	0	1.00	0
	0	0	0	−.19	0	0	0	1.00
	1.00	0	.04	1.08	2.73	−3.19	0	0
1	0	1.00	.29	−.15	0	−.87	0	0
	0	0	.42	.22	0	0	1.00	0
	0	0	0	−.19	0	0	0	1.00
	1.00	.04	0	1.08	2.73	0	−3.19	0
2	0	1.00	0	.53	0	2.38	−.68	0
	0	0	1.00	−.30	0	0	−.87	0
	0	0	0	−.19	0	0	0	1.00

3	1.00	.04	1.08	0	2.73	0	0	−3.19
	0	1.00	.53	0	0	2.38	0	−.68
	0	0	1.00	0	0	0	−5.24	−1.59
	0	0	0	1.00	0	0	0	−.87
4	1.00	0	.53	0	2.38	−.08	0	−.68
	0	1.00	1.06	0	0	2.73	0	−3.19
	0	0	1.00	0	0	0	−5.24	−1.59
	0	0	0	1.00	0	0	0	−.87
5	1.00	.53	0	0	2.38	0	−.08	−.68
	0	1.00	0	0	0	−5.24	5.56	−1.59
	0	0	1.00	0	0	0	2.73	−3.19
	0	0	0	1.00	0	0	0	−.87
6	1.00	0	0	0	−5.24	2.77	−5.55	−1.59
	0	1.00	0	0	0	2.38	−.08	−.68
	0	0	1.00	0	0	0	2.73	−3.19
	0	0	0	1.00	0	0	0	−.87

From the above example, it is seen that R and X remain upper triangular throughout, and furthermore have non-trivial entries –i.e. not equal to 0 or 1– in complementary patterns. This can be exploited such that the resulting memory requirement is only $\frac{n^2}{2} + 0\{n\}$.

A systolic algorithm for the matrix sign recursions.

Let us now return to the matrix sign recursions. Instead of using U_k, we will store $\Pi U_k^T \Pi$ which is upper triangular as well. The updating formula for U_k is then worked into an updating formula for $\Pi U_k^T \Pi$ as follows

$$\underbrace{\begin{bmatrix} \Pi & 0 \\ 0 & \Pi \end{bmatrix} \begin{bmatrix} U_{k+1}^T \Pi \\ 0 \end{bmatrix}}_{\begin{bmatrix} \Pi U_{k+1}^T \Pi \\ 0 \end{bmatrix}}$$

$$= \frac{1}{\sqrt{2c_k}} \underbrace{\begin{bmatrix} \Pi & 0 \\ 0 & \Pi \end{bmatrix} \cdot Q_U \cdot \begin{bmatrix} \Pi & 0 \\ 0 & \Pi \end{bmatrix}}_{\bar{Q}_U} \begin{bmatrix} \Pi & 0 \\ 0 & \Pi \end{bmatrix} \begin{bmatrix} U_k^T \Pi \\ c_k L_k^{-1} \Pi \end{bmatrix}$$

where $\bar{Q}_U$ is readily shown to be $\mathrm{diag}\{\Sigma, \Sigma\}$-unitary as well. This results in coupled recursions for L_k^T and $\Pi U_k^T \Pi$

$$\begin{bmatrix} L_{k+1}^T \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2c_k}} Q_L \cdot \begin{bmatrix} L_k^T \\ c_k U_k^{-1} \end{bmatrix}$$

$$\begin{bmatrix} \Pi U_{k+1}^T \Pi \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2c_k}} \bar{Q}_U \cdot \begin{bmatrix} \Pi U_k^T \Pi \\ c_k \Pi L_k^{-1} \Pi \end{bmatrix}.$$

Note that in these updating formulas, all the matrices are upper triangular, unlike in the original formulas.

An additional advantage of these revised upating formulas is the following. Suppose that L_k^T and $\Pi U_k^T \Pi$ are stored, which are both upper triangular. If we apply the inversion algorithm onto these, we obtain $\Pi L_k^{-1} \Pi$ and $\Pi\Pi U_k^{-1} \Pi\Pi = U_k^{-1}$ –both upper triangular as well– which can directly be used for the subsequent triangular updates. The coupling in the updating formulas is thus seen to be perfectly symmetric.

As for a systolic implementation, *it remains to show how the Jacobi-type inversions can be pipelined with the subsequent triangular updates. Obviously, we need a* **triangular array** *to accomodate the triangular matrices. For the inversion algorithm, the triangular array contains both R and X. The matrix sign recursions require two inversions at the same time, so that we will have $R^{(1)} = L_k^T$, $R^{(2)} = \Pi U_k^T \Pi$, and likewise $X^{(1)}$ and $X^{(2)}$. Furthermore, a version of L_k^T and $\Pi U_k^T \Pi$ should be stored for the subsequent update. The total memory requirement is thus $4 \cdot \frac{n^2}{2} + O\{n\}$, the same as in [Charlier & Van Dooren 1989b]* [8].

The matrix inversions are carried out in a forward sweep as was demonstrated earlier. The subsequent backward sweep is used for the triangular updates. A QR update with a "full" row is known to correspond to a sequence of transformations, where the pivot index takes up the values 1 through n, see Figure 3.7. In the specific case of the matrix sign recursions, it is seen that an upper triangular matrix (either L_k^T

[8] remember that R and I can be combined into one single upper triangular matrix.

or $\Pi U_k^T \Pi$) is to be updated with another upper triangular matrix (U_k^{-1} or $\Pi L_k^{-1} \Pi$). Clearly, in order to zero the last row in the latter matrix, only one transformation (for $i = n$) is required. Zeroing the one but last row requires two transformations ($i = n-1, n$), etc. One complete update then corresponds to a sequence of transformations as follows (backward sweep)

$$\begin{array}{cc} i = & \\ & n \\ & n-1 \quad n \\ & \vdots \\ & 1 \quad 2 \quad \ldots \quad n-2 \quad n-1 \quad n. \end{array}$$

Note once again that for the matrix sign recursions, two updates are carried out at the same time [9].

It is readily seen that the inversions (forward sweep) can be combined with the updates (backward sweep). After the first sequence of transformations in the inversion scheme ($i = 1, \ldots, n-1$), the last rows of U_k^{-1} and $\Pi L_k^{-1} \Pi$ are already available for the update ($i = n$), etc. As a reminder, the merger can be recast as follows, where the bars seperate between the inversions and the updates.

$$\begin{array}{cc} i = & 1 \quad 2 \quad \ldots \quad n-2 \quad n-1 \quad |n \\ & 1 \quad 2 \quad \ldots \quad n-2 \quad |n-1 \quad n \\ & \vdots \\ & 1 \quad |2 \quad \ldots \quad n-2 \quad n-1 \quad n \\ & |1 \quad 2 \quad \ldots \quad n-2 \quad n-1 \quad n. \end{array}$$

A complete iteration is thus seen to correspond to n sequences of transformations $i = 1, \ldots, n$. These can be pipelined on the processor array, as was explained earlier. Furthermore, each iteration as a whole can perfectly be pipelined with its predecessor and successor. In this way, there are no time lags, unlike in the array of [Charlier & Van Dooren 1989b]. As for the c_k-coefficients, these can be computed from the

[9] and either unitary or $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$-unitary transformations are used, corresponding to the ± 1 in $\begin{bmatrix} \Sigma & 0 \\ 0 & \Sigma \end{bmatrix}$, see [Charlier & Van Dooren 1989b] for details.

product of the diagonal entries in e.g. L_k^T, which in turn can be computed along with the first sequence of transformations. The resulting c_k should then be "backpropagated" in order to make it available in all the 2×2 blocks.

Finally let us compare this array to the one in [Charlier & Van Dooren 1989b]. The array in [Charlier & Van Dooren 1989b] has $\frac{n(n+1)}{2}$ processors in a triangular grid. The inversions are more or less pipelined with the updates, and one complete iteration takes roughly $3n$ time steps. The updates require more computations per time step than the inversions, and thus slow down the pipeline. One time step then corresponds to computing two transformations (cfr. two updates) of the type

$$\begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}.$$

The Jacobi-type array on the other hand, can be realized as a hexagonally connected processor grid with eight times less processors, as explained in Chapter 3. Each iteration takes n sequences of transformations $i = 1, \ldots, n$. With 4 time steps between two such sequences –see e.g Figure 3.7– the total time for one iteration is $4n$. Again, the updates require more computations per time step than the inversions [10]. *One time step is then seen to correspond to computing two transformations of the type*

$$\begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} a & c \\ b & d \end{bmatrix}.$$

In conclusion, one complete iteration in the Jacobi-type implementation takes $\frac{8}{3}$ of the computation time in the original implementation of [Charlier & Van Dooren 1989b]. As the number of processors is 8 times less, the overall efficiency is finally seen to be increased with a factor of roughly three.

[10] The inversions require row and column transformations, whereas the updates require only row transformations, but the transformations for the former are structured, i.e. with many zeroes involved.

5.2 Recursive Least Squares with stabilized inverse factorization

Recently developed recursive least squares schemes, where a triangular factor of both the covariance and the information matrix are stored and updated, are known to be particularly suited for parallel implementation. However, when finite precision arithmetic is used, round-off errors apparently accumulate unboundedly, so that after a number of updates the computed least squares solutions are useless. Below, we derive a Jacobi-type correction scheme –based on matrix inversion– which continuously annihilates accumulated errors and thus stabilizes the overall scheme. Finally it is shown how the resulting RLS-algorithm can be implemented on a systolic array.

RLS with inverse factorization

The **linear least squares (LS) problem** *is one of the oldest problems in applied mathematics, with numerous applications throughout modern engineering. In general, without reference to any specific application, the problem can be stated as follows* [11]*:*

> *Given a real $m \times n$ matrix A $(m \geq n)$ with full column rank n, and a real m-vector b, find x that solves*
>
> $$\min_x \|A \cdot x - b\|_2.$$

The solution to the above minimization problem is known to be

$$x_{LS} = (A^T \cdot A)^{-1} \cdot A^T \cdot b$$

[12] *which can be computed in an accurate and stable way from a QR decomposition, cfr. Chapter 1*

$$\underbrace{Q^T \cdot A}_{\begin{bmatrix} R \\ 0 \end{bmatrix}} \cdot x = \underbrace{Q^T \cdot b}_{\begin{bmatrix} y \\ z \end{bmatrix}}$$

with

$$R \cdot x_{LS} = y.$$

[11] only the full rank problem is considered here, see *Chapter 1*

[12] the matrix $A^T \cdot A$ is called *information matrix*, while its inverse is called *covariance matrix*.

In real-time signal processing applications both the data matrix A and the right-hand side b at time instant k are defined in a recursive manner

$$A_{[k]} = \begin{bmatrix} \lambda \cdot A_{[k-1]} \\ a^T_{[k]} \end{bmatrix} \qquad b_{[k]} = \begin{bmatrix} \lambda \cdot b_{[k-1]} \\ \beta_{[k]} \end{bmatrix}$$

where λ is a weighting factor. The **recursive least squares (RLS) problem** *then consists in re-computing the least squares solution after each sampling, by making use of the results of the previous time step. Briefly, the RLS problem is solved as follows*[13]*. Instead of $A_{[k-1]}$ and $b_{[k-1]}$, the corresponding triangular factor $R_{[k-1]}$ and right-hand side $y_{[k-1]}$ are stored. In a first step, these are updated with the new observation at time instant k (QR updating, cfr. Chapter 1)*

$$\underbrace{Q^T_{[k]} \cdot \begin{bmatrix} \lambda \cdot R_{[k-1]} \\ a^T_{[k]} \end{bmatrix}}_{\begin{bmatrix} R_{[k]} \\ 0 \end{bmatrix}} \cdot x = \underbrace{Q^T_{[k]} \cdot \begin{bmatrix} \lambda \cdot y_{[k-1]} \\ \beta^T_{[k]} \end{bmatrix}}_{\begin{bmatrix} y_{[k]} \\ \zeta_{[k]} \end{bmatrix}}.$$

The LS-solution at time step k can then be computed by solving the triangular system

$$R_{[k]} \cdot x_{LS_{[k]}} = y_{[k]}.$$

Each update in the above scheme takes $O\{n^2\}$ operations.

Although this RLS-algorithm is readily implemented on a sequential machine, it is hardly amenable to parallel implementation, e.g. on a systolic array. More specifically, an efficient pipelining of the two separate steps –update & triangular backsolve– appears to be impossible. For instance, on a triangular systolic array, the QR update runs from the upper-left corner to the lower-right corner of the array [Gentleman & Kung 1981], while the backsolve runs in precisely the opposite direction. One way of solving this problem consists in storing and updating R^{-1} as well, so that instead of the triangular backsolve, one only has to perform a matrix-vector multiplication :

$$x_{LS_{[k]}} = R^{-1}_{[k]} \cdot y_{[k]}.$$

[13] information matrix approach

It turns out that the same sequence of orthogonal transformations used to update R, can also be used to update R^{-1} [Pan & Plemmons 1989, Shepherd et al. 1990]. Indeed, from

$$\left[\begin{array}{cc} \frac{1}{\lambda}\cdot R_{[k-1]}^{-1} & 0 \end{array}\right]\cdot\left[\begin{array}{c} \lambda\cdot R_{[k-1]} \\ a_{[k]}^T \end{array}\right] = I_{n\times n}$$

it follows that

$$\underbrace{\left[\begin{array}{cc} \frac{1}{\lambda}\cdot R_{[k-1]}^{-1} & 0 \end{array}\right]\cdot Q_{[k]}}_{\left[\begin{array}{cc} X & \vdots \end{array}\right]}\cdot\underbrace{Q_{[k]}^T\cdot\left[\begin{array}{c} \lambda\cdot R_{[k-1]} \\ a_{[k]}^T \end{array}\right]}_{\left[\begin{array}{c} R_{[k]} \\ 0 \end{array}\right]} = I_{n\times n}$$

where then X must equal $R_{[k]}^{-1}$. In other words, if

$$Q_{[k]}^T\cdot\left[\begin{array}{c} \lambda\cdot R_{[k-1]} \\ a_{[k]}^T \end{array}\right] = \left[\begin{array}{c} R_{[k]} \\ 0 \end{array}\right]$$

then

$$Q_{[k]}^T\cdot\left[\begin{array}{c} \frac{1}{\lambda}\cdot R_{[k-1]}^{-T} \\ 0 \end{array}\right] = \left[\begin{array}{c} R_{[k]}^{-T} \\ \cdots \end{array}\right].$$

The above least squares scheme with inverse factorization can elegantly be implemented on a systolic array [Shepherd et al. 1990], see **Figure 5.1** *for a schematic view.*

However, a new problem comes into play when round-off error build-up is considered. First, as far as the R-factor is concerned, the combination of orthogonal transformations with exponential weighting –de-emphasizing older data as well as accumulated errors– is found to be sufficiently stable in practice. However, suppose that at time step $k-1$ the stored inverse factor $X_{[k-1]}$ slightly differs from the exact inverse, such that

$$X_{[k-1]} = R_{[k-1]}^{-1} - \delta X_{[k-1]}.$$

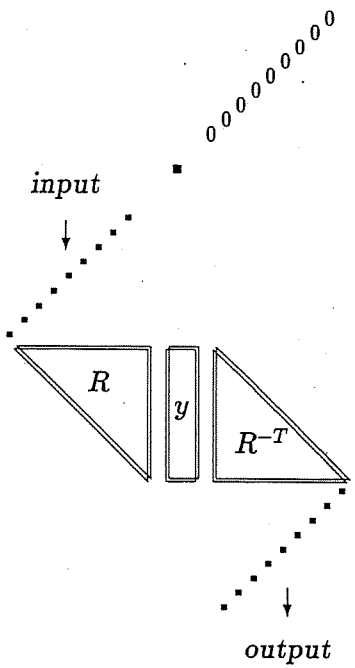


Figure 5.1: *RLS array with inverse factorization.*

From the updating formulas [14]

$$\underbrace{Q_{[k]}^T \cdot \begin{bmatrix} \frac{1}{\lambda} X_{[k-1]}^T \\ 0 \end{bmatrix}}_{\begin{bmatrix} X_{[k]}^T \\ \cdots \end{bmatrix}} = \underbrace{Q_{[k]}^T \cdot \begin{bmatrix} \frac{1}{\lambda} R_{[k-1]}^{-T} \\ 0 \end{bmatrix}}_{\begin{bmatrix} R_{[k]}^{-T} \\ \cdots \end{bmatrix}} - \underbrace{Q_{[k]}^T \cdot \begin{bmatrix} \frac{1}{\lambda} \delta X_{[k-1]}^T \\ 0 \end{bmatrix}}_{\begin{bmatrix} \delta X_{[k]}^T \\ \cdots \end{bmatrix}}$$

it then follows that the propagation of the global error is not necessarily stable. The norm $\|\delta X_{[k]}^T\|_F$ *can indeed be larger than* $\|\delta X_{[k-1]}^T\|_F$ *due to the fact that* $\frac{1}{\lambda} \cdot Q_{[k]}$ *is independent of* $\delta X_{[k]}^T$.

The result is that local errors can accumulate unboundedly, so that after a number of updates, the stored factors R and X are no longer each other's inverse. The computed LS-solutions are then subject to error as well

$$X_{[k]} \cdot y_{[k]} = \underbrace{R_{[k]}^{-1} \cdot y_{[k]}}_{x_{LS_{[k]}}} + \underbrace{\delta X_{[k]} \cdot y_{[k]}}_{\delta x_{LS_{[k]}}}.$$

An example of this will be given below. In the sequel, a procedure is developed where the inverse factor is continuously being re-computed, by making use of a Jacobi-type correction scheme applied to R and X. As all operations involving the R-factor are stable, the overall algorithm is then perfectly stable as well.

First we derive a second Jacobi-type inversion scheme, which is compatible with the RLS scheme [15]. *Next we show how the inversion scheme can be combined with the RLS scheme with inverse factorization. Finally a systolic implementation is derived.*

Jacobi-type matrix inversion -revisited

Suppose we are given a first upper triangular matrix $R_{n\times n}$ *and second one* $X_{n\times n}$, *which in view of the RLS scheme is meant to be close to* R^{-1}. *In view of our inversion algorithm, X is arbitrary. We develop a Jacobi-type algorithm for adjusting X to* R^{-1}, *wherewith R is only changed by orthogonal transformations. This algorithm generates a sequence of upper triangular matrices* R_k, X_k

$$\begin{aligned} R_k &= \Theta_k \ldots \Theta_1 \cdot R \cdot \Pi_1 \ldots \Pi_k \\ X_k &= T_k \ldots T_1 \cdot X \cdot \Theta_1^T \ldots \Theta_k^T \end{aligned}$$

with transformations Θ_k, Π_k *and* T_k *to be defined next. The transformations will be such that*

$$X_k = R_k^{-1} \qquad \text{for} \quad k = \frac{n(n-1)}{2}.$$

In other words, after a forward sweep, the triangular factors are each other's inverse. Furthermore, the procedure can be continued after the forward sweep. From then on, we will have

$$T_k^T = \Pi_k \qquad \text{for} \quad k > \frac{n(n-1)}{2}$$

so that the triangular factors stay each other's inverse. Finally, after each double sweep, the R matrix equals the original one:

$$R_k = R \qquad \text{for} \quad k = l \cdot n(n-1), \quad l \in Z.$$

The above transformations are of course embeddings of 2×2 *transformations in the* $\{i, i+1\}$ *plane for a certain pivot index i. The aim is to gradually transform* X_k *into* R_k^{-1}, *in other words to reduce* $X_k \cdot R_k$ *to* $I_{n\times n}$. *As both* R_k *and* X_k *are upper triangular* [16] *we again have*

$$\{X_k \cdot R_k\}_{i,i+1} = \{X_k\}_{i,i+1} \cdot \{R_k\}_{i,i+1}.$$

First of all, if we choose

$$\begin{aligned} \{\Pi_k\}_{i,i+1} &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ \{T_k\}_{i,i+1} &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \{R_{k-1}\}_{i,i+1}^{-1} \cdot \{X_{k-1}\}_{i,i+1}^{-1} \end{aligned}$$

[14] where for the time being we assumed that no additional local errors are introduced in time step k.

[15] only orthogonal row transformations together with column permutations are applied to the triangular factor R so that the least squares solution essentially remains unchanged –unlike in our first inversion scheme where the row transformations are not orthogonal.

[16] and the iterations will be such that the upper triangular structure is preserved at any stage

it follows that

$$\begin{aligned}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} &= \{T_k\}_{i,i+1}\cdot\{X_{k-1}\}_{i,i+1}\cdot\{R_{k-1}\}_{i,i+1}\cdot\{\Pi_k\}_{i,i+1}\\ &= \underbrace{\{T_k\}_{i,i+1}\cdot\{X_{k-1}\}_{i,i+1}\cdot\{\Theta_k\}^T_{i,i+1}}_{\{X_k\}_{i,i+1}}\cdots\\ &\qquad\cdots\underbrace{\{\Theta_k\}_{i,i+1}\cdot\{R_{k-1}\}_{i,i+1}\cdot\{\Pi_k\}_{i,i+1}}_{\{R_k\}_{i,i+1}}\end{aligned}$$

if $\{\Theta_k\}_{i,i+1}$ is chosen to be orthogonal. Hence, if $\{\Theta_k\}_{i,i+1}$ is a Givens rotation that upper triangularizes $\{R_{k-1}\}_{i,i+1}\cdot\{\Pi_k\}_{i,i+1}$, then both $\{X_k\}_{i,i+1}$ and $\{R_k\}_{i,i+1}$ will be upper triangular, and furthermore

$$\begin{aligned}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} &= \{X_k\}_{i,i+1}\cdot\{R_k\}_{i,i+1}\\ &= \{X_k\cdot R_k\}_{i,i+1}.\end{aligned}$$

The overall result of all the above transformations is that on the one hand X_k and R_k remain upper triangular throughout, while on the other hand in each iteration a 2×2 block on the main diagonal of $X_k\cdot R_k$ is set equal to I_2. Furthermore, the permutations furnish that all off-diagonal elements in $X_k\cdot R_k$ are annihilated in a succession for a forward sweep, where

$$\begin{aligned}i &= 1\ \ 2\ \ \ldots\ \ n-2\ \ n-1\\ &\quad 1\ \ 2\ \ \ldots\ \ n-2\\ &\quad\ \vdots\\ &\quad 1.\end{aligned}$$

We give an algorithmic description (forward sweep) and then illustrate the mechanism with a small example.

Algorithm triangular inversion (II) *:*

Initialization

$R \leftarrow R_0$

$X \leftarrow I_{n\times n}$ *random upper triangular*

Loop
for $j = 1, \ldots, n-1$
for $i = 1, \ldots, n-j$

compute transformations

$$\{\Pi_{i_j}\}_{i,i+1} = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$$

$$\{T_{i_j}\}_{i,i+1} = \{\Pi_{i_j}\}_{i,i+1}\cdot\{R\}^{-1}_{i,i+1}\cdot\{X\}^{-1}_{i,i+1}$$

$$\tan\theta = \frac{r_{i+1,i}}{r_{i,i}}$$

$$\{\Theta_{i_j}\}_{i,i+1} = \begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix}$$

apply transformations

$R \leftarrow \Theta_{i_j}\cdot R\cdot\Pi_{i_j}$

$X \leftarrow T_{i_j}\cdot X\cdot\Theta^T_{i_j}$

end
end

After the algorithm has terminated, X equals $R^{*^{-1}}$.

Example

R_0 and X_0 are two random upper triangular matrices

$$X_0 = \begin{bmatrix}-1.75 & -4.32 & -1.44 & -2.97\\ 0 & 2.21 & -0.24 & 4.94\\ 0 & 0 & 3.07 & 3.15\\ 0 & 0 & 0 & -4.72\end{bmatrix}$$

$$R_0 = \begin{bmatrix}-1.15 & -1.34 & -0.38 & -1.27\\ 0 & 0.37 & 0.01 & 0.40\\ 0 & 0 & 0.42 & 0.22\\ 0 & 0 & 0 & -0.19\end{bmatrix}$$

If the above iterative scheme is applied, one subsequently obtains for $X_k\cdot R_k$

	$X_k \cdot R_k$			
	2.01	0.77	−0.00	0.77
	0	0.81	−0.07	−0.12
	0	0	1.29	0.08
	0	0	0	0.90
1	1.00	0	−0.09	−0.15
	0	1.00	0.03	0.44
	0	0	1.29	0.08
	0	0	0	0.90
2	1.00	−0.09	0	−0.15
	0	1.00	0	0.06
	0	0	1.00	0.44
	0	0	0	0.90
3	1.00	−0.09	−0.15	0
	0	1.00	0.06	0
	0	0	1.00	0
	0	0	0	1.00
4	1.00	0	0.06	0
	0	1.00	−0.14	0
	0	0	1.00	0
	0	0	0	1.00
5	1.00	0.06	0	0
	0	1.00	0	0
	0	0	1.00	0
	0	0	0	1.00
6	1.00	0	0	0
	0	1.00	0	0
	0	0	1.00	0
	0	0	0	1.00

After exactly $\frac{n(n-1)}{2} = 6$ iterations, X_k is the inverse of R_k. Note that $R_6 \neq R_0$.

Suppose we would now continue this procedure. On easily checks that from now on

$$T_k = \Pi_k.$$

On the other hand, a subsequent (backward) sweep is known to restore the original column configuration. In other words

$$\begin{aligned} R_{n(n-1)} &= \Theta_{n(n-1)} \ldots \Theta_2 \cdot \Theta_1 \cdot R \cdot \underbrace{\Pi_1 \cdot \Pi_2 \ldots \Pi_{n(n-1)}}_{I_{n\times n}} \\ &= \Theta_{n(n-1)} \ldots \Theta_2 \cdot \Theta_1 \cdot R. \end{aligned}$$

Hence $\Theta_{n(n-1)} \ldots \Theta_2 \Theta_1$ must be upper triangular –and orthogonal!– in other words

$$\Theta_{n(n-1)} \ldots \Theta_2 \cdot \Theta_1 = I_{n\times n}.$$

From this we can conclude that

$$\begin{aligned} R_{n(n-1)} &= R \\ X_{n(n-1)} &= R^{-1}. \end{aligned}$$

Finally, the above formulas clearly hold true after each double sweep

$$\begin{aligned} R_{ln(n-1)} &= R \\ X_{ln(n-1)} &= R^{-1} \end{aligned}$$

for $l \in Z$.

Once again, each forward sweep can be combined with the subsequent backward sweep into a pipelined double sweep, where the pivot cyclicly takes up all the values

$$i = 1\ 2\ \ldots\ n-1.$$

A double sweep then corresponds to "n times ($i = 1 \ldots n-1$)".

RLS with stabilized inverse factorization

The above Jacobi-type algorithm can now be used in the RLS scheme with inverse factorization. In principle, the RLS algorithm is not

changed. In each time step, both R and $X \simeq R^{-1}$ are updated as was explained earlier. However, simultaneously, the Jacobi-type algorithm is applied to these matrices. As long as the two matrices are exactly each other's inverse, only permutations together with rotations Θ are performed. As soon as round-off comes into play, X is immediately being corrected. Clearly $X \cdot R - I_{n\times n}$ is kept to the level of the machine precision. In order to compute consistent results, three things should be taken care off.

1. *First off all, each time a row transformation Θ is applied to R, the same transformation should be applied to the right-hand side y as well, so that the least squares solution is not changed.*
2. *Next, if Π_{ac} denotes the accumulation of all previously applied permutations up to time step k*

   $$\Pi_{ac} = \Pi_1 \cdot \Pi_2 \ldots$$

   then a new data vector $a_{[k]}^T$ should be transformed accordingly, before the update

   $$\tilde{a}_{[k]}^T = a_{[k]}^T \cdot \Pi_{ac}$$

   whereafter the update can be computed by making use of the transformed input vector

   $$Q_{[k]}^T \cdot \begin{bmatrix} \lambda \cdot R_{[k-1]} \\ \tilde{a}_{[k]}^T \end{bmatrix} \cdot x = Q_{[k]}^T \cdot \begin{bmatrix} \lambda \cdot y_{[k-1]} \\ \beta_{[k]} \end{bmatrix}.$$

3. *Finally, the matrix-vector product $R_{[k]}^{-1} \cdot y_{[k]}$ clearly equals the least squares solution, up to the same permutation Π_{ac}*

   $$\begin{aligned} R_{[k]}^{-1} \cdot y_{[k]} &= \Pi_{ac}^{-1} \cdot x_{LS_{[k]}} \\ \Pi_{ac} \cdot R_{[k]}^{-1} \cdot y_{[k]} &= x_{LS_{[k]}}. \end{aligned}$$

Taking into account all the above changes, the overall RLS-algorithm looks as follows. We chose to perform $n-1$ corrective iterations after each update (pivot index $i = 1, \ldots, n-1$), as this leads to an efficient parallel implementation. The net result of this is that a double sweep in the correction scheme is performed after each n updates. On the other hand, the complexity of each update remains $O\{n^2\}$.

Algorithm :

Initialization

$$\begin{aligned} R &\Leftarrow R_{[0]} \\ y &\Leftarrow y_{[0]} \\ X &\Leftarrow I_{n\times n} \quad \textit{(random upper triangular matrix)} \\ \Pi_{ac} &\Leftarrow I_{n\times n} \end{aligned}$$

Loop

for $k = 1, \ldots, \infty$

input new observation $a_{[k]}$, $\beta_{[k]}$

1. *permutation*

   $$\tilde{a}_{[k]}^T = a_{[k]}^T \cdot \Pi_{ac}$$

2. *QR updating*

   $$\begin{bmatrix} R & y & X^T \\ 0 & . & \ldots \end{bmatrix} \Leftarrow Q_{[k]}^T \begin{bmatrix} \lambda \cdot R & \lambda \cdot y & \frac{1}{\lambda} \cdot X^T \\ \tilde{a}_{[k]}^T & \beta_{[k]} & 0 \end{bmatrix}$$

3. *error correction*

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

   $$\begin{aligned} R &\Leftarrow \Theta_{[i,k]} \cdot R \cdot \Pi_{[i,k]} \\ y &\Leftarrow \Theta_{[i,k]} \cdot y \\ X &\Leftarrow T_{[i,k]} \cdot X \cdot \Theta_{[i,k]}^T \\ \Pi_{ac} &\Leftarrow \Pi_{ac} \cdot \Pi_{[i,k]} \end{aligned}$$

   end

4. *output LS solution vector*

   $$x_{LS_{[k]}} = \Pi_{ac} \cdot X \cdot y$$

end

Note that the initialization of X is arbitrary, as it will be corrected anyway. The following example clearly shows the necessity of the additional correction scheme.

Example :

Suppose we choose a fixed solution vector x_{LS}° and generate random observations

$$a_{[k]} = \textit{random } n\textit{-vector}$$

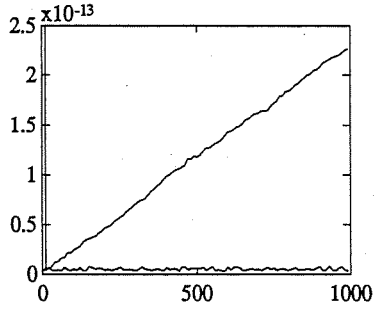


Figure 5.2: *Error vs. number of updates. Upper curve: without error correction. Lower curve: with error correction*

$$\beta_{[k]} = a_{[k]}^T \cdot x_{LS}^\circ.$$

For this noiseless time invariant system, the RLS algorithm is supposed to give an exact solution $x_{LS} = x_{LS}^\circ$ at each time step. The RLS algorithm was run for (only) 1000 time steps, both with and without the correction scheme, weighting factor equal to 0.9. The upper curve in **Figure 5.2** *shows the error $\|x_{LS} - x_{LS}^\circ\|_2$ for RLS with inverse factorization, without error corrections. Clearly, the error grows linearly* [17]*. With the additional error corrections, the algorithm is apparently stable, lower curve.*

Systolic implementation

In Chapter 3, it was shown how Jacobi-type algorithms can be combined with QR updating on a systolic array, resulting in an SVD updating array. The RLS-array to be presented next is merely a replica of this, with a few minor changes. Therefore, we only give a brief description of the array, referring to Chapter 3 for further details.

An outline of the array is exhibited in **Figure 5.3.** *It consists of two triangular processor arrays that contain R (upper triangular) and $X^T \simeq R^{-T}$ (lower triangular) respectively. The right-hand side vector*

[17] the machine precision is approximately 1e-16.

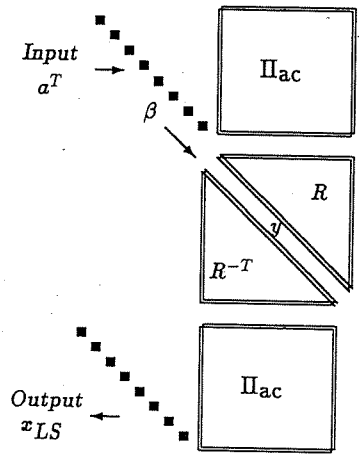


Figure 5.3: *Outline RLS array*

y is stored in between on the main diagonal. Above and below this array, a square array is added, in which the permutations are accumulated (permutation matrix Π_{ac}). Both arrays perform exactly the same operations, although the pipelining is different, see below. Observation vectors are continuously fed in to the left-hand side of the upper array, while LS-solution vectors become available (at the same rate) at the left-hand side of the lower array.

Figure 5.4 *shows the array operations in more detail. For the sake of clarity, only two successive updates are shown, although it should be stressed that new data vectors are fed in continuously. The 2×2 frames in the first place correspond to operations in the error correction scheme. Processor elements on the main diagonal compute and (locally) apply the necessary transformations . Row transformations $\{\Theta\}_{i,i+1}$ are applied to the available y-components as well. The row transformation parameters are then passed on to the right (R-matrix)*

and to the left (X-matrix), while column transformation parameters are passed on upwards for R (permutations $\{\Pi\}_{i,i+1}$), downwards for X^T (transformations $\{T\}^T_{i,i+1}$). Column permutations $\{\Pi\}_{i,i+1}$ are also propagated through the upper and lower square part, containing the permutation matrix Π_{ac} [18]. The data vectors a^T are fed in into the upper Π_{ac}-array, where the matrix-vector product $\tilde{a}^T = a^T \cdot \Pi_{ac}$ is computed. The resulting vector $\tilde{a}$ becomes available at the top end of the square array, and is bounced back into the array towards the triangular arrays, where the QR updating is interlaced with the Jacobi procedure. The rotations for the QR update are propagated both to the left (X^T-update) and to the right (R-update).

The y-vector, stored on the main diagonal, can easily be updated along with the R-update, if β is fed in as soon as the first $\tilde{a}^T$-component reaches the triangular array. Finally, the LS solution vector (equal to $\Pi_{ac} \cdot X \cdot y$) is generated as follows. The matrix-vector product $X \cdot y$ is computed along with the X^T-update, wherewith y-components are propagated to the left, while intermediate results are propagated downwards. The intermediate result $X \cdot y$ runs into the lower square array, where the last multiplication $\Pi_{ac} \cdot (X \cdot y)$ can be performed in between two permutation fronts. The LS solution vector becomes available at the left-hand side of the array.

For further details we refer to Chapter 3.

5.3 Recursive least squares with orthogonal-inverse updating

While the example of the previous section shows the necessity of the additional correction scheme, it should be clear that there is a significant computational overhead here. First of all, if the operation count per update for the basic RLS scheme is $O\{n^2\}$, then it is easily seen that the RLS scheme with inverse factorization requires roughly twice as much operations, i.e. $O\{2n^2\}$ per update, as two triangular factors are updated instead of one. The correction scheme introduces additional computations, roughly $O\{n^2\}$ for the R-adjustment (the right-hand transformations are only permutations) and $O\{2n^2\}$ for the X-correction. The overall operation count is thus $O\{5n^2\}$,

[18] Π_{ac}'s first row in the top row, etc.

Figure a

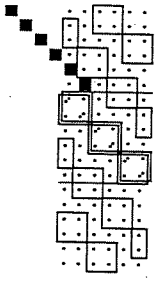


Figure b

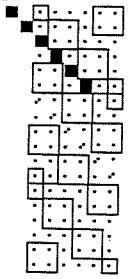


Figure c

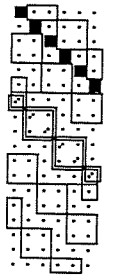


Figure d

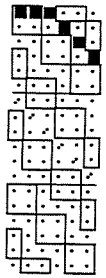


Figure e

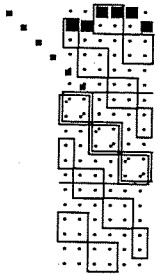


Figure f

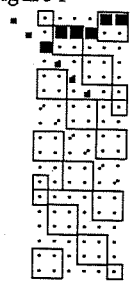


Figure g

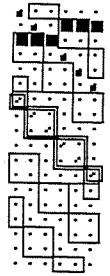


Figure h

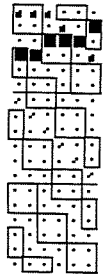


Figure i

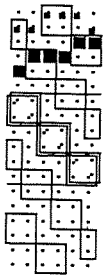


Figure j

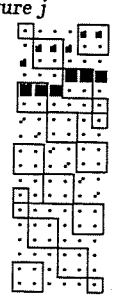


Figure k

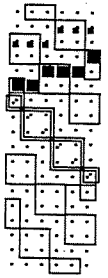


Figure l

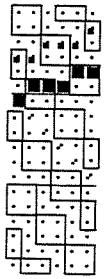


Figure m

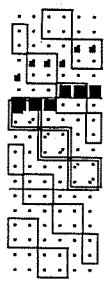


Figure n

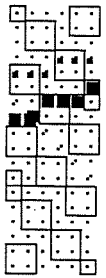


Figure o

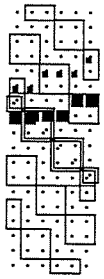


Figure p

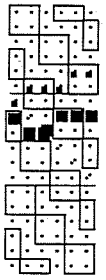


Figure q

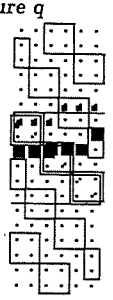


Figure r

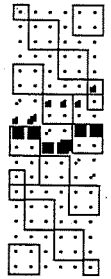


Figure s

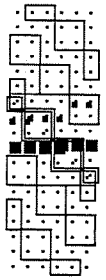


Figure t

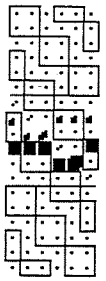


Figure u

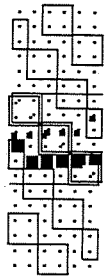


Figure v

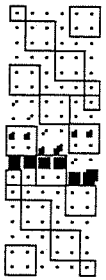


Figure w

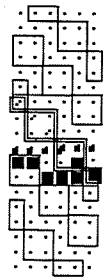


Figure x

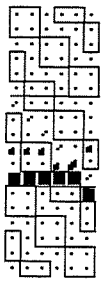


Figure y

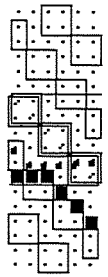


Figure z

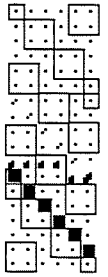


Figure A

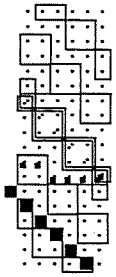


Figure B

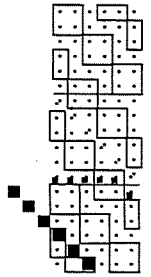


Figure C

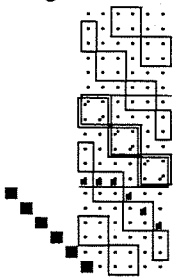


Figure D

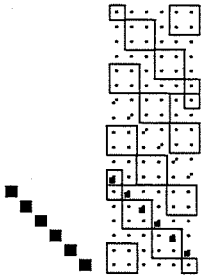


Figure E

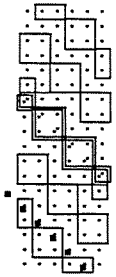


Figure F

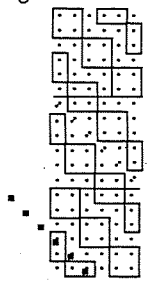


Figure 5.4: *RLS* array

which is excessive. In this section, a much more efficient implementation is derived. We adopt an algorithm from [Pan & Plemmons 1989], and map it onto a systolic array that again has crossing computational fronts in a Jacobi-type partitioning, similar to the SVD updating array. As far as we know, this array is the first to implement a complete RLS algorithm in a stable and efficient manner. The correctness of the presented array has been verified by software simulation.

In the RLS scheme of [Pan & Plemmons 1989], only R^{-1} is stored and updated, where the transformations for updating R^{-1} follow from the matrix-vector product $-a_{[k]}^T \cdot R^{-1}$ as follows.

Algorithm Orthogonal-Inverse Updating :

Initialization

$$R^{-1} \Leftarrow R_{[0]}^{-1}$$
$$x_{LS} \Leftarrow x_{LS_{[0]}}$$

Loop

for $k = 1, \ldots, \infty$

input new observation $a_{[k]}$, $\beta_{[k]}$

1. *matrix-vector multiplication*

$$s_{[k]}^T = -a_{[k]}^T \cdot R^{-1}$$

2. *Determine orthogonal plane rotations* $\underline{Q}_l$ *so that*

$$\begin{bmatrix} 1 & \underbrace{s_{[k]}^T}_{1\times n} \end{bmatrix} \cdot \underline{Q}_1 \underline{Q}_2 \cdots \underline{Q}_n = \begin{bmatrix} \delta_{[k]} & \underbrace{0^T}_{1\times n} \end{bmatrix}$$

3. *Update* R^{-1}

$$\begin{bmatrix} u_{[k]} & R^{-1} \end{bmatrix} \Leftarrow \begin{bmatrix} 0 & R^{-1} \end{bmatrix} \cdot \underline{Q}_1 \underline{Q}_2 \cdots \underline{Q}_n$$

4. *Update LS solution*

$$\underline{x}_{LS} = x_{LS} - \frac{\beta_{[k]} - a_{[k]}^T x_{LS}}{\delta_{[k]}} \cdot u_{[k]}$$

end

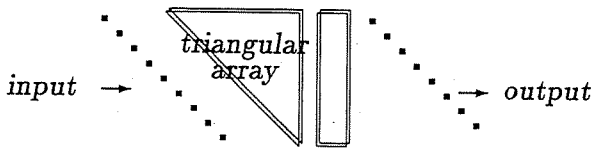


Figure 5.5: *Preview RLS array*

The vector $-\frac{u_{[k]}}{\delta_{[k]}}$ is known as the Kalman gain vector. The plane rotations in Step 2 are defined by

$$\underline{Q}_l = \begin{bmatrix} \cos\theta_l & 0 & \sin\theta_l & 0 \\ 0 & I_{l-1} & 0 & 0 \\ -\sin\theta_l & 0 & \cos\theta_l & 0 \\ 0 & 0 & 0 & I_{n-l} \end{bmatrix}$$

where θ_l is chosen such that the l+1st element in $\begin{bmatrix} 1 & s_{[k]}^T \end{bmatrix} \cdot \underline{Q}_1 \underline{Q}_2 \cdots \underline{Q}_{l-1}$ is zeroed. For details, we refer to [Pan & Plemmons 1989].

In the sequel we develop a systolic array for this orthogonal-inverse updating algorithm. First we show how to implement one recursive update. Then we show how subsequent updates can be pipelined, once again by introducing additional computations. The computational overhead is such that the overall efficiency is roughly 67 %, while the throughput is $O\{n^0\}$.

A preview of the array is given in **Figure 5.5**. *Briefly, the triangular array stores and updates R^{-1}, while the additional column to the right stores and updates the LS solution x_{LS}. New data are fed in to the left, one vector at a time, while the LS solutions run out to the right at the same rate. Note at once that the data flow is completely different in the original Gentleman-Kung array.*

Implementing one single update

One single update of the above orthogonal-inverse scheme is easily implemented on a triangular array. **Figure 5.6** *exhibits a* 10×10 *example.*

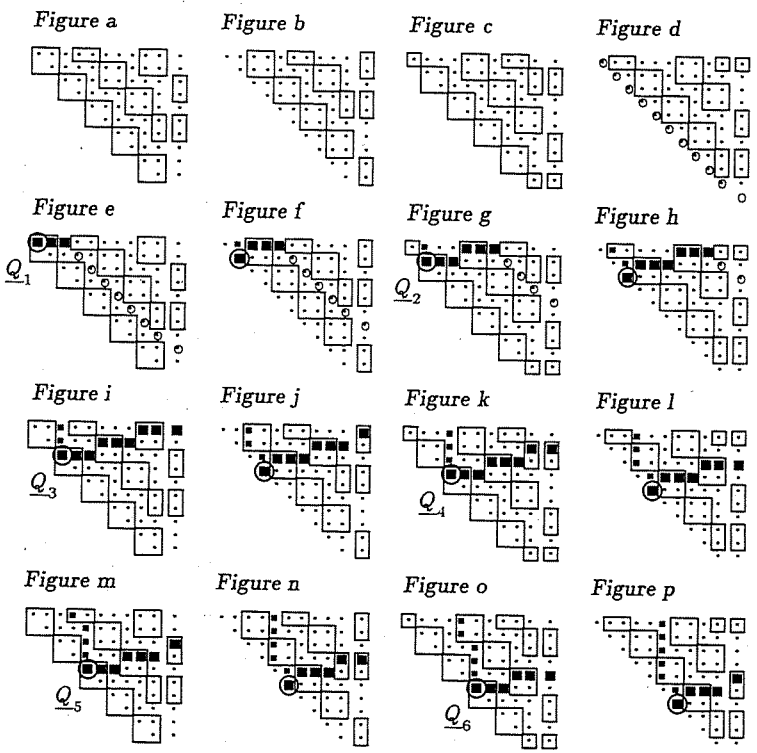
1. The first step in the orthogonal-inverse updating scheme is a matrix-vector multiplication

$$s^T = -a^T \cdot R^{-1}$$

and this is readily implemented on the array. The a-vector is fed in as indicated in Figure 2d with the ∘'s, and propagated to the right. Meanwhile the product terms are accumulated and propagated upwards. The s-components run out at the top end of the array. As for the right-hand column containing x_{LS}*, we can adopt much the same functionality to compute* $\beta - a^T x_{LS}$ *(used in step 4 of the algorithm), if the* β *is fed in from the bottom. We can already point out that also throughout the rest of our description, the operations in the right-hand column hardly differ from those in the triangle.*

2. As soon as the s-components become available at the top end of the array, they are propagated downwards, towards the diagonal, as indicated with the ■'s. Similarly, $\beta - a^T x_{LS}$ *is propagated downwards in the right-hand column. The plane rotations* $\underline{Q}_1$ *through* $\underline{Q}_n$ *in the second step can then be computed on the main diagonal. Rotation* $\underline{Q}_1$ *is computed in Figure 2e as indicated,* $\underline{Q}_2$ *in Figure 2g, etc. Here we make use of an auxiliary variable* $\delta_\circ$*, propagated along the diagonal and indicated with the* $\odot$*. This variable is initially equal to 1 (for* $\underline{Q}_1$*) and equal to* δ *after* $\underline{Q}_{10}$ *(*$\underline{Q}_n$*), see step 2 of the algorithmic description. Rotation* $\underline{Q}_1$ *zeroes the first component of s, which then drops out,* $\underline{Q}_2$ *zeroes the second component, etc. After Figure 2w, the s-vector is completely zeroed, while* $\delta_\circ$ *equals* δ*, which will be used for the* x_{LS} *update in step 4.*

3. In the third step, the inverse factor R^{-1} *is updated by making use of the computed rotations* $\underline{Q}_1$ *through* $\underline{Q}_n$ *and an auxiliary vector* $u_\circ$ *which initially equals zero. Rotation* $\underline{Q}_1$ *creates a fill-in in the first position of this vector,* $\underline{Q}_2$ *creates a fill-in in the second position, etc. After* $\underline{Q}_{10}$*, one ends up with the u-vector. The rotations which are*



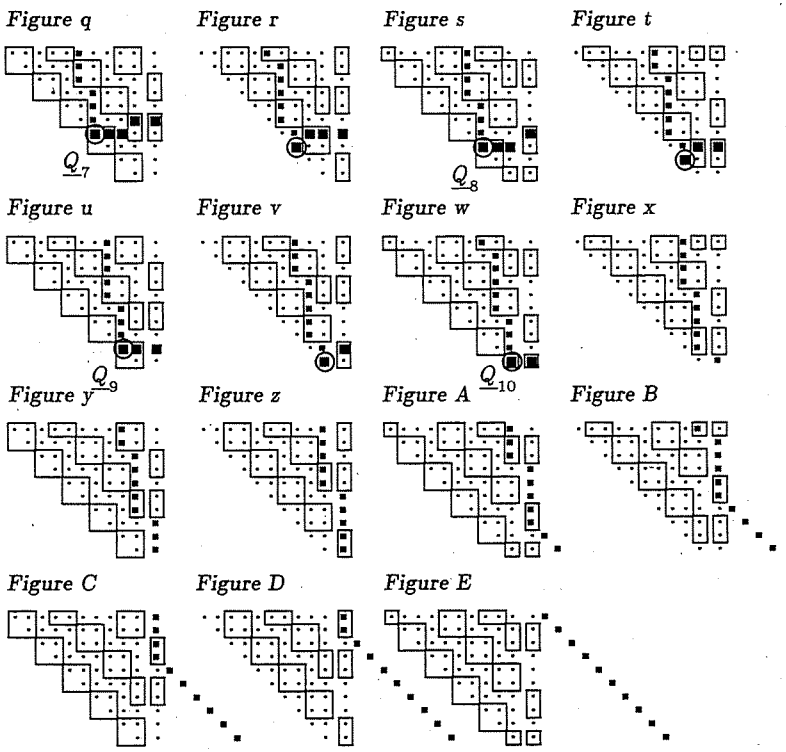


Figure 5.6: *Data flow for one update*

computed on the main diagonal are therefore propagated upwards, while the auxiliary vector u_o, indicated with the ▪'s, is propagated to the right.

4. Again the same functionality can be adopted for the right-hand column to perform the fourth step. Instead of a rotation $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ *one can now propagate the following transformation*

$$\begin{bmatrix} & 1 & & 1 \\ -\frac{\beta - a^T x_{LS}}{\delta} & & -\frac{\beta - a^T x_{LS}}{\delta} & \end{bmatrix}$$

such that the x_{LS} in the last column is updated, and furthermore copied into the ▪-vector, which is then output. Note that both $\beta - a^T x_{LS}$ (propagated downwards) and δ (propagated along the diagonal) are by then available in the lower right corner.
The updated least squares solution $\underline{x}_{LS}$ is completely available in Figure 5.6E.

Pipelining different updates

One complete update, as it was detailed in the previous section, requires $O\{n\}$ time steps (from Figure 5.6e through 5.6E). An $O\{n^0\}$ throughput can only be obtained if successive updates can be pipelined. Suppose we would input a new data vector at each odd phase, in other words in Figure 5.6e-i-m-etc. Different updates then slightly interfere and in order to take care of this, we will have to introduce a few additional operations.

In order to make things clear, let us first pick out two arbitrary updates and see how they interfere. In **Figure 5.7**, *a first update is still being completed, while a second one starts off in Figure 5.7e (o's). Let us refer to the first update with a_1, β_1, s_1 and u_1, and to the second update with a_2, β_2, s_2 and u_2. The first update generates $\underline{R}^{-1}$ and $\underline{x}_{LS}$, the second one generates $\underline{\underline{R}}^{-1}$ and $\underline{\underline{x}}_{LS}$. As the first update is not yet completed, the computed matrix-vector multiplication in the second update will be $s_{\natural 2}^T = -a_2^T \cdot \underline{R}_{\natural}^{-1}$, instead of $s_2^T = -a_2^T \cdot \underline{R}^{-1}$, where $\underline{R}_{\natural}^{-1}$ is some intermediate version of R^{-1}. The matrices $\underline{R}^{-1}$ and $\underline{R}_{\natural}^{-1}$ are related as follows*

$$\begin{bmatrix} u_{\natural} & \underline{R}_{\natural}^{-1} \end{bmatrix} \cdot Q_{\natural} = \begin{bmatrix} u_1 & \underline{R}^{-1} \end{bmatrix}$$

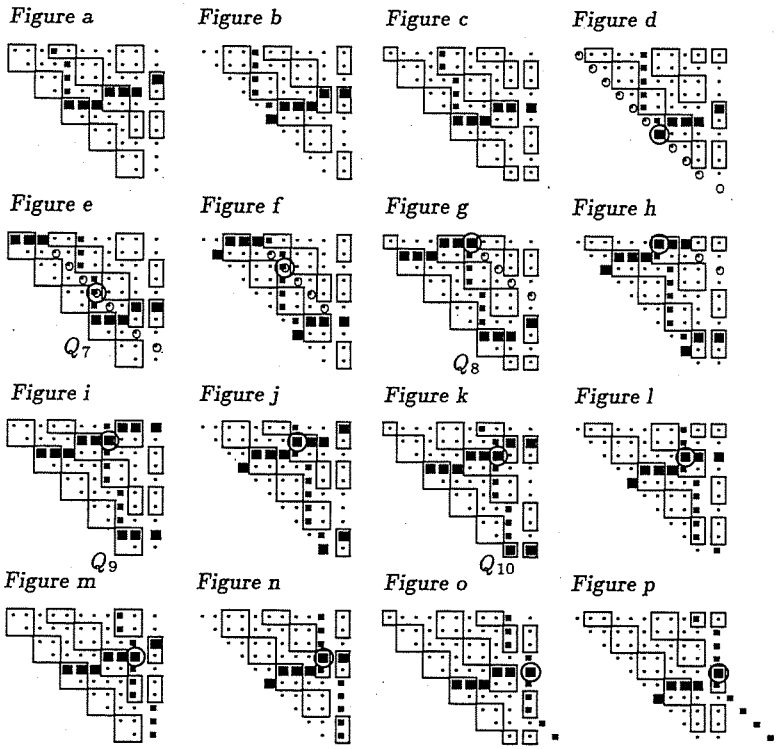


Figure 5.7: *Data flow and interaction of successive updates*

where $u_\natural$ is the current value for the $u_\circ$ vector in the first update, and $Q_\natural$ represents a number of rotations which are yet to be carried out in that same update. In our case

$$Q_\natural = Q_7 Q_8 Q_9 Q_{10}.$$

From this it readily follows that

$$\begin{bmatrix} -a_2^T \cdot u_\natural & \underbrace{-a_2^T \cdot \underline{R}_\natural^{-1}}_{s_{\natural 2}^T} \end{bmatrix} \cdot Q_\natural = \begin{bmatrix} -a_2^T \cdot u_1 & \underbrace{-a_2^T \cdot \underline{R}^{-1}}_{s_2^T} \end{bmatrix}$$

so that the vector s_2^T can be recovered by applying the rotations $Q_\natural$ onto $\begin{bmatrix} -a_2^T \cdot u_\natural & s_{\natural 2}^T \end{bmatrix}$. The trick is thus to compute $-a_2^T \cdot \underline{R}_\natural^{-1}$ as well as $-a_2^T \cdot u_\natural$ and then afterwards apply the $Q_\natural$ rotations to get s_2. This is a computational overhead of course, but it allows for an efficient pipelining of successive updates, so that the throughput is $O\{n^0\}$.

In Figure 5.7, the vector-vector product $-a_2^T \cdot \mathrm{aux}_\natural$ is computed along with the matrix-vector product. Intermediate products are indicated with the $\odot$ in Figures 5.7e through 5.7g. The resulting $\odot = -a_2^T \cdot u_\natural$ is then propagated downwards, together with the $s_{\natural 2}$-components. Rotations Q_7 through Q_{10} are seen to be applied to $\begin{bmatrix} \odot & s_{\natural 2}^T \end{bmatrix}$ in Figure 5.7h-j-l-n respectively. The $\odot$ is seen to be propagated downwards and rightwards, thus always following the crossing of $u_\natural$ and s_2.

As for the right-hand column, one can similarly verify that first $\beta_2 - a_2^T x_{LS}$ is accumulated from the bottom to the top, instead of $\beta_2 - a_2^T \underline{x}_{LS}$. The correct value is then obtained as follows

$$\begin{aligned} \beta_2 - a_2^T \underline{x}_{LS} &= \beta_2 - a_2^T \cdot (x_{LS} - \frac{\beta_1 - a_1^T x_{LS}}{\delta_1} \cdot u_1) \\ &= \begin{bmatrix} \beta_2 - a_2^T x_{LS} & -a_2^T u_1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -\frac{\beta_1 - a_1^T x_{LS}}{\delta_1} \end{bmatrix}. \end{aligned}$$

The $\odot$ that runs out of the triangular array into the right-hand column (Figure 5.7n-0) is known to be $-a_2^T u_1$, and as $\begin{bmatrix} 1 \\ -\frac{\beta_1 - a_1^T x_{LS}}{\delta_1} \end{bmatrix}$ happens to be (part of) the upgoing transformation in the right-hand column,

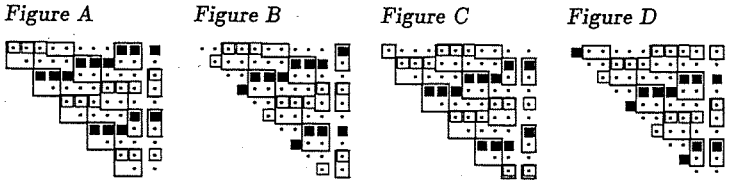


Figure 5.8: *Distribution of successive s-vectors over the array*

the functionality in the right-hand column is again seen to be roughly the same as in the triangular array.

In Figure 5.7, it has been outlined how two (only two) updates interfere. With one update starting in each odd phase, the situation is of course not any different. Several vector-vector products are then computed along with each matrix-vector multiplication, etc. Remarkably enough, the functionality of the 2×2 *blocks is extremely simple. Successive s-vectors are then distributed over the grid as depicted in* **Figure 5.8**. *Here successive s-vectors are indicated alternately with* ■*'s and* □*'s. It is seen that each framed* 2×2 *block has two s-components in its upper part. Similarly, the* u_o *vectors are distributed over the grid as depicted in* **Figure 5.9**, *were we used* ▪*'s and* ▫*'s. It is seen that each framed* 2×2 *(off-diagonal) block also has two auxiliary vector components in its left part. The functionality of all the* 2×2 *blocks could then be as explained above. Additionally it will also have a* $\odot$ *in its upper left corner. All* 2×2 *blocks therefore always perform exactly the same operations. The operations in the right-hand column are essentially the same. Functional descriptions are left out for the sake of brevity, and can be found in [Moonen & Vandewalle 1990f]. Finally, from this it is also seen that one third of these operations are essentially overhead, so that the overall efficiency is roughly 67%. As far as we know, this array is the first to implement a complete RLS algorithm in a stable and efficient manner.*

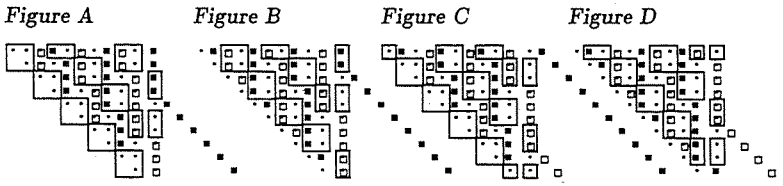


Figure 5.9: *Distribution of successive* u_o*-vectors over the array*

5.4 Kalman filtering

The Kalman filter is a fundamental tool for state estimation in such areas as signal processing and modern control. However, its high computational demand has limited its use in e.g. broadband real-time applications. Recently, with the advent of VLSI technology, making parallel information processing more and more feasible, several dedicated systolic or wavefront architectures for Kalman filtering have been suggested [Sung & Hu 1986, Jover & Kailath 1986, Kung 1988, Gaston & Irwin 1989].

Most of the suggested arrays make use of efficient updating formulas for square root information filtering, which reduce the Kalman filtering problem to a structured QR *updating problem. However, if the* QR *updating is implemented as such –e.g. on a Gentleman-Kung* QR *updating array– without taking into account its special structure, the overall efficiency is seen to be particularly low, with a processor utilization of hardly 17% for large values of* n*, the system order [Gaston & Irwin 1989]. The challenge is thus to design a tailor-made array, exploiting the available structure as much as possible.*

The array in [Kung 1988] seems to be the most efficient array for Kalman filtering ever described in the literature. The processor utilization ranges from 50% (for $l \ll n$*, where* l *is the number of outputs) up to 60% (for* $l = n$*), for large values of* n*. This array with* $\frac{1}{2}n^2 + O\{n\}$ *processors essentially consists of a Gentleman-Kung array, plus a square data buffer to its right to store certain rotation parameters. The top row of the data buffer is linked to the diagonal elements of the triarray. These cross-links run all the way around the array, and therefore disturb the array's overall regularity –see Figure 8.9 in [Kung 1988].*

A different approach was taken in [Gaston & Irwin 1989], where the

triarray is turned into a rectangular array with $n^2 + O\{n\}$ processors, by superimposing two sections of the original array. For large values of n, the processor utilization ranges from 33% (for $l \ll n$) up to 37.5% (for $l = n$). An advantage seems to be that data always move into the same direction (characteristic not shared by the Kung-array). On the other hand, the array has similar cross-links.

In this section, yet another approach is taken, where use is made of a triangular array of $\frac{1}{4}n^2 + O\{n\}$ processors in a Jacobi-type partitioning. Use is made of a Jacobi-type algorithm for QR factorization on the one hand, together with QR updating on the other hand. As far as efficiency is concerned, it keeps the mean between the above two arrays. For $l \ll n$, the overall efficiency is the same as for the Kung array (50%), whereas for $l = n$ the efficiency is approximately 37.5%, similar to the array by Gaston & Irwin. The present architecture however possesses a much more regular structure, -e.g. without additional cross-links or data buffers- and is therefore thought to offer a useful alternative in view of hardware implementation.

We first briefly recall the recursive least squares formulation for the Kalman filter, leading to efficient updating formulas for square root information filtering. Next, we slightly re-organize these computations, resulting in a combination of QR updating on the one hand, with a Jacobi-type algorithm for QR factorization on the other hand. Finally, it is shown how all this can be implemented on a systolic array.

RLS formulation of square root information filtering (SRIF)

Suppose we are given a time-varying linear system, with a state space model as follows

$$\begin{aligned} x_{[k]} &= A_{[k]}x_{[k-1]} + B_{[k]}u_{[k]} + v_{[k]} \\ y_{[k]} &= C_{[k]}x_{[k]} + D_{[k]}u_{[k]} + w_{[k]}. \end{aligned}$$

The matrices $A_{[k]}$, $B_{[k]}$, $C_{[k]}$ and $D_{[k]}$ are the known system matrices at time step k. As a reminder, vectors $u_{[k]}$, $y_{[k]}$ and $x_{[k]}$ denote the input (m-vector), output (l-vector) and the state vector at time step k, the dimension of $x_{[k]}$ being the minimal system order n. Only an initial expected value for $x_{[0]}$ is available, namely $\hat{x}_{[0]}$, together with its error covariance matrix

$$\begin{aligned} E\{x_{[0]}\} &= \hat{x}_{[0]} \\ E\{(x_{[0]} - \hat{x}_{[0]})(x_{[0]} - \hat{x}_{[0]})^T\} &= P_{[0]}. \end{aligned}$$

Furthermore, $v_{[k]}$ and $w_{[k]}$ are unknown noise sequences -process noise and measurement noise- with zero mean and known covariance matrices

$$\begin{aligned} E\{v_{[k]}\} &= 0 \\ E\{v_{[k]}v_{[k]}^T\} &= V_{[k]} \\ \\ E\{w_{[k]}\} &= 0 \\ E\{w_{[k]}w_{[k]}^T\} &= W_{[k]} \\ \\ E\{v_{[k]}w_{[k]}^T\} &= 0. \end{aligned}$$

In the sequel, we often make use of the square roots (Cholesky factors) of the inverses of these covariance matrices, defined as follows

$$\begin{aligned} V_{[k]}^{-1} &= R_{[k]}^{v^T} \cdot R_{[k]}^{v} \\ W_{[k]}^{-1} &= R_{[k]}^{w^T} \cdot R_{[k]}^{w} \\ P_{[0]}^{-1} &= R_{[0]}^{p^T} \cdot R_{[0]}^{p} \end{aligned}$$

where each time $R_{[k]}^{\cdot}$ denotes an upper triangular matrix.

The aim of the Kalman filter is to track the state vector $x_{[k]}$, by making use of observations of the in- and outputs up to time k, and the available system matrices together with the known covariances.

A least squares formulation of the Kalman filter follows from the observation that at a certain time step k, the state space equations can be assembled in a large overdetermined set of linear equations, with a vector of unknowns containing all previous state vectors, up to $x_{[k]}$

[Duncan & Horn 1972, Paige & Saunders 1977]:

$$\begin{bmatrix} \hat{x}_{[0]} \\ -B_{[1]}u_{[1]} \\ y_{[1]} - D_{[1]}u_{[1]} \\ -B_{[2]}u_{[2]} \\ y_{[2]} - D_{[2]}u_{[2]} \\ \vdots \\ -B_{[k]}u_{[k]} \\ y_{[k]} - D_{[k]}u_{[k]} \end{bmatrix} = \begin{bmatrix} \boxed{\begin{matrix} I & 0 & 0 & \cdots \\ A_{[1]} & -I & 0 & \cdots \\ 0 & C_{[1]} & 0 & \cdots \\ 0 & A_{[2]} & -I & \cdots \\ 0 & 0 & C_{[2]} & \cdots \\ \vdots & \vdots & \vdots & \end{matrix}} & \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{matrix} \\ \begin{matrix} 0 & 0 & 0 & A_{[k]} \\ 0 & 0 & 0 & 0 \end{matrix} & \begin{matrix} -I \\ C_{[k]} \end{matrix} \end{bmatrix} \cdot \begin{bmatrix} x_{[0]} \\ x_{[1]} \\ x_{[2]} \\ \vdots \\ x_{[k]} \end{bmatrix} + \begin{bmatrix} e_{[0]} \\ v_{[1]} \\ w_{[1]} \\ v_{[2]} \\ w_{[2]} \\ \vdots \\ v_{[k]} \\ w_{[k]} \end{bmatrix}.$$

Alternatively, this is recast as follows

$$\mathbf{b}_{[k]} = \underbrace{\mathbf{A}_{[k]}}_{(kn+n+kl)\times(kn+n)} \cdot \mathbf{x}_{[k]} + \mathbf{e}_{[k]}.$$

In each time step, this set of equations is updated by appending appropriate rows and columns –indicated by the frames. If the covariances for $e_{[0]}(= \hat{x}_{[0]} - x_{[0]})$, $v_{[k]}$ and $w_{[k]}$ differ from the identity, it is necessary to first perform a pre-whitening of **e**, *by making use of the known square roots:*

$$\begin{bmatrix} \underline{\hat{x}}_{[0]} \\ -\underline{B}_{[1]}u_{[1]} \\ \underline{y}_{[1]} - \underline{D}_{[1]}u_{[1]} \\ -\underline{B}_{[2]}u_{[2]} \\ \underline{y}_{[2]} - \underline{D}_{[2]}u_{[2]} \\ \vdots \\ -\underline{B}_{[k]}u_{[k]} \\ \underline{y}_{[k]} - \underline{D}_{[k]}u_{[k]} \end{bmatrix} = \begin{bmatrix} \boxed{\begin{matrix} R^p_{[0]} & 0 & 0 & \cdots \\ \underline{A}_{[1]} & -R^v_{[1]} & 0 & \cdots \\ 0 & \underline{C}_{[1]} & 0 & \cdots \\ 0 & \underline{A}_{[2]} & -R^v_{[2]} & \cdots \\ 0 & 0 & \underline{C}_{[2]} & \cdots \\ \vdots & \vdots & \vdots & \end{matrix}} & \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{matrix} \\ \begin{matrix} 0 & 0 & 0 & \underline{A}_{[k]} \\ 0 & 0 & 0 & 0 \end{matrix} & \begin{matrix} -R^v_{[k]} \\ \underline{C}_{[k]} \end{matrix} \end{bmatrix} \cdot \begin{bmatrix} x_{[0]} \\ x_{[1]} \\ x_{[2]} \\ \vdots \\ x_{[k]} \end{bmatrix} + \begin{bmatrix} \underline{e}_{[0]} \\ \underline{v}_{[1]} \\ \underline{w}_{[1]} \\ \underline{v}_{[2]} \\ \underline{w}_{[2]} \\ \vdots \\ \underline{v}_{[k]} \\ \underline{w}_{[k]} \end{bmatrix}.$$

where

$$\begin{aligned} \underline{\hat{x}}_{[0]} &= R^p_{[0]} \cdot \hat{x}_{[0]} \\ \underline{e}_{[0]} &= R^p_{[0]} \cdot e_{[0]} \\ \underline{A}_{[i]} &= R^v_{[i]} \cdot A_{[i]} \end{aligned}$$

$$\begin{aligned} \underline{B}_{[i]} &= R^v_{[i]} \cdot B_{[i]} \\ \underline{C}_{[i]} &= R^w_{[i]} \cdot C_{[i]} \\ \underline{D}_{[i]} &= R^w_{[i]} \cdot D_{[i]} \\ \\ \underline{y}_{[i]} &= R^w_{[i]} \cdot y_{[i]} \\ \underline{v}_{[i]} &= R^v_{[i]} \cdot v_{[i]} \\ \underline{w}_{[i]} &= R^w_{[i]} \cdot w_{[i]} \end{aligned}$$

or in short

$$\underline{\mathbf{b}}_{[k]} = \underline{\mathbf{A}}_{[k]} \cdot \mathbf{x}_{[k]} + \underline{\mathbf{e}}_{[k]}$$

with

$$E\{\underline{\mathbf{e}}_{[k]} \cdot \underline{\mathbf{e}}^T_{[k]}\} = I.$$

The set of equations is solved in a least squares sense by first performing a QR factorization of $\underline{\mathbf{A}}_{[k]}$,

$$\underbrace{Q^T \cdot \underline{\mathbf{b}}_{[k]}}_{\begin{bmatrix} \mathbf{y}_{[k]} \\ \mathbf{z}_{[k]} \end{bmatrix}} = \underbrace{Q^T \cdot \underline{\mathbf{A}}_{[k]}}_{\begin{bmatrix} \mathbf{R}_{[k]} \\ 0 \end{bmatrix}} \cdot \mathbf{x}_{[k]} + Q^T \cdot \underline{\mathbf{e}}_{[k]}$$

followed by a triangular backsolve

$$\mathbf{y}_{[k]} = \mathbf{R}_{[k]} \cdot \hat{\mathbf{x}}_{[k]}.$$

The last part of the solution vector $\hat{\mathbf{x}}_{[k]}$ is an estimate for $\mathbf{x}_{[k]}$, given observations up to time step k, and is exactly equal to what one would obtain by applying the conventional formulas for Kalman filtering.

Let us now have a look at one specific update, e.g. at time step k. Suppose that the set of equations at time step $k-1$ has been reduced to

a triangular set, and we append new rows and columns corresponding to time step k:

$$\begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \\ -\underline{B}_{[k]}u_{[k]} \\ \underline{y}_{[k]} - \underline{D}_{[k]}u_{[k]} \end{bmatrix} \simeq \begin{bmatrix} \boxed{\begin{matrix} * & * & * & * & * & * & * & * \\ & * & * & * & * & * & * & * \\ & & * & * & * & * & * & * \\ & & & * & * & * & * & * \\ & & & & * & * & * & * \\ & & & & & * & * & * \\ & & & & & & * & * \\ & & & & & & & * \end{matrix}} & & 0 \\ & \underline{A}_{[k]} & -R^v_{[k]} \\ & 0 & \underline{C}_{[k]} \end{bmatrix} \cdot \begin{bmatrix} x_{[0]} \\ x_{[1]} \\ x_{[2]} \\ \vdots \\ x_{[k-1]} \\ x_{[k]} \end{bmatrix}.$$

Clearly, in order to reduce this set to a triangular set, it suffices to upper triangularize a smaller lower right submatrix:

$$\begin{bmatrix} \\ \boxed{\begin{matrix} * \\ * \\ * \end{matrix}} \\ -\underline{B}_{[k]}u_{[k]} \\ \underline{y}_{[k]} - \underline{D}_{[k]}u_{[k]} \end{bmatrix} \simeq \begin{bmatrix} * & * & * & * & * & & & \\ & * & * & * & * & & & \\ & & * & * & * & & & \\ & & & * & * & & & \\ & & & & * & & & \\ & & & & & \boxed{\begin{matrix} * & * & * \\ & * & * \\ & & * \end{matrix}} & & 0 \\ & & & & & \underline{A}_{[k]} & -R^v_{[k]} \\ & & & & & 0 & \underline{C}_{[k]} \end{bmatrix} \cdot \begin{bmatrix} x_{[0]} \\ x_{[1]} \\ x_{[2]} \\ \vdots \\ x_{[k-1]} \\ x_{[k]} \end{bmatrix}.$$

Furthermore, as we are only interested in an estimate for $x_{[k]}$, the rest of the matrix can be dropped, and we end up with a reduced set as follows

$$\begin{bmatrix} \boxed{\begin{matrix} * \\ * \\ * \end{matrix}} \\ -\underline{B}_{[k]}u_{[k]} \\ \underline{y}_{[k]} - \underline{D}_{[k]}u_{[k]} \end{bmatrix} \simeq \begin{bmatrix} \boxed{\begin{matrix} * & * & * \\ & * & * \\ & & * \end{matrix}} & 0 \\ \underline{A}_{[k]} & -R^v_{[k]} \\ 0 & \underline{C}_{[k]} \end{bmatrix} \cdot \begin{bmatrix} x_{[k-1]} \\ x_{[k]} \end{bmatrix}.$$

The framed triangular part that is propagated from the previous time step then equals the square root of the information matrix [19].

$$\begin{bmatrix} \boxed{R^p_{[k-1]}\hat{x}_{[k-1]}} \\ -\underline{B}_{[k]}u_{[k]} \\ \underline{y}_{[k]} - \underline{D}_{[k]}u_{[k]} \end{bmatrix} \simeq \begin{bmatrix} \boxed{R^p_{[k-1]}} & 0 \\ \underline{A}_{[k]} & -R^v_{[k]} \\ 0 & \underline{C}_{[k]} \end{bmatrix} \begin{bmatrix} x_{[k-1]} \\ x_{[k]} \end{bmatrix}$$

[19] inverse of the error covariance for the estimate $\hat{x}_{[k-1]}$.

where

$$E\{(x_{[k]} - \hat{x}_{[k]}) \cdot (x_{[k]} - \hat{x}_{[k]})^T\} = R^{p^{-1}}_{[k]} \cdot R^{p^{-T}}_{[k]}.$$

The update then reduces to first upper triangularizing the above matrix (QR factorization)

$$\begin{bmatrix} \\ \\ \\ \\ \\ \\ \vdots \end{bmatrix} \simeq \begin{bmatrix} * & * & * & * & * & * \\ & * & * & * & * & * \\ & & * & * & * & * \\ 0 & & & * & * & * \\ & & & & * & * \\ & & & & & * \\ 0 & & & 0 & & \end{bmatrix} \begin{bmatrix} x_{[k-1]} \\ x_{[k]} \end{bmatrix}$$

followed by a triangular backsolve in order to compute the estimate $\hat{x}_{[k]}$

$$\begin{bmatrix} \\ \\ \boxed{R^p_{[k]}\hat{x}_{[k]}} \\ \vdots \end{bmatrix} \simeq \begin{bmatrix} * & * & * & * & * & * \\ & * & * & * & * & * \\ & & * & * & * & * \\ 0 & & & \boxed{R^p_{[k]}} & & \\ & & & & & \\ 0 & & & 0 & & \end{bmatrix} \begin{bmatrix} x_{[k-1]} \\ \boxed{x_{[k]}} \end{bmatrix}.$$

Finally, the framed lower right block of the triangular matrix, together with the corresponding left-hand side is propagated to the next update.

A parallel algorithm for SRIF

From the previous section, it follows that the main part in computing an update consists in computing the upper triangular factor (QR factorization) of a structured matrix:

$$\begin{bmatrix} R^p_{[k-1]} & 0 \\ \underline{A}_{[k]} & -R^v_{[k]} \\ 0 & \underline{C}_{[k]} \end{bmatrix} = \begin{bmatrix} * & * & * & & & \\ & * & * & & & \\ & & * & & & \\ * & * & * & * & * & * \\ * & * & * & & * & * \\ * & * & * & & & * \\ & & & * & * & * \\ & & & & * & * \\ & & & & & * \end{bmatrix}.$$

The aim is to exploit the available structure as much as possible.

First of all, we rewrite the set of equation as follows, by simply permuting block rows and columns

$$\begin{bmatrix} -\underline{B}_{[k]}u_{[k]} \\ R^p_{[k-1]}\hat{x}_{[k-1]} \\ \underline{y}_{[k]} - \underline{D}_{[k]}u_{[k]} \end{bmatrix} \simeq \begin{bmatrix} -R^v_{[k]} & \underline{A}_{[k]} \\ 0 & R^p_{[k-1]} \\ \underline{C}_{[k]} & 0 \end{bmatrix} \begin{bmatrix} x_{[k]} \\ x_{[k-1]} \end{bmatrix}$$

so that the upper part in the above matrix becomes upper triangular:

$$\begin{bmatrix} -R^v_{[k]} & \underline{A}_{[k]} \\ 0 & R^p_{[k-1]} \end{bmatrix} = \begin{bmatrix} x & x & x & x & x & x \\ & x & x & x & x & x \\ & & x & x & x & x \\ & & & x & x & x \\ & & & & x & x \\ & & & & & x \end{bmatrix} \stackrel{def}{=} \mathbf{R}^{2n\times 2n}_{[k]}.$$

We can now apply a Jacobi-type procedure for QR factorization onto $\mathbf{R}_{[k]}$, which restores the original column ordering, while however the triangular structure is preserved throughout. A QR updating is then performed to incorporate the rows containing $\underline{C}_{[k]}$.

The above column and row permutations lead to a slightly revised problem statement that is somewhat more amenable to parallel implementation. Remember that the lower right block of the updated triangular matrix becomes the upper left block in the matrix for the next time step, at least in the original problem statement. In a parallel implementation, this results in cross-connections running all the way around the array. On the other hand, by introducing the permutations, the lower right block that is propagated to the next time step, can stay in that position –at least initially– so that cross-connections are avoided.

The Jacobi-type algorithm, applied to the initial triangular factor $\mathbf{R}_{[k]}$, consists in performing a sequence of permutations of adjacent columns in order to restore the original ordering, together with orthogonal row transformations in order to keep $\mathbf{R}_{[k]}$ upper triangular (see also [Luk 1986b]). A description of this algorithm is as follows

Initialization

$\mathbf{R} \leftarrow \mathbf{R}_{[k]}$

Loop
  for $j = 1, \ldots, n$
    for $i = n - j + 1, \ldots, 2n - j$
      compute transformations

$$\{\Pi_{[i,j]}\}_{i,i+1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\tan\theta = \frac{\mathbf{R}_{i+1,i+1}}{\mathbf{R}_{i,i+1}}$$

$$\{\Theta_{[i,j]}\}_{i,i+1} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

      apply transformations

$$\mathbf{R} \leftarrow \Theta_{[i,j]} \cdot \mathbf{R} \cdot \Pi_{[i,j]}$$

    end
  end

The following small example ($n = 3$) shows the effect of subsequent sequences of transformations. The first sequence of n transformations for $j = 1$ ($i = 3, 4, 5$) turns the original column configuration

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ & 2 & 3 & 4 & 5 & 6 \\ & & 3 & 4 & 5 & 6 \\ & & & 4 & 5 & 6 \\ & & & & 5 & 6 \\ & & & & & 6 \end{bmatrix}$$

into

$$\begin{bmatrix} 1 & 2 & 4 & 5 & 6 & 3 \\ & 2 & 4 & 5 & 6 & 3 \\ & & 4 & 5 & 6 & 3 \\ & & & 5 & 6 & 3 \\ & & & & 6 & 3 \\ & & & & & 3 \end{bmatrix}.$$

Subsequent sequences for $j = 2$ $(i = 2,3,4)$ and $j = 3$ $(i = 1,2,3)$ respectively result in

$$\begin{bmatrix} 1 & 4 & 5 & 6 & 2 & 3 \\ & 4 & 5 & 6 & 2 & 3 \\ & & 5 & 6 & 2 & 3 \\ & & & 6 & 2 & 3 \\ & & & & 2 & 3 \\ & & & & & 3 \end{bmatrix}$$

and

$$\begin{bmatrix} 4 & 5 & 6 & 1 & 2 & 3 \\ & 5 & 6 & 1 & 2 & 3 \\ & & 6 & 1 & 2 & 3 \\ & & & 1 & 2 & 3 \\ & & & & 2 & 3 \\ & & & & & 3 \end{bmatrix}.$$

Clearly, the overall result is a block permutation, as required in order to compensate for the original permutation. In conclusion, after applying the above procedure the set of equations in time step k reduces to

$$\begin{bmatrix} \\ \\ \\ \\ \\ \underline{y}_{[k]} - \underline{D}_{[k]}\underline{u}_{[k]} \end{bmatrix} \simeq \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \\ 0 & & & & & \underline{C}_{[k]} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{[k-1]} \\ \boldsymbol{x}_{[k]} \end{bmatrix}.$$

Note that row transformations in the Jacobi procedure should of course be applied to the left-hand side as well.

In a second step, a QR updating is performed, annihilating the rows containing the $\underline{C}_{[k]}$ matrix, and resulting in a triangular set

$$\begin{bmatrix} \\ \\ \\ \\ \\ \\ \vdots \end{bmatrix} \simeq \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \\ 0 & & & 0 & & \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{[k-1]} \\ \boldsymbol{x}_{[k]} \end{bmatrix}.$$

Finally, this set is solved for $\boldsymbol{x}_{[k]}$. In the next time step, the upper part of the triangular matrix is overwritten with $(-R^v_{[k+1]}, \underline{A}_{[k+1]})$, and the whole procedure starts over again.

A systolic array for SRIF

The re-organized SRIF algorithm is seen to be a combination of QR updating on the one hand, with a Jacobi-type algorithm on the other hand. Again, these can be combined on a systolic array.

Obviously we will have to provide for a $2n \times 2n$ triangular array to accomodate the $2n \times 2n$ matrix $\mathbf{R}$. It is by now well known that such an array performs pipelined sequences of $2n-1$ transformations, where

$$i = 1 \quad \ldots \quad 2n-2 \quad 2n-1 \,.$$

In our QR factorization algorithm however, the pivot index successively takes up the following values

$$i = \begin{array}{cccccccc} & & & & n & n+1 & \ldots & 2n-1 \\ & & & n-1 & n & n+1 & \ldots & 2n-2 \\ & & & & \vdots & & & \\ & & 2 & \ldots & n-1 & n & n+1 & \\ & 1 & 2 & \ldots & n-1 & n & & \end{array}$$

which indeed corresponds to n of the aforementioned sequences, although there are clearly a number of empty slots. In other words, our Jacobi-type algorithm is straightforwardly implemented on the array, albeit that a significant part of the applied transformations are trivial "nill operations". One can verify that the empty slots correspond to e.g 75% of the computations in the first sequence, 25% in the last sequence. The overall processor utilization then equals 50%, at least for this part of the computations.

The subsequent QR updating with the $(0, \underline{C}_{[k]})$ matrix, can be implemented on the same array (modified Gentleman-Kung array, Figure 3.7). A QR updating then corresponds to sequences of transformations, where

$$i = 1 \quad \ldots \quad 2n-1 \quad 2n \,.$$

The number of sequences equals the number of rows in $\underline{C}_{[k]}$, i.e. the number of outputs. However, due to the zeroes in $(0, \underline{C}_{[k]})$, again we will have empty slots as follows

$$i = \begin{array}{cccccc} \ldots & n+1 & n+2 & \ldots & 2n-1 & 2n \\ \ldots & n+1 & n+2 & \ldots & 2n-1 & 2n \\ & & \vdots & & & \\ \ldots & n+1 & n+2 & \ldots & 2n-1 & 2n. \end{array}$$

The processor utilization for this part of the computations therefore equals 25%.

Finally, as both the Jacobi procedure and the QR updating correspond to sequences of transformations where $i = 1, \ldots, 2n$ (with empty slots), they can be pipelined on the array. The overall efficiency then equals

$$\frac{n}{n+l} \cdot 50\% + \frac{l}{n+l} \cdot 25\%.$$

As $l \leq n$, the processor utilization ranges from 37.5% for $l = n$, up to 50% for $l \ll n$.

Finally, a few remarks on the described array

1. *Note that –on the main diagonal– the computations for $i = x$ in one sequence of transformations are computed in parallel with the computations for $i = x+2$ in the previous sequence, etc. One easily checks that by the time the first QR update starts off for $i = n+1$, the execution of the Jacobi algorithm is completely finished as far as the main diagonal is concerned [20]. The permutations of the Jacobi scheme therefore (fortunately) do not interfere with the QR updates, so that the new rows $(0, \underline{C}_{[k]})$ can be fed in as such, without "pre-permutations".*
2. *The left-hand sides of the equations in the previous section are stored in an additional column to the right of the triarray. Row transformations obviously have to be applied to these elements as well.*

[20] as $i = n + 3$ is an empty slot in the last sequence of the Jacobi algorithm, etc.

3. *As for moving data in and out of the array, one can efficiently make use of the empty slots in the Jacobi procedure. Clearly the leading empty slots*

$$i = \begin{array}{ccccc} 1 & 2 & \ldots & n-1 \\ & \vdots & & \\ 1 & 2 & & \\ 1. & & & \end{array}$$

can be used for downloading $(-\underline{R}^v_{[k+1]}, \underline{A}_{[k+1]})$ already. The trailing empty slots on the other hand can be used for pushing $\underline{R}^p_{[k-1]}$ out of the array, into a linear array for computing the back substitution. Alternatively, they can be used to perform the back substitution on the triarray itself. For the sake of conciseness, we will not go into the details of these technicalities.

5.5 Conclusion

By means of a few examples, we have shown how various combined algorithms can be pieced together. Use is made of Jacobi-type algorithms mainly, plus additional matrix operations such as triangular updates, matrix-vector multiplications, etc. The combined Jacobi approach results in regularly structured algorithms, that are efficiently implemented on systolic arrays. It should be stressed that always the same type of systolic array is used for implementing each one of these algorithms. We believe that the above examples are only a few representatives out of a huge collection of similar algorithms. Everyone can go out shooting.

Conclusions and open problems

In this thesis, we have focussed on efficient algorithms for signal processing, systems identification and control. We started from one acute problem in present-day digital signal processing : the computational complexity of SVD based algorithms, particularly in view of possible real time implementation. Unlike e.g. QR updating, SVD updating is hardly used in practice. Computing an SVD requires $O\{n^3\}$ operations, whereas a QR update takes only $O\{n^2\}$ operations, hence the difficulty. First of all, we developed a **novel SVD updating algorithm**, *with a computational complexity which is reduced from $O\{n^3\}$ to $O\{n^2\}$. In each time step an approximate SVD is computed from a previous approximation, hence the computational saving. QR updating and SVD updating are then much alike. SVD updating in fact boils down to QR updating, supplemented with a Jacobi-type SVD algorithm applied to the triangular factor. Such an SVD updating can for instance be used in recursive total least squares computations, much like QR updating is used for recursive least squares computations. We also showed how it can be used for recursive systems identification, where we developed a new SVD based state space identification procedure.*

Second, we have focussed on the **parallel implementation** *of these algorithms, namely on systolic arrays. Currently, much research is devoted to the use of such processor arrays for high throughput applications like for instance beamforming. For QR based algorithms, one can then make use of the well known Gentleman-Kung QR updating array. SVD updating had not really been looked at in this respect. We developed a new type of systolic array that implements the combined SVD updating algorithm. Similar to the QR updating array, it has $O\{n^2\}$ processors and the throughput is $O\{n^0\}$, which means that new data vectors can be fed in at a rate that is independent*

of the problem size.

Finally –but maybe most interestingly– we have shown how the approach to developing the SVD updating algorithm/array can also be used for developing **similar (updating) algorithms/arrays** *within the broad field of signal processing and systems and control. Jacobi-type algorithms are seen to play a crucial role here. Apart from the extensions for PSVD and QSVD updating –problems which mostly had not been solved before– we also developed algorithms and systolic arrays for Kalman filtering, for recursive least squares computations with inverse factorization, and for solving Riccati and Lyapunov equations. These examples are only a few representatives out of a whole collection of similar (combined Jacobi-type) algorithms.*

Further research *can therefore obviously be devoted to a further exploration of the field, and developing similar algorithms for various other problems. More particularly one can for instance think of H_∞ control theory. As all of the underlying problems and solutions are being rethought in terms of state space models, the need for efficient matrix algorithms will obviously be huge. Optimization –as another example– is closely related to this and equally well gives rise to interesting matrix problems. Parallel algorithms are a hot research topic here. Briefly, there is still a lot of work to be done on (parallel) algorithms development, and we believe our approach –at least in some cases– might lead to efficient solutions.*

On the other hand, there is also still much work to be done on implementational issues. Taking into account the limitations of present-day VLSI technology, an ASIC implementation of for instance the SVD updating algorithm is more or less within reach, while a systolic implementation might be far off yet. Whether the small grain parallelism of a systolic array gives useful indications for a higher level partitioning, is still an open problem. Also, as fixed point arithmetic requires less hardware, fixed point algorithms are preferable to floating point algorithms. Issues like wordlength optimization and such are thus in order. This research will be carried out within the framework of the NANA-project, ESPRIT Basic Research Action 3280 (Novel algorithms for new real time VLSI architectures).

Finally, we would like to briefly mention one more subject for possible further research, somewhat more out-of-the-way but by no means less challenging. In general, on-line algorithms like the SVD updating algorithm are expected to be stable over a possible infinite number of updates. Error build-up is avoided by including error correction schemes. We developed two such

schemes, one for re-orthogonalizing an approximately orthogonal matrix in the SVD updating scheme (Chapter 2), and one for stabilizing recursive least squares computations which exploit inverse factorization (Chapter 5). Error build-up is explicitly corrected here. During the preparation of this thesis, we also came across an alternative solution for the first problem[1]*. It appears that if the columns of the orthogonal V-matrix are re-normalized from time to time, then the orthogonal transformations of the SVD updating scheme automatically restore the orthogonality (and one can easily prove this, cfr. [Moonen et al. 1990g]). The algorithm thus becomes self-correcting, in other words stable. Finally, let us only mention that recently several people have come up with error feedback schemes as a means of stabilizing adaptive algorithms. Briefly several concepts from systems theory and control seem to show up in algorithms design. A general theory is lacking, but definitely within reach. The research encompasses linear algebra and numerical analysis, as well as systems and control theory, and beautifully captures the spirit of combining different fields.*

[1] which we preferred not to include in the text, as both its numerical analysis and systolic implementation are much more involved.

Appendices

A : proof second tracking error bound, page 61

Let

$$\begin{aligned} R_{[k]} &= \begin{bmatrix} R_{\mathrm{n}} & R_{\mathrm{sn}} \\ 0 & R_{\mathrm{s}} \end{bmatrix} \\ V_{[k]} &= \begin{bmatrix} V_{\mathrm{n}} & V_{\mathrm{s}} \end{bmatrix}. \end{aligned}$$

Time indices are omitted for brevity For the symmetric matrix

$$R_{[k]}^t \cdot R_{[k]} = \begin{bmatrix} R_{\mathrm{n}}^t \cdot R_{\mathrm{n}} & R_{\mathrm{n}}^t \cdot R_{\mathrm{sn}} \\ R_{\mathrm{sn}}^t \cdot R_{\mathrm{n}} & R_{\mathrm{s}}^t \cdot R_{\mathrm{s}} + R_{\mathrm{sn}}^t \cdot R_{\mathrm{sn}} \end{bmatrix}$$

one can compute modified values $\tilde{\delta}$ for the gap

$$\begin{aligned} \tilde{\delta} &= \sigma_{\min}\{R_{\mathrm{s}}^t \cdot R_{\mathrm{s}} + R_{\mathrm{sn}}^t \cdot R_{\mathrm{sn}}\} - \sigma_{\max}\{R_{\mathrm{n}}^t \cdot R_{\mathrm{n}}\} \\ &\geq \sigma_{\min}\{R_{\mathrm{s}}^t \cdot R_{\mathrm{s}}\} - \sigma_{\max}\{R_{\mathrm{n}}^t \cdot R_{\mathrm{n}}\} \\ &\geq \sigma_{\min}\{R_{\mathrm{s}}\} \cdot (\sigma_{\min}\{R_{\mathrm{s}}\} - \sigma_{\max}\{R_{\mathrm{n}}\} \cdot \frac{\sigma_{\max}\{R_{\mathrm{n}}\}}{\sigma_{\min}\{R_{\mathrm{s}}\}}) \\ &\geq \sigma_{\min}\{R_{\mathrm{s}}\} \cdot (\sigma_{\min}\{R_{\mathrm{s}}\} - \sigma_{\max}\{R_{\mathrm{n}}\}) \\ &\geq \sigma_{\min}\{R_{\mathrm{s}}\} \cdot \delta \end{aligned}$$

and $\tilde{\varepsilon}$ for the norm of the cross terms

$$\begin{aligned} \tilde{\varepsilon} &= \|R_{\mathrm{n}}^t \cdot R_{\mathrm{sn}}\|_F \\ &\leq \|R_{\mathrm{sn}}\|_F \cdot \|R_{\mathrm{n}}\|_2 \\ &\leq \varepsilon \cdot \sigma_{\max}\{R_{\mathrm{n}}\}. \end{aligned}$$

By making use of these, one ends up with a sharper bound for the subspace error.

$$\begin{aligned} TE_d^{\{k\}} &< 2 \cdot \frac{\tilde{\varepsilon}}{\tilde{\delta}} \\ &< 2 \cdot \frac{\varepsilon}{\delta} \cdot (\frac{\sigma_{\max}\{R_{\mathrm{n}}\}}{\sigma_{\min}\{R_{\mathrm{s}}\}}). \end{aligned}$$

B: cross-term reduction Kogbetliantz's algorithm, page 62

Let us assume that the initial configuration is as follows –we consider a small 6×6 example, from which the results for the general case obviously follow–

$$\begin{aligned} R &= \begin{bmatrix} R_{\mathrm{s}} & R_{\mathrm{sn}} \\ 0 & R_{\mathrm{n}} \end{bmatrix} \\ &= \begin{bmatrix} r_{11}^{\mathrm{s}} & r_{12}^{\mathrm{s}} & r_{13}^{\mathrm{s}} & \boxed{\varepsilon_{14}} & \boxed{\varepsilon_{15}} & \boxed{\varepsilon_{16}} \\ & r_{22}^{\mathrm{s}} & r_{23}^{\mathrm{s}} & \boxed{\varepsilon_{24}} & \boxed{\varepsilon_{25}} & \boxed{\varepsilon_{26}} \\ & & r_{33}^{\mathrm{s}} & \boxed{\varepsilon_{34}} & \boxed{\varepsilon_{35}} & \boxed{\varepsilon_{36}} \\ & & & r_{44}^{\mathrm{n}} & r_{45}^{\mathrm{n}} & r_{46}^{\mathrm{n}} \\ & & & & r_{55}^{\mathrm{n}} & r_{56}^{\mathrm{n}} \\ & & & & & r_{66}^{\mathrm{n}} \end{bmatrix}. \end{aligned}$$

Time indices are omitted for brevity. We assume that

$$\varepsilon \ll \delta$$

where

$$\varepsilon = \|R_{\mathrm{sn}}\|_F$$

is the Frobenius norm of the matrix with cross terms, and

$$\delta = \sigma_{\min}\{R_{\mathrm{s}}\} - \sigma_{\max}\{R_{\mathrm{n}}\}$$

is the gap between the singular values of R_{s} and R_{n}. As these are then known to be $\frac{\varepsilon^2}{\delta}$ -close to the singular values of R [Charlier & Van Dooren 1987], we end up with

$$\delta \simeq \sigma_{\min}\{R_{\mathrm{s}}\} \qquad \text{for } SN \gg 1.$$

The aim is to investigate the effect on ε of one cyclic-by-rows sweep in Kogbetliantz's SVD algorithm, modified for triangular matrices. This essentially consists of a number of 2×2 SVD's on the main diagonal, where the pivot index takes up the values (see [Stewart 1985, Luk 1986a] for details)

$$\begin{aligned} i \;=\; & 1,2,3,4,5 \\ & 1,2,3,4 \\ & 1,2,3 \\ & 1,2 \\ & 1. \end{aligned}$$

As a reminder, each 2×2 SVD can be described as [1]

$$\underbrace{\begin{bmatrix} r_{i,i} & 0 \\ 0 & r_{i+1,i+1} \end{bmatrix}}_{\{R\}_{i,i+1}} \leftarrow \underbrace{\begin{bmatrix} -\sin\theta & \cos\theta \\ \cos\theta & \sin\theta \end{bmatrix}}_{\{\Pi\{i,i+1\}G\{i,i+1,\theta\}\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} r_{ii} & r_{i,i+1} \\ 0 & r_{i+1,i+1} \end{bmatrix}}_{\{R\}_{i,i+1}} \cdot \underbrace{\begin{bmatrix} -\sin\phi & \cos\phi \\ \cos\phi & \sin\phi \end{bmatrix}}_{\{G\{i,i+1,\phi\}^t\Pi\{i,i+1\}^t\}_{i,i+1}}$$

where

$$\begin{aligned} \tan 2\theta \;&=\; \frac{-2r_{i+1,i+1}\cdot r_{i,i+1}}{r_{i,i}^2 - r_{i+1,i+1}^2 + r_{i,i+1}^2} \\ \tan\phi \;&=\; \frac{+r_{i+1,i+1}\cdot\tan\theta - r_{i,i+1}}{r_{i,i}}. \end{aligned}$$

First of all, we can slightly re-order the 2×2 transformations as follows

$$\textit{Part a} \quad \begin{cases} i \;=\; 1,2 \\ \qquad\; 1 \end{cases}$$

$$\textit{Part b} \quad \begin{cases} i \;=\; .,.,3,4,5 \\ \qquad\; .,2,3,4 \\ \qquad\; 1,2,3 \end{cases}$$

$$\textit{Part c} \quad \begin{cases} i \;=\; 1,2 \\ \qquad\; 1 \end{cases}.$$

–Part a corresponds to rotations within the (approximate) signal subspace, reducing the off-norm in R_S. Both ε and δ remain unchanged, so that for the time being these transformations are irrelevant.

[1] for the sake of clarity, we prefer to use inner rotations + permutations, instead of outer rotations.

–Part b corresponds to annihilations of cross-terms, and needs further investigation. Referring to the initial configuration –which is basically not changed in Part a– let us first remark that the diagonal entries in R_S satisfy

$$r_{ii}^S \geq \sigma_{\min}\{R_S\}$$

–as R_S is triangular– while on the other hand the diagonal entries in R_N obviously satisfy

$$r_{ii}^N \leq \sigma_{\max}\{R_N\}.$$

One easily verifies that the above upper and lower bounds remain valid throughout the computations in Part b (r_{ii}^S -elements always increase, r_{ii}^N -elements decrease)

The first series of transformations, where the pivot index takes up the values

$$i = .,.,3,4,5$$

turns the initial configuration into [2]

$$\begin{bmatrix} r_{11}^S & r_{12}^S & \boxed{\varepsilon_{13}} & \boxed{\varepsilon_{14}} & \boxed{\varepsilon_{15}} & r_{16}^S \\ & r_{22}^S & \boxed{\varepsilon_{23}} & \boxed{\varepsilon_{24}} & \boxed{\varepsilon_{25}} & r_{26}^S \\ & & r_{33}^N & r_{34}^N & r_{35}^N & \boxed{\varepsilon_{36}^\star} \\ & & & r_{44}^N & r_{45}^N & \boxed{\varepsilon_{46}^\star} \\ & & & & r_{55}^N & \boxed{\varepsilon_{56}^\star} \\ & & & & & r_{66}^S \end{bmatrix}.$$

When $i = 3$, the pivot element ε_{34} is ε-small, from which we can estimate the rotation angles as follows

$$\begin{aligned} \sin\theta \;\simeq\; \tan\theta \;&\simeq\; \frac{\sigma_{\max}\{R_N\}}{\sigma_{\min}^2\{R_S\}}\cdot O\{\varepsilon\} \\ \sin\phi \;\simeq\; \tan\phi \;&\simeq\; \frac{1}{\sigma_{\min}\{R_S\}}\cdot O\{\varepsilon\}. \end{aligned}$$

[2] iteration indices are left out for the sake of clarity, subscripts ij refer to row and column numberings in the full R-matrix.

From the fact that θ (row transformation) is particularly small, it follows that the pivot for $i = 4$ is ε-small as well

$$\begin{aligned} \varepsilon_{35} \cdot \cos\theta - r^{\mathrm{n}}_{45} \cdot \sin\theta &\simeq O\{\varepsilon\} + \sigma_{\max}\{R_{\mathrm{n}}\} \cdot \frac{\sigma_{\max}\{R_{\mathrm{n}}\}}{\sigma^2_{\min}\{R_{\mathrm{s}}\}} \cdot O\{\varepsilon\} \\ &\simeq O\{\varepsilon\}. \end{aligned}$$

Finally, from a similar reasoning it follows that the pivot for $i = 5$ is also ε-small. The estimates for $\tan\theta$ and $\tan\theta$ therefore hold for $i = 3$ as well as for $i = 4, 5$.

From the estimates for the rotation angles, we can estimate the magnitude of r^{n}_{ij}, $\varepsilon^{\star}_{ij}$, ε_{ij} after the the first series of transformations:

elements r^{n}_{ij} remain $\sigma_{\max}\{R_{\mathrm{n}}\}$ -small, which follows from

$$\begin{aligned} r^{\mathrm{n}}_{ij} \cdot \cos\theta + \varepsilon^{\mathrm{n}}_{kj} \cdot \sin\theta &\simeq O\{\sigma_{\max}\{R_{\mathrm{n}}\}\} + \frac{\sigma_{\max}\{R_{\mathrm{n}}\}}{\sigma^2_{\min}\{R_{\mathrm{s}}\}} \cdot O\{\varepsilon^2\} \\ &\simeq O\{\sigma_{\max}\{R_{\mathrm{n}}\}\} \end{aligned}$$

cross terms $\varepsilon^{\star}_{ij}$ are $\frac{\varepsilon}{SN}$ -small, which follows from

$$\begin{aligned} \varepsilon^{\star}_{ij} &\simeq \sum_k r^{\mathrm{n}}_{ik} \cdot \sin\phi \\ &\simeq \frac{\sigma_{\max}\{R_{\mathrm{n}}\}}{\sigma_{\min}\{R_{\mathrm{s}}\}} \cdot O\{\varepsilon\} \\ &\simeq O\{\frac{\varepsilon}{SN}\} \end{aligned}$$

cross terms ε_{ij} remain ε-small, which follows from

$$\begin{aligned} \varepsilon_{ij} &\simeq \varepsilon_{ij} + \sum_k r^{\mathrm{s}}_{ik} \cdot \sin\phi \\ &\simeq O\{\varepsilon\} + \frac{O\{\|R_{\mathrm{s}}\|_{\mathrm{off}}\}}{\sigma_{\min}\{R_{\mathrm{s}}\}} \cdot O\{\varepsilon\} \\ &\simeq O\{\varepsilon\} \end{aligned}$$

where we made an opportunist (but mostly fair) assumption, namely that $r^{\mathrm{s}}_{ik} = O\{\|R_{\mathrm{s}}\|_{\mathrm{off}}\} \leq \sigma_{\min}\{R_{\mathrm{s}}\}$. This corresponds to a certain degree of convergence within R_{s}, brought about by Part a.

We can now repeat this reasoning for the second and the third series of transformations in Part b

$$\begin{aligned} i &= ., 2, 3, 4 \\ &\quad 1, 2, 3 \end{aligned}$$

in turn transforming the triangular factor into

$$\begin{bmatrix} r^{\mathrm{s}}_{11} & \boxed{\varepsilon_{12}} & \boxed{\varepsilon_{13}} & \boxed{\varepsilon_{14}} & r^{\mathrm{s}}_{15} & r^{\mathrm{s}}_{16} \\ & r^{\mathrm{n}}_{22} & r^{\mathrm{n}}_{23} & r^{\mathrm{n}}_{24} & \boxed{\varepsilon^{\star}_{25}} & \boxed{\varepsilon^{\star}_{26}} \\ & & r^{\mathrm{n}}_{33} & r^{\mathrm{n}}_{34} & \boxed{\varepsilon^{\star}_{35}} & \boxed{\varepsilon^{\star}_{36}} \\ & & & r^{\mathrm{n}}_{44} & \boxed{\varepsilon^{\star}_{45}} & \boxed{\varepsilon^{\star}_{46}} \\ & & & & r^{\mathrm{s}}_{55} & r^{\mathrm{s}}_{56} \\ & & & & & r^{\mathrm{s}}_{66} \end{bmatrix}$$

and

$$\begin{bmatrix} r^{\mathrm{n}}_{11} & r^{\mathrm{n}}_{12} & r^{\mathrm{n}}_{13} & \boxed{\varepsilon^{\star}_{14}} & \boxed{\varepsilon^{\star}_{15}} & \boxed{\varepsilon^{\star}_{16}} \\ & r^{\mathrm{n}}_{22} & r^{\mathrm{n}}_{23} & \boxed{\varepsilon^{\star}_{24}} & \boxed{\varepsilon^{\star}_{25}} & \boxed{\varepsilon^{\star}_{26}} \\ & & r^{\mathrm{n}}_{33} & \boxed{\varepsilon^{\star}_{34}} & \boxed{\varepsilon^{\star}_{35}} & \boxed{\varepsilon^{\star}_{36}} \\ & & & r^{\mathrm{s}}_{44} & r^{\mathrm{s}}_{45} & r^{\mathrm{s}}_{46} \\ & & & & r^{\mathrm{s}}_{55} & r^{\mathrm{s}}_{56} \\ & & & & & r^{\mathrm{s}}_{66} \end{bmatrix}$$

where all cross terms $\varepsilon^{\star}_{ij}$ are seen to remain $\frac{\varepsilon}{SN}$ -small.

–Finally, subsequent transformations in Part c do not alter the norm of the submatrix with cross terms.

As a main conclusion, we can state that the matrix with cross terms is $\frac{\varepsilon}{SN}$ -small after the forward sweep. The backward sweep, returning the triangular matrix to the original configuration, again reduces this norm by a factor $\frac{1}{SN}$ (where this time the column transformations are particularly small). A double sweep thus corresponds to a reduction by a factor $(\frac{1}{SN})^2$.

References

H.C. Andrews and C.L. Patterson 1976a. Singular value decomposition and digital image processing. *IEEE Trans.* **ASSP-24**, pp 26-53.

H.C. Andrews and C.L. Patterson 1976b. SVD image coding. *IEEE Trans.* **COM-24**, pp 424-432.

P. Arbenz and G.H. Golub 1988. On the spectral decomposition of hermitian matrices modified by low rank perturbations with applications. *SIAM J. Matrix Anal. Appl.* **9**, No. 1, pp 40-58.

K.S. Arun and B.D. Rao 1988. An improved Toeplitz approximation method. *Proc. ICASSP-1988, New York*, pp 2352-2355.

K.S. Arun 1989. A unitarily constrained total least squares problem in signal processing. *Technical Report, Dept. Electr. and Computer Eng., Univ. of Illinois, Urbana-Champaign, IL.*

Z. Bai 1987. The direct GSVD algorithm and its parallel implementation. *Comp. Sc. Technical Report series, Univ. of Maryland,* **CS-TR-1901**.

J.L. Barlow and I.C.F. Ipsen 1987. Scaled Givens rotations for the solution of linear least squares problems on systolic arrays. *SIAM J. Sci. Stat. Comput.* **8**, No. 5, pp 716-733.

R.Bartels and G.Stewart 1972. Solution of the matrix equation $AX + XB = C$. Comm. ACM **15**, pp 820-826.

Biglieri and Yao 1989. Some properties of the SVD and their application to digital signal processing. *Signal Processing* **18**, pp 277-289.

Å. Björck 1987. Least squares methods. *Dept. of Math., Linköping Univ.* **S-581 83**.

Å. Björck and G.H. Golub 1973. Numerical methods for computing angles between linear subspace. *Math. Comp.* **27**, pp 579-594.

A.W. Bojanczyk, R.P. Brent and F.R. De Hoog 1986. QR factorization of Toeplitz matrices. *Numer. Math.* **49**, pp 81-94.

A.W. Bojanczyk, R.P. Brent, P. Van Dooren and F.R. De Hoog 1987. A Note on downdating the Cholesky factorization. *SIAM J. Sci. Stat. Comput.* **8**, No. 3, pp 210-221.

R.P. Brent, F.T. Luk and C.F. Van Loan 1982. Computation of the singular value decomposition using mesh-connected processors. *Technical report Australian Nat. Univ., TR-CS-83-05.*

R.P. Brent and F.T. Luk 1985. The solution of singular value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM J. Sci. Stat. Comput.* **6**, No. 1, 69-84.

J.R. Bunch, C.P. Nielsen and D.C. Sorensen 1978. Rank one modification of the symmetric eigenproblem. *Numer. Math.* **31**, pp 31-48.

J.R. Bunch and C.P. Nielsen 1978. Updating the singular value decomposition. *Numer. Math.* **31**, pp 111-129.

P.A. Businger 1970. Updating the singular value decomposition. *BIT* **10**, pp 376-397.

R. Byers 1987. Solving the algebraic Riccati equation with the matrix sign function. *Lin. Alg. Appl.* **85**, pp 267-279.

D. Callaerts 1989. *Signal separation methods based on SVD and their application to the real-time extraction of the fetal electrocardiogram from cutaneous recordings.* PhD Thesis Kath. Univ. Leuven, Dept. El. Eng.

T.F. Chan 1982. An improved algorithm for computing the singular value decomposition. *ACM Trans. Math. Soft.* **3**, pp 72-83.

T.F. Chan 1987. Rank revealing QR factorizations. *Lin. Alg. Appl.* **88/89**, pp 67-82.

T.F. Chan and PC. Hansen 1990. Some applications of the rank revealing QR factorizations. *Technical Report, Dept. of Math., Univ. of California, Los Angeles, CA.*

J.-P. Charlier and P. Van Dooren 1987. On Kogbetliantz's SVD algorithm in the presence of clusters. *Lin. Alg. Appl.* **95**, pp 135-160.

J.-P. Charlier, V. Vanbegin, P. Van Dooren 1988. *On efficient implementation of Kogbetliantz's algorithm for computing the singular value decomposition. Numer. Math.* **52**, *pp 279-300.*

J.-P. Charlier and P. Van Dooren 1989a. *A Jacobi-like algorithm for computing the generalized Schur form of a regular pencil. J. Comp. Appl. Math.* **27**, *pp 17-36.*

J.-P. Charlier and P. Van Dooren 1989b. *A systolic algorithm for Riccati and Lyapunov equations. MCSS* **2**, *pp 109-136.*

J. Chun, T. Kailath and H. Lev-Ari 1987. *Fast parallel algorithms for QR and triangular factorization. SIAM J. Sci. Stat. Comput.* **8**, *No. 6, pp 899-913.*

P. Comon and G.H. Golub 1989. *Tracking a few extreme singular values and vectors in signal processing. Numerical Analysis Project. Manuscript* **NA-89-01**. *Dept. of Comp. Sc., Stanford Univ.*

J.J. Cuppen 1981. *A divide and conquer method for the symmetric tridiagonal eigenproblem. Numer. Math.* **31**, *pp 31-48.*

G. Cybenko 1987. *Fast Toeplitz orthogonalization using inner products. SIAM J. Sci. Stat. Comput.* **8**, *No. 5, pp 734-740.*

E. Deprettere (editor) 1988. *SVD and signal processing : algorithms, applications and architectures. Elsevier Science Publishers, North Holland.*

R.D. DeGroat and R.A. Roberts 1988. *A family of rank-one subspace updating methods. In E. Deprettere (Ed.), Singular value decomposition and signal processing, North-Holland, Amsterdam. pp 277-300.*

R.D. DeGroat and R.A. Roberts 1990. *Efficient, numerically stabilized rank-one eigenstructure updating. IEEE Trans.* **ASSP-38**, *pp 301-316.*

B. De Moor and J. Vandewalle 1987. *An adaptive SVD algorithm based on generalized Chebyshev recursion. In Mathematics in Signal Processing. (T.S. Durrani, J.B. Abbiss, J.E. Hudson, R.W. Madan, J.G. McWhirter and T.A. Moore, Eds.) Clarendon Press, Oxford. pp 607-635.*

B. De Moor, J. Vandewalle and J. Staar 1988. *Oriented energy and oriented signal-to-signal ratio concepts in the analysis of vector sequences and time series. in E. Deprettere (Ed.), Singular value decomposition and signal processing, North-Holland, Amsterdam. pp 209-232.*

B. De Moor 1988. *Mathematical Concepts and techniques for modelling static and dynamic systems. PhD Thesis Kath. Univ. Leuven, Dept. El. Eng.*

B. De Moor 1989. *On the structure and geometry of the product singular value decomposition. Numerical Analysis Project. Manuscript* **NA-89-05**. *Dept. of Comp. Sc., Stanford Univ.*

B. De Moor and G.H. Golub 1989a. *Generalized Singular Value Decompositions: A proposal for a standardized nomenclature. Numerical Analysis Project. Manuscript* **NA-89-04**. *Dept. of Comp. Sc., Stanford Univ.*

B. De Moor and G.H. Golub 1989b. *The Restricted SVD : properties and applications. Numerical Analysis Project. Manuscript* **NA-89-03**. *Dept. of Comp. Sc., Stanford Univ.*

B. De Moor and S. Boyd 1989. *Analytical properties of singular values and vectors. ESAT-SISTA Report* **1989-28**. *Dept. El. Eng., Kath. Univ. Leuven.*

J.J. Dongarra, J.R. Bunch, C.B. Moler and G.W. Stewart 1979. *LINPACK user guide. SIAM publications, Philidelphia.*

J.J. Dongarra and D.C. Sorensen 1987. *A fully parallel algorithm for the symmetric eigenvalue problem. SIAM J. Sci. Stat. Comput.* **8**, *No. 2, pp s139-s154.*

D.B. Duncan and S.D Horn 1972. *Linear dynamic recursive estimation from the viewpoint of regression analysis. J. Amer. Statist. Assoc.* **67**, *pp 815-821.*

C. Eckart and G. Young 1936. *The approximation of one matrix by another of lower rank. Psychometrika* **1**, *pp 211-218.*

K.V. Fernando and S.J. Hammarling 1987. *A product induced singular value decomposition for two matrices and balanced realisation. NAG Technical Report* **TR-8/87**.

K.V. Fernando 1987. The Kalman reachability/observability canonical form and the ΠSVD. *NAG Technical Report* **TR9/87**.

K.V. Fernando 1989. Linear convergence of the row cyclic Jacobi and Kogbetliantz methods. *Numer. Math.* **56**, No. 1, pp 73-92.

W. Ferzali and J.G. Proakis 1990. Adaptive SVD algorithm for covariance eigenstructure computation. *Proc. ICASSP-90.* In Press.

G.E. Forsythe and P. Henrici 1960. The cyclic Jacobi method for computing the principal values of a complex matrix. *Trans. Am. Math. Soc.* **94**, pp 1-23.

B.S. Garbow, J.M. Boyle, J.J. Dongarra and C.B. Moler 1972. *Matrix eigensystem routines, EISPACK guide extension*, Springer Verlag, New York.

J. Gardiner and A. Laub 1986. A generalization of the matrix sign function solution for algebraic Riccati equations. *Int. J. Control* **44**, pp 823-832.

F. Gaston and G. Irwin 1989. Systolic approach to square root information Kalman filtering. *Int. J. Control* **50**, No. 1, pp 225-248.

W.M. Gentleman 1973. Least squares computations by Givens transformations without square roots. *J. Inst. Math. Applics.*, **12**, pp 329-336.

W.M. Gentleman 1975. Error analysis of QR decompositions by Givens transformations. *Lin. Alg. Appl.*, **10**, pp 189-197.

W.M Gentleman and H.T. Kung 1981. Matrix triangularization by systolic arrays. *Real-Time Signal Processing IV, Proc. SPIE, Vol.* **298**, pp 19-26.

P.E. Gill, G.H. Golub, W. Murray, M.A. Saunders 1974. Methods for modifying matrix factorizations. *Math. Comp.* **28**, No. 126, pp 505-535.

Givens 1958. Computation of plane unitary rotations transforming a general matrix to triangular form. *SIAM J. App. Math.* **6**, pp 26-50.

G.H. Golub 1965. Numerical methods for solving linear least squares problems. *Numer. Math.* **7**, pp 206-216.

G.H. Golub 1973. Some modified eigenvalue problems. *SIAM Review* **15**, No. 2, pp 318-334.

G.H. Golub and W. Kahan 1965. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Numer. Anal.* **2**, pp 205-224.

G.H. Golub, S. Nash and C. Van Loan 1979. A Hessenberg-Schur method for the problem $AX + XB = C$. *IEEE Trans. Aut. Control* **AC-24**, pp 909-913.

G.H. Golub and C. Reinsch 1970. Singular value decomposition and least squares solution. *Numer. Math.* **14**, pp 403-420.

G.H. Golub and C.F. Van Loan 1980. An analysis of the total least squares problem. *SIAM J. Numer. Anal.*, **17**, No. 6, pp 883-893.

G.H. Golub, F.T. Luk and M. Overton 1981. A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix. *ACM Trans. Math. Soft.* **7**, pp 149-169.

G.H. Golub en C.F. Van Loan 1983. *Matrix computations*. North Oxford academic Publishing Co., Johns Hopkins Press.

G.H. Golub en C.F. Van Loan 1988. *Matrix computations*. Second Edition. North Oxford academic Publishing Co., Johns Hopkins Press.

S. Hammarling 1974. A note on modifications to the Givens plane rotations. *J. Inst. Maths Applics.* **13**, pp 215-218.

S. Hammarling 1982. Numerical solution of the stable non-negative definite Lyapunov equation. *IMA J. Numer. Anal.*, **2**, pp 303-323.

S. Hammarling 1985. The singular value decomposition in multivariate statistics. *AGM SIGNUM Newsletter* **20**, No. 3, pp 2-25.

P.C. Hansen 1984. SVD – theory and applications. *Report* **NI-84-05** Techn. Univ. Denmark.

V. Hari and Veselić 1987. On Jacobi methods for singular value decompositions. *SIAM. J. Sci. Stat. Comput.* **8**, No. 5, pp 741-754.

M.T. Heath, A.J. Laub, C.C. Paige and R.C. Ward 1986. Computing the singular value decomposition of a product of two matrices. *SIAM J. Sci. Stat. Comput.* **7**, No. 4, pp 1147-1159.

P. Henrici 1958. On the speed of convergence of cyclic and quasicyclic Jacobi methods for computing eigenvalues of Hermitian matrices. *J. Soc. Indust. Appl. Math.* **6**, No. 2, pp 144-162.

P. Henrici and K. Zimmermann 1968. An estimate for the norms of certain cyclic Jacobi operators. *Lin. Alg. Appl.* **1**, pp 489-501.

M.R. Hestenes 1958. Inversion of matrices by biorthogonalization and related results. *J. Soc. Indust. Appl. Math.* **6**, pp 51-90.

J. Hudson, T.J. Shepherd, J.G. McWhirter 1989. Parallel weight extraction by a systolic least squares algorithm. *Advanced Algorithms and Architectures for Signal Processing IV, Proc. SPIE, Vol.* **1152**.

B.L. Ho and R.E. Kalman 1966. Effective construction of linear state variable models from input output functions. *Regelungstechnik* **14**, pp 545-548.

H. Hotelling 1936. Relations between two sets of variates. *Biometrika* **28**, pp 321-377.

A.S. Householder 1958. Unitary triangularization of a nonsymmetric matrix. *J. Assoc. Comp. Mach.* **5**, pp 339-342.

C.G.J. Jacobi 1846. Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommended Gleichungen numerisch aufzulösen. *J. Reine Angew. Math.* **30**, pp 51-94.

J.M. Jover and T. Kailath 1986. A parallel architecture for Kalman filter measurement update and parameter estimation. *Automatica* **22**, No. 1, pp 43-57.

T. Kailath 1980. Linear system theory. *Prentice-Hall, New Jersey.*

R.E. Kalman 1982. System identification from noisy data. In *Dynamical Systems II*, A.R.Bednarek and L.Cesari editors, Academic Press.

I. Karasalo 1986. Estimating the covariance matrix by signal subspace avaraging. *IEEE Trans.* **ASSP-43**, pp 8-12.

V.C. Klema and A.J. Laub 1980. The singular value decomposition : its computation and some applications. *IEEE Trans.*, **AC-25** pp 164-176.

E. Kogbetliantz 1954. Diagonalization of general complex matrices as a new method for solution of linear equations. *Proc. Int. Congr. Math., Amsterdam* **2**, pp 356-357.

E. Kogbetliantz 1955. Solution of linear equations by diagonalization of coefficient matrices. *Quart. Appl. Math.*, **13**, pp 123-132.

S.Y. Kung 1978. A new identification and model reduction algorithm via singular value decomposition. *Proc. 12th Asilomar Conf. on Circuits, Systems and Computers, Pacific Grove*, pp 705-714.

S.Y. Kung, K.S. Arun and B.D. Rao 1983. State-space and SVD based methods for the harmonic retrieval problem. *Journal of the Optical Society of America* **73**, pp 1799-1811.

S.Y. Kung 1988. *VLSI array processors.* Englewood Cliffs, NJ., Prentice Hall.

H.T. Kung and C.E. Leiserson 1980. Algorithms for VLSI processor arrays. In *Introduction to VLSI systems (C.Mead and L.Conway, eds.)*, Addison-Wesley, pp 271-292.

W.E. Larimore 1983. System identification, reduced-order filtering and modelling via canonical variate analysis. *Proc. of the 1983 Automatic Control Conference, San Francisco.*

W.E. Larimore and F.T. Luk 1988. System identification and control using SVD's of systolic arrays. *Proc. SPIE High Speed Computing, Vol.* **880**, pp 37-48.

A. Laub 1979. A Schur method for solving the algebraic Riccati equation. *IEEE Trans.* **AC-24**, pp 913-921.

A.J. Laub, M.T. Heath, C.C. Paige and R.C. Ward 1987. Computation of system balancing transformations and other applications of simultaneous diagonalization algorithms. *IEEE Trans.* **AC-32**, No. 2, pp 115-122.

C. Lawson and R. Hanson 1974. *Solving least squares problems.* Prentice Hall Series in Automatic Computation, Englewood Cliffs, New Jersey.

F.T. Luk 1980. Computing the singular value decomposition on the ILLIAC IV. *ACM Trans. Math. Soft.* **6**, pp 524-539.

F.T. Luk 1985. A parallel method for computing the GSVD. *Int. J. Parallel & Distr. Comp.* **2**, pp 250-260.

F.T. Luk 1986a. A triangular processor array for computing singular values. *Lin. Alg. Appl.* **77**, pp 259-273.

F.T. Luk 1986b. A rotation method for computing the QR-decomposition. *SIAM J. Sci.Stat. Comput.* **7**, No. 2, pp 452-459.

F.T. Luk and H. Park 1989. *On parallel Jacobi orderings. SIAM J. Sci. Stat. Comput.* **10**, *No. 1, pp 18-26.*

J.V. McCanny and J.G. McWhirter 1987. *Some systolic array developments in the United Kingdom. IEEE Computer,* **20**, *No. 7, pp 51-63.*

J.G. McWhirter 1983. *Recursive Least Squares Minimization using a systolic array. Real-Time Signal Processing VI, Proc. SPIE, Vol.* **431**, *pp 105-112.*

J.G. McWhirter 1988. *A systolic array for recursive least squares minimisation and adaptive beamforming. Proc. NATO-ASI on Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms, Leuven, Belgium (in press).*

J.G. McWhirter 1989. *Algorithmic engineering: an emerging discipline. Advanced Algorithms and Architectures for Signal Processing IV, Proc. SPIE, Vol.* **1152**.

L. Mirsky 1960. *Symmetric Gauge functions and unitary invariant norms. Quart. J. Math. Oxford* **2**, *No. 11, pp 50-59.*

J.J. Modi 1988. *Parallel Algorithms and Matrix Computation.* Clarendon Press, Oxford.

C.B. Moler, J. Little, S. Bangert and S. Kleiman 1980. *Matlab user's Guide. The Mathworks Inc.*

M. Moonen, B. De Moor, L. Vandenberghe, J. Vandewalle 1989a. *On- and off-line identification of linear state space models. Int. Journal of Control* **49**, *No. 1, pp 219-232.*

M. Moonen, P. Van Dooren, J. Vandewalle 1989b. *An SVD updating algorithm for subspace tracking. Submitted for publication (ESAT-SISTA report 89-13a).*

M. Moonen, P. Van Dooren, J. Vandewalle 1989c. *A systolic array for SVD updating. Submitted for publication (ESAT-SISTA report 89-13b).*

M. Moonen, P. Van Dooren, J. Vandewalle 1989d. *Parallel updating algorithms, part two : generalized singular value decompositions. ESAT-SISTA report 89-23.*

M. Moonen, J. Vandewalle 1989e. *A systolic array for square root Kalman filtering. Submitted for publication (ESAT-SISTA report 89-27).*

M. Moonen, J. Vandewalle 1990a. *A hexagonally connected processor array for Jacobi-type matrix algorithms. Electronics Letters* **26**, *No. 6, pp 400-401.*

M. Moonen, J. Vandewalle 1990b. *A QSVD approach to on- and off-line state space identification. Int. Journal of Control* **51**, *No. 5, pp 1133-1146.*

M. Moonen, J. Vandewalle 1990c. *Recursive least squares with stabilized inverse factorization. Signal Processing* **21**, *No. 1, pp 1-15.*

M. Moonen, J. Vandewalle 1990d. *A systolic algorithm for Riccati and Lyapunov equations. Submitted for publication (ESAT-SISTA report 90-19).*

M. Moonen, P. Van Dooren, J. Vandewalle 1990e. *A Jacobi-type algorithm for LDU and Cholesky decomposition. Submitted for publication (ESAT-SISTA report 90-21).*

M. Moonen, J. Vandewalle 1990f. *A systolic array for recursive least squares computations. Submitted for publication (ESAT-SISTA report 90-22).*

M. Moonen, P. Van Dooren, J. Vandewalle 1990g. *A note on 'Efficient numerically stabilized rank-one eigenstructure updating. Submitted for publication (ESAT-SISTA report 90-22).*

Some of the material in the above reports is also available in the form of conference papers:

B. De Moor, M. Moonen, L. Vandenberghe en J. Vandewalle. *Identification of linear state space models with singular value decomposition using canonical correlation concepts. In : E. Deprettere (Ed.), Singular value decomposition and signal processing,* North-Holland, Amsterdam, *1988, pp 161-169.*

B. De Moor, M. Moonen, L. Vandenberghe en J. Vandewalle. *A geometrical approach for the identification of state space models with singular value decomposition. In : Proceedings of the International Conference on Acoustics, Speech and Signal Processing 1988,* New York, April *1988, pp 2244-2247.*

B. De Moor, M. Moonen, L. Vandenberghe en J. Vandewalle. *The application of the canonical correlation concept to the identification of linear state space models. In : A. Bensousan, J.L. Lions, (Eds.) Analysis and Optimization of Systems,* Springer-Verlag, Heidelberg, *1988, pp 1103-1114.*

B. De Moor, J. Vandewalle, M. Moonen, L. Vandenberghe, P. Van Mieghem. A geometrical strategy for the identification of linear multivariable systems with singular value decomposition. In : Proceedings of the 1988 IFAC Symposium on Identification and System Parameter Estimation, Beijing, August 1988, pp 700-704.

J. Swevers, M. Moonen, B. De Moor en J. Vandewalle. A black box identification algorithm for analysis of structural dynamics. In : Proceedings of the 13th International Seminar on Modal Analysis. Leuven, September 1988, C-36.

M. Moonen, P. Van Dooren, J. Vandewalle. SVD-updating for tracking slowly time-varying systems. In Advanced Algorithms and Architectures for Signal Processing IV, Proc. SPIE conference, Vol 1152. San Diego 1989.

J. Vandewalle, L. Vandenberghe, M. Moonen. The impact of singular value decomposition in systems theory, signal processing and circuit theory. In Three decades of mathematical system theory. A collection of surveys at the occasion of the fiftieth birthday of J.C. Willems. Lecture notes in control and information sciences, Springer Verlag, Berlin, 1989, Vol. 135, pp 453-479.

M. Moonen, J. Vandewalle. SVD-updating for tracking slowly time-varying systems. In Proc. of the International Symposium MTNS-89. Progress in systems and control theory series, Birkhäuser Boston Inc.

M. Moonen, P. Van Dooren, J. Vandewalle. Combined Jacobi-type algorithms in signal processing. In : R.J. Vaccaro(Ed.), The 2nd International Workshop on SVD and Signal Processing, North-Holland, Amsterdam. To appear.

M. Moonen, J. Vandewalle. A class of Jacobi-type updating algorithms. In : E. Deprettere (Ed.), Algorithms and Parallel VLSI Architectures, North-Holland, Amsterdam. To appear.

B.C. Moore 1981. Principal component analysis in linear systems : controllability, observability and model reduction. IEEE Trans. **AC-26**, *pp 17-32.*

J.G. Nash 1975. a one-sided transformation method for the singular value decomposition and algebraic eigenproblems. Comp. J. **18**, *pp 74-76.*

D.P. O'Leary and G.W. Stewart 1985. Data flow algorithms for parallel matrix computations. Comm. ACM **28**, *pp 840-853.*

C.C. Paige and M. Saunders 1977. Least squares estimation of discrete linear dynamic systems using orthogonal transformations. SIAM J. Numer. Anal. **14**, *No. 2, pp 180-193.*

C.C. Paige and M. Saunders 1981. Towards a generalized singular value decomposition. SIAM J. Numer. Anal., **18**, *No. 3, pp 398-405.*

C.C. Paige 1984. A note on a result of Sun Ji-Guang: Sensitivity of the CS and GSV decompositions. SIAM J. Numer. Anal., **21**, *No. 1, pp 186-191.*

C.C Paige 1985. The general linear model and the generalized singular value decomposition. Lin. Alg. Appl. **70**, *pp 269-284.*

C.C. Paige 1986. Computing the generalized singular value decomposition. SIAM J. Sci. Stat. Comput. **7**, *pp 1126-1146.*

C.C. Paige and P. Van Dooren 1986. On the convergence of Kogbetliantz's Algorithm for computing the singular value decomposition. Lin. Alg. Appl. **77**, *pp 301-313.*

C.T. Pan, R.J. Plemmons 1989. Least squares modifications with inverse factorization: parallel implications. J. Comp. App. Math. **27**, *No. 1-2, pp 109-127.*

B.N. Parlett 1980. The symmetric eigenvalue problem. Prentice Hall, Englewood Cliffs, New Jersey.

A. Paulraj, R. Roy and T. Kailath 1986. A subspace rotation approach to signal parameter estimation. Proceedings of the IEEE, pp 1044-1045.

V.F. Pisarenko 1973. The retrieval of harmonics from a covariance function. Geophys. J. Royal Astronomical Soc. **33**, *pp 347-366.*

D.A. Pope and C. Tompkins 1957. Maximizing functions of rotations : experiments concerning speed of diagonalization of symmetric matrices using Jacobi's method. J. Assoc. Comp. Mach. **4**, *pp 459-466.*

R.H. Roy 1987. ESPRIT : Estimation of signal parameters via rotational invariance techniques. PhD thesis, Stanford Univ., Stanford CA.

R. Schreiber 1986. Implementation of adaptive array algorithms. IEEE Trans. **ASSP-34**, *pp 1038-1045.*

T.J. Shepherd and J.G. McWhirter 1987. *A pipelined array for linearly constrained least squares optimization. In Mathematics in Signal Processing. (T.S. Durrani, J.B. Abbiss, J.E. Hudson, R.W. Madan, J.G. McWhirter and T.A. Moore, Eds.) Clarendon Press, Oxford. pp 607-635.*

T.J. Shepherd, J.G. McWhirter and J.E. Hudson 1990. *Parallel weight extraction from a systolic adaptive beamformer. In Mathematics in Signal Processing II. (J.G. McWhirter, Ed.) Clarendon Press, Oxford. pp 775-790.*

B.T. Smith, J.M. Boyle, Y. Ikebe, V.C. Klema and C.B. Moler 1970. *Matrix eigensystem routines : EISPACK Guide, 2nd edition. Springer Verlag, New York, NY.*

J.M Speiser 1986. *Signal processing computational needs. Proc. SPIE, Vol.* **696**, *pp 2-6.*

G.W. Stewart 1973a. *Introduction to matrix computations. Academic Press, New York.*

G.W. Stewart 1973b. *Error and perturbation bounds for subspaces associated with certain eigenvalue problems. SIAM Review,* **15**, *No. 4, pp 727-764.*

G.W. Stewart 1985. *A Jacobi-like algorithm for computing the Schur decomposition of a nonhermitian matrix. SIAM J. Sci. Stat. Comput.* **6**, *No. 4, pp 853-863.*

J.-G. Sun 1983a. *Perturbation analysis for the generalized singular value problem. SIAM J. Numer. Anal.* **20**, *No. 3, pp 611-625.*

J.-G. Sun 1983b. *Perturbation theorems for generalized singular values. J. of Comp. Math.* **1**, *No. 3, pp 233-242.*

T.Y. Sung and Y.H. Hu 1987. *Parallel VLSI implementation of the Kalman filter. IEEE Trans.* **AES-23**, *No.2, pp 215-224.*

D.R. Sweet 1984. *fast Toeplitz orthogonalization. Numer. Math.* **43**, *pp 1-21.*

J. Swevers, D. Torfs, J. De Schutter, H. Van Brussel. *Accurate tracking control of a flexible one-link robot. Proceedings of the 15th International Seminar on Modal Analysis. Leuven, September 1988, pp 353-366.*

D.W. Tufts and R. Kumaresan 1982. *Singular value decomposition and improved frequency estimation using linear prediction. IEEE Trans* **ASSP-30**, *pp 671-675.*

J. Vandewalle and B. de Moor 1988. *A variety of applications of singular value decomposition in identification and signal processing. in E. Deprettere (Ed.), Singular value decomposition and signal processing, North-Holland, Amsterdam. pp 43-92.*

P. Van Dooren 1981. *A generalized eigenvalue approach for solving Riccati equations. SIAM J. Sci. Stat. Comput.* **2**, *pp 121-135.*

S. Van Huffel 1987a. *Analysis of the total least squares problem and its use in parameter estimation. PhD Thesis Kath. Univ. Leuven, El. Eng. Dept.*

S. Van Huffel and J. Vandewalle 1987b. *Algebraic relations between classical regression and total least squares estimation. Lin. Alg. Appl., pp. 149-162.*

S. Van Huffel 1989. *Analysis and properties of the generalized total least squares problem $AX \approx B$ when some or all columns in A are subject to error. SIAM J. Matrix Anal. Appl.* **10**, *No. 3, pp 294-315.*

H.P.M. Van Kempen 1966. *On the convergence of the classical Jacobi method for real symmetric matrices with non-distinct eigenvalues. Numer. Math.* **9**, *pp 11-18.*

H.P.M. Van Kempen 1966. *On the quadratic convergence of the secial cyclic Jacobi method. Numer. Math.* **9**, *pp 19-22.*

C.F. Van Loan 1976. *Generalizing the singular value decomposition. SIAM J. Numer. Anal.* **13**, *No. 1, pp 76-83.*

J.H. Wilkinson 1962. *Note on the quadratic convergence of the cyclic Jacobi process. Numer. Math.* **4**, *pp 296-300.*

J.H. Wilkinson 1965. *The Algebraic Eigenvalue Problem. Oxford University Press.*

J.H. Wilkinson 1968. *Almost diagonal matrices with multiple or close singular values. Lin. Alg. Appl.* **1**, *pp 1-12.*

H.P. Zeiger and A.J. Mc Ewen 1974. *Approximate linear realizations of given dimensions via Ho's algorithm. IEEE Trans.* **AC-19**, *p 153.*

H. Zha 1989. restricted singular value decomposition of matrix triplets. Report **SC 89-2**. *Konrad-Zuse-Zentrum für Informationstechnik Berlin.*

M.D. Zoltowski 1987. Signal processing applications of the method of total least squares. Proc. 21st Asilomar Conf. on Signals, Systems and Computers. Pacific Grove, CA. pp 290-296.

M.D. Zoltowski and D. Stavrinides 1989. Sensor array signal processing via a Procrustes rotation based eigenanalysis of the ESPRIT data pencil. IEEE Trans. **ASSP-37**, *pp 832-861.*