KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT DER TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK-ESAT
Kard. Mercierlaan 94, 3001 Leuven (Heverlee)

CONTINUOUS-TIME MATRIX ALGORITHMS, SYSTOLIC ALGORITHMS AND ADAPTIVE NEURAL NETWORKS

Promotoren:
Prof. Dr. Ir. J. Vandewalle,
Dr. Ir. M. Moonen.

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

door

Jeroen DEHAENE

Oktober 1995

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT DER TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK-ESAT
Kard. Mercierlaan 94, 3001 Leuven (Heverlee)

CONTINUOUS-TIME MATRIX ALGORITHMS, SYSTOLIC ALGORITHMS AND ADAPTIVE NEURAL NETWORKS

Jury:
Prof. W. Dutré, vice-decaan, voorzitter,
Prof. J. Vandewalle, promotor,
Dr. M. Moonen, promotor,
Prof. A. Bultheel,
Prof. B. De Moor,
Dr. F. Gaston (Univ. of Birmingham, UK),
Prof. U. Helmke (Univ. Würzburg, Germany),
Prof. P. Van Dooren (U.C.L., Louvain-la-Neuve)

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

door

Jeroen DEHAENE

U.D.C. 519.61

Oktober 1995

Voorwoord

Bij het beëindigen van dit werk, houd ik eraan enkele mensen te bedanken.

In de eerste plaats bedank ik mijn promotoren, Prof. J. Vandewalle en Dr. M. Moonen. Joos Vandewalle wil ik bedanken omdat ik sterk apprecieer hoe hij vanuit zijn eigen interesse voor zowel fundamentele als toegepaste wetenschappen, mij en anderen de kans en de vrijheid geeft om naar ingenieursnormen sterk theoretisch onderzoek te verrichten, en tegelijk aandacht blijft vragen voor de toepasbaarheid. Marc Moonen verdient mijn dank omdat hij de hand had in enkele beslissende wendingen, omdat zijn constructieve kritiek heeft bijgedragen tot de begrijpbaarheid van deze tekst en omdat de dreiging van zijn perfectionistisch oordeel mijn spontane afkeer voor formele details heeft kunnen kleinkrijgen.

Verder dank ik de andere leden van het leescomité, Prof. B. De Moor en Prof. A. Bultheel. Bart De Moor moet ik ook bedanken om zijn bulderend enthousiasme, dat zo luid weergalmt dat er ook in dit werk van alles meetrilt, en omdat het dankzij gelijke interesses en ondanks een verschil in stijl, aangenaam samenwerken is met hem. Adhemar Bultheel wil ik bedanken voor de zorgvuldigheid waarmee hij dit werk ondanks de examentijd heeft nagelezen.

Mijn dank gaat ook uit naar de voorzitter van de jury, Prof. W. Dutré, voor zijn bereidwillige medewerking.

I would also like to thank the other members of the jury, Dr. F. Gaston (University of Birmingham), Prof. U. Helmke (Universität Würzburg) and Prof. P. Van Dooren (Université Catholique de Louvain, Louvain-la-Neuve), for their cooperation. I owe special thanks to Fiona Gaston, who has made some of my sentences sound more English (I hope this one is all right). I thank Uwe Helmke for the enthusiasm with which he promotes differential geometry in the world of engineers. I also want to apologize to the readers, and especially the members of the jury, for descriptions with too many words or notations with too many subscripts, where I simply wanted to say 'this here on this figure'.

I also thank Prof. R.W. Brockett (Harvard University) for being interested in my work. And I am looking forward to visiting him and his research group. I

am grateful to Prof. J.A. Nossek (Technische Universität München) for having invited us in his special session at MTNS '93 in Regensburg, Germany.

Verder wil ik ook een hele rij collega's bedanken. I want to thank Yi Cheng for the fruitful and pleasant cooperation. Ik had verschillende nuttige discussies met Filiep Vanpoucke, met wie ik reeds vele voetsporen deelde, Johan Suykens en Yves Moreau, van de neurale-netwerkengroep, Lieven Vandenberghe, die indruk maakte op mij als verse beginneling door de inhoud en de stijl van zijn doctoraat, Johan David, die mij ook in contact bracht met het werk van Brockett, Lieven De Lathauwer en Peter Van Overschee. I also thank Johan David, Jairo Espinosa and Yves Moreau for the shared walks for a cup of tea or coffee.

Ik wil ook de secretaressen van het eerste en het laatste uur bedanken, Ann Deforce, Ingrid Tokka, Rita De Wolf, Rebecca Crabbé en Ida Tassens, en dan mag ik ook het organisatorische werk van Bart Motmans niet vergeten. Tenslotte wil ik ook de gehele systeemploeg bedanken.

Ik dank ook het Nationaal Fonds voor Wetenschappelijk Onderzoek, voor de financiële steun.

Van de vele andere mensen, die mijn dank wel langs minder officiële kanalen zullen vernemen, wens ik tenslotte nog speciaal mijn ouders en ook mijn broers te vermelden voor de steun die ik kreeg en krijg op mijn weg.

Abstract

In the domain of 'continuous-time matrix algorithms', matrix based algorithms are studied from the viewpoint of continuous-time systems theory and differential geometry. We put emphasis on formulas for tracking decompositions of a time-varying matrix, and present them as tools for the design and analysis of matrix algorithms. We define a class of continuous-time matrix algorithms with a uniform parallel signal flow graph. We derive algorithms for recursive least-squares estimation, belonging to this class, which are continuous-time limits of known systolic algorithms. Some of them are candidates for analog realization. For algorithms for subspace tracking, belonging to the same class, we present new analysis results based on continuous-time concepts. From these algorithms we also derive new fully pipelined systolic algorithms, inheriting the main properties of their continuous-time counterparts. We reinterpret the presented continuous-time adaptive signal processing algorithms as adaptation laws for neural networks and derive other adaptation laws, based on isospectral algorithms, for neural memory applications. Finally, we illustrate the relevance of the differential geometric approach for the design of optimization algorithms, by studying an application in robust control theory, the computation of the structured singular value.

Abstract

Het domein van de 'tijdscontinue matrixalgoritmen' bestudeert matrixgebaseerde algoritmen vanuit het standpunt van de tijdscontinue systeemtheorie en de differentiaalmeetkunde. We benadrukken formules voor het volgen van ontbindingen van een tijdsafhankelijke matrix, die gebruikt kunnen worden voor het ontwerp en de analyse van matrixalgoritmen. We definiëren een klasse van tijdscontinue matrixalgoritmen met een uniform parallelle signaalgrafe. We leiden algoritmen voor recursieve kleinste-kwadratenschatting af, die tot deze klasse behoren, en die tijdscontinue limieten zijn van gekende systolische algoritmen. Sommige komen in aanmerking voor analoge realisatie. Voor algoritmen voor deelruimtevolgen, die tot dezelfde klasse behoren, geven we een nieuwe analyse, op basis van tijdscontinue concepten. Uit deze algoritmen worden ook nieuwe volledig 'gepipelinede' systolische algoritmen afgeleid, die de belangrijkste eigenschappen van hun tijdscontinue tegenhangers overerven. We herinterpreteren de voorgestelde tijdscontinue signaalverwerkingsalgoritmen als adaptatiewetten voor neurale netwerken en leiden ook andere adaptatiewetten af, gebaseerd op isospectrale algoritmen, voor neurale geheugentoepassingen. Tenslotte, illustreren we de relevantie van de differentaalmeetkundige benadering voor het ontwerp van optimisatie-algoritmen, aan de hand van een toepassing in robuuste controletheorie, de berekening van de gestructureerde singuliere waarde.

Notation

notation	meaning	def./p.
A^T	transpose of matrix A	
A^{-T}	inverse transpose of matrix A	
A^H	Hermitian transpose of matrix A	
c^*	complex conjugate of c	
$A_{\alpha,\beta}$	submatrix of A with row indices in α and column indices in β. Subscripts $_{\leqslant i}$, $_{<i}$, $_{\geqslant i}$ or $_{>i}$ denote indices (strictly) less or (strictly) greater than i. A subscript $_{,}$ denotes the set of all indices. Single entries are denoted by the corresponding lower-case symbol if no confusion is possible: $a_{i,j} = A_{i,j}$.	
$\dot{x}$	time derivative of x	10
$\langle A, B\rangle$	standard inner product of matrices A and B	2.7,3.27
$\|A\|_F$	Frobenius norm of matrix A	2.7
$[X, Y]$	(Lie) bracket of matrices X and Y	2.10
$X. * Y$	elementwise matrix product	2.9
$\angle(x, y)$	angle between vectors x and y	
$\langle f, g\rangle_{(t)}$	inner product of functions $f, g \in L_{2,\lambda}(-\infty, t]$ or generalized matrix product of vector functions $f, g \in (L_{2,\lambda}(-\infty, t])^n$.	6.2
$\|f\|_{(t)}$	norm of function $f \in L_{2,\lambda}(-\infty, t]$	6.2
$y \parallel \{x_1, \ldots, x_n\}$		78
$y \perp \{x_1, \ldots, x_n\}$		78
$\mathbb{C}$	set of complex numbers	
$\mathcal{C}_m(t)$	cross product subspace of dimension m	7.12
$\mathcal{C}_A$	space of $n \times n$-matrices commutable with $A \in \mathbb{R}^{n\times n}$	2.8
c_A	commutator of matrix A	2.9
$\mathrm{cf}(A)$	Cholesky factor of matrix A	3.1
$\mathrm{col}(A)$	column space of matrix A	
$\mathrm{D}\Phi\|_X$	derivative of a map Φ at X	31

d_A	elementwise multiplication with matrix A	2.9
$\mathrm{diag}\{a_1, \ldots, a_n\}$	diagonal matrix with diagonal entries $a_1, \ldots, a_n$	
$\mathrm{diag}[A_1, \ldots, A_n]$	block diagonal matrix with block diagonal entries $A_1, \ldots, A_n$	
$\mathrm{diag}(A)$	diagonal part of matrix A	2.9
$\dim \mathcal{S}$	dimension of set $\mathcal{S}$	
$E\{x\}$	expectation of random variable x	
$E^{(i,j)}$	matrix of standard basis	2.6
$e_{R,S}$	equivalence transformation with matrices R and S	2.9
$\mathcal{G}_{diag}(n)$	group of nonsingular diagonal $n \times n$-matrices	2.24
$\mathsf{GL}(n)$	general linear group of $n \times n$-matrices	2.24
$\mathcal{G}_{low}(n)$	group of nonsingular lower triangular $n \times n$-matrices	2.24
$\mathcal{G}_{lows}(n)$	group of lower triangular $n \times n$-matrices with unit diagonal	2.24
$\mathcal{G}_{upp}(n)$	group of nonsingular upper triangular $n \times n$-matrices	2.24
$\mathcal{G}_{upp}(n)^+$	group of upper triangular $n \times n$-matrices with positive diagonal	2.24
$\mathcal{G}_{upps}(n)$	group of upper triangular $n \times n$-matrices with unit diagonal	2.24
$\mathfrak{g}_{diag}(n)$	Lie algebra of $\mathcal{G}_{diag}(n)$	2.25
$\mathfrak{gl}(\mathrm{n})$	Lie algebra of $\mathsf{GL}(n)$	2.25
$\mathfrak{g}_{low}(n)$	Lie algebra of $\mathcal{G}_{low}(n)$	2.25
$\mathfrak{g}_{lows}(n)$	Lie algebra of $\mathcal{G}_{lows}(n)$	2.25
$\mathfrak{g}_{upp}(n)$	Lie algebra of $\mathcal{G}_{upp}(n)$	2.25
$\mathfrak{g}_{upp}(n)^+$	Lie algebra of $\mathcal{G}_{upp}(n)^+$	2.25
$\mathfrak{g}_{upps}(n)$	Lie algebra of $\mathcal{G}_{upps}(n)$	2.25
$G^{i\|j}$	Givens rotation affecting components i and j	2.16
$\mathrm{grad}_x f$	gradient of f at x	3.21
$\mathcal{I}_A$	isospectral manifold through matrix A	2.17
$\mathrm{icf}(A)$	inverse Cholesky factor of matrix A	3.1
id	identity map	
$\mathrm{ilf}(A)$	inverse L-factor of matrix A	3.1
$\mathrm{Im}\, c$	imaginary part of a complex number c	
$\mathrm{inv}(A)$	inverse of matrix A	3.1
$\mathrm{irf}(A)$	inverse R-factor of matrix A	3.1
$\mathrm{iuf}(A)$	inverse U-factor of matrix A	3.1
$L_{2,\lambda}(-\infty, t]$	set of square integrable functions with exponential weight over $(-\infty, t]$	6.2
l_A	left multiplication with A	2.9
$\mathrm{lf}(A)$	L-factor of matrix A	3.1
$\mathrm{low}(A)$	lower triangular part of matrix A	2.9
$\mathrm{lowh}(A)$	lower triangular half of matrix A	2.9
$\mathrm{lows}(A)$	strictly lower triangular part of matrix A	2.9
N_c	continuous-time information matrix	6.5
N_d	discrete-time information matrix	6.12

$\mathrm{null}(A)$	null space of matrix A	
$\mathcal{O}(a)$	orbit of element a	25
$\mathsf{O}(n)$	orthogonal group of $n \times n$-matrices	2.24
$\mathrm{O}()$	order of magnitude	
$\mathcal{Q}$	group of structured unitary matrices	11.2
$\mathfrak{q}$	Lie algebra of $\mathcal{Q}$	11.4
$\mathrm{qf}(A)$	Q-factor of matrix A	3.1
$\mathbb{R}$	set of real numbers	
r_A	right multiplication with A	2.9
$\mathrm{Re}\, c$	real part of a complex number c	
$\mathrm{rf}(A)$	R-factor of matrix A	3.1
$\mathcal{S}_{diag}(n)$	space of diagonal $n \times n$-matrices	2.8
$\mathcal{S}_{low}(n)$	space of lower triangular $n \times n$-matrices	2.8
$\mathcal{S}_{lows}(n)$	space of strictly lower triangular $n \times n$-matrices	2.8
$\mathcal{S}_{skew}(n)$	space of skew symmetric $n \times n$-matrices	2.8
$\mathcal{S}_{symm}(n)$	space of symmetric $n \times n$-matrices	2.8
$\mathcal{S}_{upp}(n)$	space of upper triangular $n \times n$-matrices	2.8
$\mathcal{S}_{upps}(n)$	space of strictly upper triangular $n \times n$-matrices	2.8
$\mathsf{SL}(n)$	special linear group of $n \times n$-matrices	2.24
$\mathsf{SO}(n)$	special orthogonal group of $n \times n$-matrices	2.24
$\mathfrak{sl}(\mathrm{n})$	Lie algebra of $\mathsf{SL}(n)$	2.25
$\mathfrak{so}(\mathrm{n})$	Lie algebra of $\mathsf{SO}(n)$	2.25
s_R	similarity transformation with matrix R	2.9
$\mathrm{span}\{v_1, \ldots, v_n\}$	span of vectors $v_1, \ldots, v_n$	
$\mathrm{St}(m, n)$	Stiefel manifold of $n \times m$-matrices	2.21
$\mathrm{Stab}(a)$	Stabilizer of element a	25
$T_X\mathcal{M}$	tangent space to a manifold $\mathcal{M}$ at $X \in \mathcal{M}$	2.3
$\mathrm{Tr}(A)$	trace of matrix A	
$\mathsf{U}(n)$	group of unitary $n \times n$-matrices	3.28
$\mathrm{uf}(A)$	U-factor of matrix A	3.1
$\mathrm{upp}(A)$	upper triangular part of matrix A	2.9
$\mathrm{upph}(A)$	upper triangular half of matrix A	2.9,3.31
$\mathrm{upps}(A)$	strictly upper triangular part of matrix A	2.9
$\mathrm{vec}(A)$	Vector representation of matrix A	2.5
$\mathbb{Z}$	set of integers	
$\mathbf{\Delta}$	uncertainty structure of a structured singular value problem	11.1
$\delta_{i,j}$	Kronecker-δ, $\delta_{i,j} = [i = j]$	
$\mu_{\mathbf{\Delta}}$ or μ	structured singular value	11.1,11.2
$\rho(A)$	spectral radius of matrix A	11.2
$\pi(A)$	projection of matix A on $\mathbf{\Delta}$	11.5
$\rho_{\mathrm{skew}}(A)$	projection of matrix A on $\mathcal{S}_{skew}(n)$, parallel to $\mathcal{S}_{upp}(n)$	2.9
$\rho_{\mathrm{upp}}(A)$	projection of matrix A on $\mathcal{S}_{upp}(n)$, parallel to $\mathcal{S}_{skew}(n)$	2.9

Acronyms

EVD	EigenValue Decomposition
GK	Gentleman-Kung
LCMV	Linearly Constrained Minimum Variance
LMS	Least Mean Squares
MM	Moonen-McWhirter
RLS	Recursive Least Squares
SVD	Singular Value Decomposition

Contents

Voorwoord i

Abstract iii

Notation v

Contents ix

Nederlandse Samenvatting xiii

1 Introduction 1

1.1 General overview 2

1.2 Chapter by chapter overview 4

2 Definitions and concepts 9

2.1 Continuous-time matrix algorithms 10

2.2 Matrix space as a Euclidean vector space 13

2.3 Differentiable manifolds, Lie groups and homogeneous spaces . . 16

2.3.1 The orthogonal group $\mathsf{O}(n)$, as a regular manifold 17

2.3.2 Isospectral manifolds 20

2.3.3 The Stiefel manifold of rectangular orthogonal matrices . 22

2.3.4 Lie groups and homogeneous spaces 23

2.4 Continuous-time algorithms on manifolds 26

2.5 Conclusion 27

3 Tools for design and analysis 29

3.1 Tracking matrix decompositions 29

3.1.1 Tracking the inverse and decompositions of the form $X = YZ$ 30

3.1.2 Tracking the symmetric eigenvalue decomposition and the singular value decomposition . . . 33
3.2 Matrix decompositions as nonlinear coordinate transformations . 38
3.2.1 The polar coordinate picture . . . 38
3.2.2 The QR decomposition as a coordinate transformation . . 39
3.2.3 The symmetric EVD as a coordinate transformation . . . 42
3.3 Gradients and gradient flows . . . 47
3.4 The complex case . . . 52
3.5 Conclusion . . . 54

4 The QR flow and the double bracket flow 57
4.1 The QR flow . . . 57
4.1.1 The power algorithm . . . 58
4.1.2 The QR algorithm . . . 59
4.1.3 A square-root QR algorithm . . . 61
4.2 The double bracket flow . . . 62
4.3 Conclusion . . . 67

5 Uniform parallel algorithms 69
5.1 Definition . . . 70
5.2 Signal flow graphs . . . 70

6 Continuous-time recursive least-squares estimation 75
6.1 RLS estimation . . . 76
6.2 An algorithm based on Cholesky factor tracking . . . 78
6.2.1 Solution by Cholesky factor tracking . . . 78
6.2.2 Derivation of the continuous-time GK algorithm . . . 80
6.2.3 Local interpretation . . . 81
6.3 The link with discrete-time systolic algorithms . . . 83
6.4 Related algorithms by LDL tracking . . . 86
6.5 An algorithm by inverse Cholesky factor tracking . . . 88
6.6 Analog realization . . . 90
6.7 Conclusion . . . 92

7 Continuous-time subspace tracking 99
7.1 Subspace estimation, subspace tracking . . . 100
7.2 Cross-product subspace tracking algorithms . . . 102
7.3 Interpretations in different contexts . . . 104

7.3.1 Relation with stochastic gradient algorithms 104
7.3.2 Relation with spherical subspace trackers 107
7.3.3 Relation with autonomous continuous-time algorithms . . 109
7.3.4 Geometric interpretations 110
7.4 Analysis of the algorithm 111
7.4.1 Evolution of the signal subspace estimate col(A) 112
7.4.2 Attraction to orthogonality 120
7.5 Notes on a statistical performance analysis 122
7.6 Conclusion 123

8 Discrete-time subspace tracking **125**
8.1 Exact integration for a piecewise constant input 126
8.2 Pipelining 130
8.3 An LU based algorithm 132
8.4 Comparison with related algorithms 133
8.4.1 Comparison with stochastic gradient algorithms 135
8.4.2 Comparison with the spherical subspace tracker 135
8.4.3 Step sizes 137
8.5 Conclusion 138

Appendix **153**
8.A Derivation of the signal flow graph of algorithm 8.5 153

9 Neural network adaptation laws for adaptive signal processing **159**
9.1 General concepts 159
9.2 Neural interpretation of systems of type I and II 162
9.3 Neural interpretation of systems of type III 164
9.4 Conclusion 167

10 Adaptation laws for neural network memories **169**
10.1 Associative memory 170
10.2 A nonautonomous variant of the double bracket flow 174
10.3 Stabilization of the spectrum 178
10.4 Stabilization of the spectrum with laws of type I 180
10.5 Adaptation laws for memory 183
10.6 Comparison with other adaptation laws 187
10.7 Simulation examples 189
10.8 Conclusion 191

Appendix **193**
10.A Comparison of three related neural network models 193

11 Optimization algorithms for the structured singular value **199**
11.1 Optimization strategies for the structured singular value problem 200
11.1.1 Problem definition 200
11.1.2 Optimization strategy 201
11.2 The manifold $\mathcal{Q}$ of structured unitary matrices 202
11.3 The Steepest Ascent Algorithm 204
11.3.1 The algorithm 204
11.3.2 Computational cost 207
11.3.3 Remarks on the convergence issue 208
11.4 The Conjugate Gradient Algorithm 208
11.5 Continuous-time algorithms 210
11.5.1 A continuous-time gradient ascent algorithm 210
11.5.2 Combination with the power method 210
11.6 Numerical experiments 212
11.7 Conclusion 214

12 Conclusion **217**
12.1 Parallelism 217
12.2 Analog realizations 219
12.3 Systolic algorithms 219
12.4 Discrete-time optimization algorithms 221
12.5 Analysis of continuous-time algorithms 223
12.6 Neural network adaptation laws 223
12.7 Main contributions 224

Tijdscontinue matrix-algoritmen, systolische algoritmen en adaptieve neurale netwerken

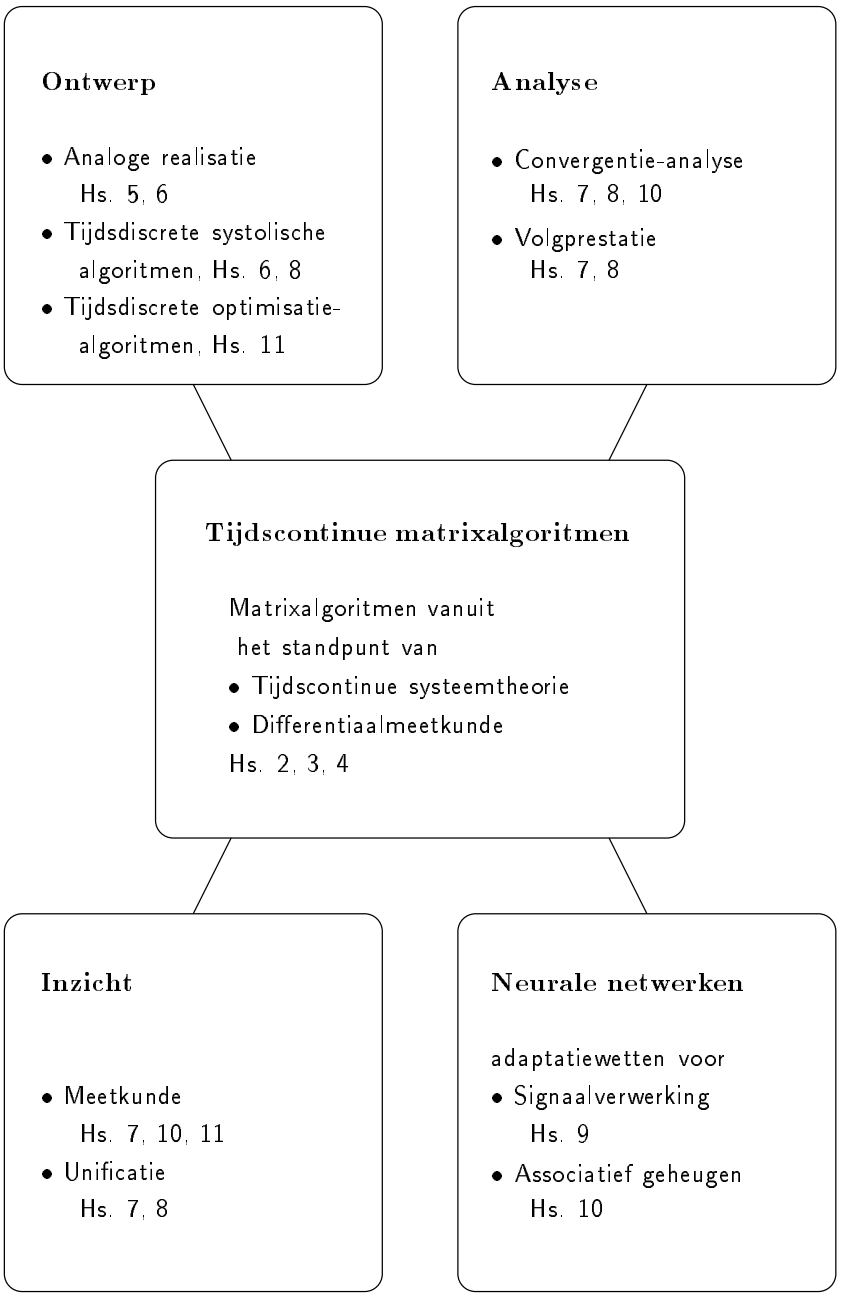
Nederlandse samenvatting

Hoofdstuk 1. Inleiding

Het bindende thema van dit werk is de theorie van *tijdscontinue matrixalgoritmen*. Deze theorie werd uitgewerkt in een aantal recente publicaties, waarin klassieke matrixproblemen bestudeerd worden vanuit het standpunt van tijdscontinue systeemtheorie en differentiaalmeetkunde. Deze aanpak, die verder de 'tijdscontinue benadering' genoemd wordt, heeft reeds geleid tot nieuwe inzichten in gekende algoritmen, unificatie van op het eerste gezicht verschillende resultaten en nieuwe, ook tijdsdiscrete, algoritmen. Voor een overzicht en referenties verwijzen we naar het recente boek van Helmke en Moore, [55].

Figuur 0.1 geeft een algemeen inhoudelijk overzicht. Een *eerste* aspect hierbij is het ontwerp van zowel tijdscontinue als tijdsdiscrete algoritmen via de tijdscontinue benadering. Voor alle beschouwde toepassingen, worden verschillende tijdscontinue algoritmen afgeleid. Belangrijke instrumenten hierbij zijn het gebruik van gradiënten van functies over manifolds, en het gebruik van de afgeleiden van matrixontbindingen, voor het ontwerp en de analyse van algoritmen.

Hoewel *analoge realisatie* van deze algoritmen aantrekkelijk is, komen alleen de meest eenvoudige algoritmen hiervoor in aanmerking. We bespreken hiervan enkele hoogniveau-aspecten.



Figuur 0.1: Algemeen overzicht

Tijdscontinue algoritmen kunnen echter ook nuttig zijn bij het *ontwerp van tijdsdiscrete algoritmen*. Een nieuwe bijdrage van dit werk ligt in het ontwerp van *systolische algoritmen* via de tijdscontinue benadering. Veel tijscontinue algoritmen blijken eenvoudige uniform parallelle signaalgrafen te hebben. We tonen hoe sommige van deze signaalgrafen continue versies van systolische algoritmen zijn en leiden nieuwe systolische algoritmen af, die eigenschappen overerven van hun continue tegenhangers. Hiervoor bestaat echter geen automatische methode.

Behalve voor het ontwerp van parallelle algoritmen kan de tijdscontinue (differentiaalmeetkundige) benadering ook van nut zijn bij het ontwerp van tijdsdiscrete optimisatie-algoritmen, zeker in het geval van regelmatige beperkingen. Differentiaalmeetkundige concepten zijn nuttig voor het ontwerp van algoritmen die 'leven' in een ruimte, gegeven door een gelijkheidsbeperking. Bovendien zijn tijdscontinue gradiëntalgoritmen nuttig voor verkennende experimenten omdat ze vaak eenvoudiger te programmeren zijn (met behulp van standaardintegratiepaketten) dan tijdsdiscrete versies.

Het *tweede* aspect is de analyse van algoritmen. We concentreren ons vooral op de analyse van adaptieve signaalverwerkingsalgoritmen. We bewijzen resultaten betreffende de *convergentie* en de *volgprestatie*, met behulp van tijdscontinue resultaten voor het volgen van singuliere waarden en canonieke hoeken. Gelijkaardige ideeën worden ook uitgewerkt voor de analyse van neurale adaptatiewetten voor geheugentoepassingen.

Als *derde* aspect, stellen we dat de tijdscontinue benadering een *inzichtelijke* benadering is. Hierbij leggen we vooral de nadruk op meetkundige interpretaties. Hoewel de appreciatie van dit meer subjectieve aspect uiteraard afhangen van persoonlijke smaak, is het duidelijk dat tijdscontinue algoritmen en hun afleidingen vaak verrassend eenvoudiger en anders zijn dan hun tijdsdiscrete tegenhangers. En alleen al het feit dat het gezichtspunt verschillend is, kan de inspiratie slechts ten goede komen. Bovendien biedt de tijdscontinue benadering vaak een *unificerend* kader voor verschillende algoritmen. Zo blijkt ondermeer dat ogenschijnlijk sterk verschillende algoritmen voor deelruimtevolgen, dezelfde vorm aannemen in de tijdscontinue limiet.

Als *vierde* aspect beschouwen we neurale netwerkinterpretaties. De studie van adaptatiewetten voor artificiële neurale netwerken was van in het begin een motivatie voor dit werk, maar omdat het onderzoek geleidelijk uit dit domein gegroeid is, worden de meeste resultaten voorgesteld als speciale gevallen of andere interpretaties.

De algemene idee, ontleend aan het domein van de neurale netwerken, is de volgende manier van taken uit te voeren met een dynamisch systeem. Op een locale schaal (neuronen, gewichten), vertoont het systeem eenvoudig en vaak uniform gedrag, waarbij alle elementen evolueren volgens dezelfde wet (of een klein aantal verschillende wetten) op basis van informatie die locaal beschikbaar

is in het netwerk. Op een *globale* schaal resulteert dit in interessant gedrag, zoals geheugen, controle, of verschillende types van informatieverwerking. Een van de problemen is dat de relatie tussen zelfs eenvoudig locaal gedrag en het resulterende globale gedrag heel complex kan zijn. In dit werk bestuderen we een klasse van systemen, die kan gerealiseerd worden op een netwerkstructuur met eenvoudige locale wetten, en zich tegelijkertijd leent tot een grondige wiskundige analyse van het globale gedrag. We beschouwen hierbij toepassingen in adaptieve signaalverwerking en associatieve geheugens.

Hoofdstuk 2. Definities en concepten van tijdscontinue matrixalgoritmen

De hoofdstukken 2, 3 en 4 vormen een inleiding tot tijdscontinue matrixalgoritmen. Hierbij wordt zoveel mogelijk het jargon van de differentiaalmeetkunde vermeden, en wordt de nadruk gelegd op meetkundige interpretaties.

Tijdscontinue algoritmen zijn tijdscontinue dynamische systemen waarvan het gedrag geïnterpreteerd wordt als een algoritme. Met de benaming *matrix*algoritmen duiden we systemen aan waarvan de toestand (of een deel ervan) als een matrix geïnterpreteerd wordt. In hoofdstuk 2, worden enkele definities en eigenschappen van de matrixruimte als Euclidische vectorruimte in herinnering gebracht, en wordt aandacht besteed aan enkele differentieerbare manifolds, die een belangrijke rol spelen in dit werk.

In dit werk volstaat het over een differentieerbare manifold te denken als een deelverzameling van een Euclidische ruimte, met op ieder punt een unieke raakruimte van een vaste dimensie, gelijk aan de dimensie van de manifold. De raakruimte $T_X\mathcal{M}$ aan een manifold $\mathcal{M}$ in een punt $X \in \mathcal{M}$ is de ruimte van alle mogelijke snelheidsvectoren in X van differentieerbare paden door X.

De meeste manifolds die van belang zijn in dit werk zijn Lie-groepen en homogene ruimten. De structuur van Lie-groepen en homogene ruimten, zorgt voor een hoge regelmaat in de geometrie van deze manifolds. Drie belangrijke manifolds zijn de Lie-groep van orthogonale matrices en de homogene ruimten van symmetrische matrices met een gegeven spectrum ('isospectrale manifolds'), en van rechthoekige matrices met orthogonale kolommen ('Stiefel manifold').

Voor een beschrijving van hun raakruimten verwijzen we naar de proposities 2.12, 2.19 en 2.22. Deze raakruimten zijn ondermeer van belang voor het beschrijven van algoritmen die op een manifold blijven. Dergelijke algoritmen worden beschreven door systemen van de vorm $\dot{X} = F(X,t)$ waarbij $F(X,t) \in T_X\mathcal{M}$ voor elke $X \in \mathcal{M}$.

Hoofdstuk 3. Instrumenten voor het ontwerp en de analyse van tijdscontinue matrixalgoritmen

In dit werk leggen we de nadruk op het gebruik van formules voor het volgen van ontbindingen van een in de tijd variërende matrix. Deze formules spelen een centrale rol in dit werk als belangrijke instrumenten voor het ontwerp en de analyse van matrixalgoritmen. Daarnaast besteden we ook aandacht aan een meer klassiek middel, het gebruik van gradiënten.

Het volgen van matrixontbindingen en meetkundige interpretaties

De volgende lemma's geven formules voor de evolutie van de Cholesky-ontbinding, de QR-ontbinding en de eigenwaardenontbinding. We veronderstellen dat aan de nodige bestaans- en uniciteitsvoorwaarden is voldaan.

Lemma 0.1
Stel dat $X(t) \in \mathbb{R}^{n\times n}$ een continu differentieerbaar pad beschrijft.
(i) Als $C = C(t) = \mathsf{cf}(X(t))$ (Cholesky-factor), dan is

$$\dot{C} \;=\; \mathsf{D}\,\mathsf{cf}|_X(\dot{X}) \;=\; \mathsf{upph}(C^{-T}\dot{X}C^{-1})C,$$

(ii) Als $Q = Q(t) = \mathsf{qf}(X(t))$, $R = R(t) = \mathsf{rf}(X(t))$, (Q- en R-factor in een QR-ontbinding), dan is

$$\begin{aligned}\dot{Q} &= \mathsf{D}\,\mathsf{qf}|_X(\dot{X}) = Q\rho_{\mathsf{skew}}(Q^T\dot{X}R^{-1}),\\ \dot{R} &= \mathsf{D}\,\mathsf{rf}|_X(\dot{X}) = \rho_{\mathsf{upp}}(Q^T\dot{X}R^{-1})R,\end{aligned}$$

met upph, ρ_{skew} en ρ_{upp} zoals in definitie 2.9.

Lemma 0.2
(i) Stel dat het pad $X(t) \in \mathcal{S}_{symm}(n)$ een eigenwaardenontbinding

$$X(t) = P(t)\Lambda(t)P(t)$$

heeft met $P(t) \in \mathsf{O}(n)$ en $\Lambda(t) \in \mathcal{S}_{diag}(n)$ continu differentieerbaar. Dan is

$$\begin{aligned}\dot{\Lambda} &= \mathsf{diag}(M),\\ \dot{P} &= P\Omega,\end{aligned}$$

met $M = P^T \dot{X} P$, en waarbij Ω uit M volgt via

$$\begin{aligned}
\omega_{i,j}(t) &= 0 \\
&\quad \text{als } i = j, \\
\omega_{i,j}(t) &= m_{i,j}(t)/(\lambda_j(t) - \lambda_i(t)) \\
&\quad \text{als } \lambda_i(t) \neq \lambda_j(t), \qquad i = 1, \ldots, n \\
\omega_{i,j}(t) &= \lim_{\tau \to t} m_{i,j}(\tau)/(\lambda_j(\tau) - \lambda_i(\tau)) \qquad j = 1, \ldots, n. \\
&\quad \text{als } \lambda_i(t) = \lambda_j(t) \text{ voor een geïsoleerde waarde van } t, \\
\omega_{i,j}(t) &= -\omega_{j,i} = \text{ willekeurig} \\
&\quad \text{als } \lambda_i(\tau) = \lambda_j(\tau) \text{ voor alle } \tau \in [a,b] \text{ en } i \neq j,
\end{aligned}$$

Gelijkaardige resultaten worden bekomen voor de LU-ontbinding en de singuliere-waardenontbinding.

Deze formules kunnen in een meetkundige context geplaatst worden door matrixontbindingen te interpreteren als niet-lineaire coördinatentransformaties in de matrixruimte. Als aanschouwelijk voorbeeld wordt in figuur 3.4 de symmetrische eigenwaardenontbinding in $\mathcal{S}_{symm}(2)$ als een cylindrische coördinatentransformatie geïnterpreteerd.

Elke isospectrale manifold $\mathcal{I}_\Lambda$ (met alle matrices met gegeven eigenwaarden λ_1 en λ_2) is een cirkel met straal $\frac{1}{\sqrt{2}}|\lambda_1 - \lambda_2|$, gecenterd rond de verticaal afgebeelde symmetrieas $\mathbb{R}I$, die alle matrices met een dubbele eigenwaarde bevat. Een verzameling met alle matrices met constante gegeven eigenvectoren is een vlak door die symmetrieas. De snelheid van een matrix X kan ontbonden worden in een 'radiale' en een 'tangentiële' component, waaruit de evolutie van respectievelijk de eigenwaarden en de eigenvectoren kan worden afgeleid. In lemma 0.2 komt de transformatie $\dot{X} \to M = P^T \dot{X} P$ overeen met een dergelijke coördinatentransformatie.

Ook de QR-ontbinding kan als een veralgemeende poolcoördinatentransformatie geïnterpreteerd worden. Deze interpretaties liggen aan de basis van enkele verder besproken resultaten en zijn van nut voor inzicht in uitzonderingsgevallen voor matrices met herhaalde eigenwaarden of singuliere waarden.

Tenslotte vermelden we nog dat de meeste formules ook kunnen veralgemeend worden voor het geval van complexe matrices.

Gradiënten

Een Riemanniaanse manifold is een manifold waarvoor op elk punt x een inwendig product gedefinieerd is met een zacht ('smooth') verloop in functie van x. De gradiënt $\text{grad}_x f$ in een punt x van een differentieerbare functie f over een Riemanniaanse manifold $\mathcal{M}$ is de unieke raakvector in $T_x\mathcal{M}$ waarvoor

$$\dot{f} = \langle \text{grad}_x f, \dot{x} \rangle,$$

voor elk differentieerbaar pad $x(t)$ door x.

De 'gradient flow' van f wordt beschreven door

$$\dot{x} = \text{grad} f.$$

Onder ruime voorwaarden voor f en de initiële conditie voor x convergeert dit algoritme naar een locaal maximum van f. Bij het berekenen van gradiënten van functies die gedefinieerd zijn in termen van matrixontbindingen (b.v. canonieke hoeken tussen deelruimten) of over manifolds die verband houden met matrixontbindingen, kunnen ook de hoger beschreven resultaten nuttig aangewend worden.

Hoofdstuk 4. De 'QR flow' en de 'double bracket flow'

Twee 'klassieke' voorbeelden van tijdscontinue matrixalgoritmen zijn de 'QR flow' en de 'double bracket flow'. Beide algoritmen kunnen gebruikt worden voor de berekening van een eigenwaardenontbinding, en spelen ook een rol in de rest van dit werk.

De QR flow is een tijdscontinue versie van het bekende QR algoritme. Waar het QR algoritme, zoals bekend, in de opeenvolgende iteraties de Q-factor van A^k, $(k = 1, 2, \ldots)$ berekent, houdt de QR flow de Q-factor van e^{At} bij. De evolutie van deze Q-factor wordt bekomen door toepassing van lemma 0.1(ii). Voor de overeenkomstige evolutie van $E(t) = Q^T(t)AQ(t)$ vindt men de zeer eenvoudige vorm

$$\dot{E} = [E, \rho_{\text{skew}}(E)],$$

waarbij $[X, Y] = XY - YX$ en $\rho_{\text{skew}}(.)$, voor symmetrische E, de diagonaal van E op 0 zet en het deel onder de diagonaal van teken verandert. Door toepassing van lemma 0.1(i) kan ook een eenvoudige vorm bekomen worden die de Cholesky factor van E bijhoudt.

De double bracket flow wordt beschreven door

$$\dot{H} = [H, [H, N]],$$

waarbij H en N symmetrische matrices zijn. De matrix H is de toestand van het systeem, terwijl N het systeem parametriseert. Dit algoritme is isospectraal: H behoudt zijn eigenwaarden tijdens de evolutie van het systeem. Bovendien convergeert de matrix H naar een matrix met dezelfde eigenvectoren als N. Dit algoritme minimiseert de afstand van H tot N, terwijl H op een isospectrale manifold blijft. Indien N een diagonaalmatrix is, convergeert H naar een diagonaalmatrix met de eigenwaarden van $H(0)$ op de diagonaal.

Hoofdstuk 5. Een klasse van uniform parallelle tijdscontinue matrixalgoritmen

In dit hoofdstuk wordt een klasse van algoritmen gedefinieerd, die alsvolgt in verband staat met de hoofdthema's van dit werk.
1. Het gaat om tijdscontinue matrixalgoritmen die geanalyseerd en gemanipuleerd kunnen worden door gebruik te maken van de theorie van de hoofdstukken 2 en 3.
2. Verschillende algoritmen in de klasse zijn tijdscontinue limieten van systolische algoritmen. (Hoofdstukken 6, 7 en 8).
3. De algoritmen zijn veralgemeningen van Hebbiaanse adaptatiewetten voor neurale netwerken. (Hoofdstukken 9 en 10).

De klasse bevat algoritmen van de vorm $\dot{A} = \sum_{i=1}^{N} T_i(A, x, y)$, met toestand $A(t) \in \mathbb{R}^{n\times m}$ en mogelijke invoer $x(t) \in \mathbb{R}^n$ en $y(t) \in \mathbb{R}^m$. De termen $T_i(A, x, y)$ kunnen één van de volgende 3 vormen aannemen.
Termen van type I zijn van de vorm

$$T = cd^T.$$

Termen van type II zijn van de vorm

$$T = \mathrm{tri}(ce^T)A \text{ of } T = A\,\mathrm{tri}(df^T),$$

waarbij tri voor upp, upph, upps, low, lowh of lows staat (definitie 2.9).
Termen van type III zijn van de vorm

$$T = \mathrm{tri}(cd^TA^{-1})A \text{ of } T = A\,\mathrm{tri}(A^{-1}cd^T),$$

waarbij ofwel $A \in \mathcal{S}_{upp}(n)$ en tri voor upp, upph of upps staat, ofwel $A \in \mathcal{S}_{low}(n)$ and tri voor low, lowh of lows staat. De vectoren $c, e \in \mathbb{R}^n$ zijn van de vorm $(AA^T)^k x$ of $(AA^T)^k Ay$ en de vectoren $d, f \in \mathbb{R}^m$ zijn van de vorm $(A^TA)^k y$ of $(A^TA)^k A^T x$, met k geheel en $k \geqslant 0$.

Voor de algoritmen van deze klasse kan een eenvoudige uniform parallelle signaalgrafe worden afgeleid. Hiertoe worden de matrixelementen van A gerealiseerd in de cellen van een rechthoekig rooster. De componenten van de invoer x en de vectoren c en e worden gerealiseerd als signalen op horizontale lijnen. De componenten van y, d en f worden gerealiseerd als signalen op verticale lijnen.

De centrale eigenschap die eenvoudige parallelle realisatie mogelijk maakt, is dat door het berekenen van signalen als c, d, e, f, A^Tc, Ad, A^{-T} of $A^{-1}c$ uit A en x of y door middel van een voor de hand liggende implementatie van matrix-vectorvermenigvuldigingen (of terugsubstituties voor $A^{-T}d$ en $A^{-1}c$), elke cel (i, j) over voldoende informatie beschikt om $a_{i,j}$ aan te passen.

De benaming *uniform* parallel slaat op het feit dat alle cellen van de signaalgrafe identiek zijn.

Hoofdstuk 6. Tijdscontinue recursieve kleinste-kwadratenschatting

In dit hoofdstuk leiden we drie tijdscontinue algoritmen af, voor tijscontinue recursieve kleinste-kwadratenschatting. Dit probleem bestaat erin een gegeven signaal zo goed mogelijk te benaderen door een lineaire combinatie van andere gegeven signalen, en dit op adaptieve wijze terwijl voortdurend nieuwe informatie binnenkomt. Dit is een klassiek probleem met verscheidene toepassingen. We verwijzen naar [50].

De drie algoritmen worden afgeleid door toepassing van lemma 0.1 (of varianten, zie lemma 3.4). De drie algoritmen behoren tot de klasse van uniform parallelle algoritmen beschreven in hoofdstuk 5. Hun parallelle realisatie kan geïnterpreteerd worden als een neuraal netwerk (hoofdstuk 9) en als een tijdscontinue versie van gekende systolische algoritmen. Voor onderzoekers in neurale netwerken illustreren deze tijdscontinue algoritmen hoe eenvoudige locale adaptiewetten kunnen aanleiding geven tot globaal gedrag dat een klassiek adaptief signaalverwerkingsprobleem oplost. Voor onderzoekers in systolische algoritmen is de verrassend eenvoudige tijdscontinue limiet van systolische algoritmen van belang, en de eenvoudige manier waarop deze algoritmen kunnen worden afgeleid en gemanipuleerd. In hoofdstuk 8 wordt dit verband tussen systolische algoritmen en hun tijdscontinue versies benut voor het ontwerp van nieuwe tijdsdiscrete algoritmen. Tenslotte komen de algoritmen van dit hoofdstuk, dankzij hun eenvoudige signaalgrafe, in aanmerking voor analoge realisatie. Hiervan worden enkele hoogniveau-aspecten besproken.

Een mogelijke oplossing van het recursieve kleinste-kwadratenprobleem bestaat erin, naar analogie met het tijdsdiscrete geval, de Cholesky-factor van een tijdscontinue informatiematrix $N_c(t) = \int_0^t x(\tau)x(\tau)^T e^{-\lambda(t-\tau)}d\tau$ bij te houden, waarbij x een vector is met de invoersignalen. Daaruit kan dan de kleinste-kwadratenoplossing eenvoudig worden afgeleid. Door toepassing van lemma 0.1(i) vinden we een algoritme met een uniform parallelle realisatie zoals beschreven in hoofdstuk 5 (figuur 6.1).

$$\dot{R} = \text{upph}(R^{-T}xx^TR^{-1})R - \frac{\lambda}{2}R.$$

Dit algoritme is een tijdscontinue limiet van het bekende Gentleman-Kung systolisch algoritme [47].

Indien de inverse Cholesky factor wordt bijgehouden (opnieuw door toepassing van lemma 3.4), wordt een ander algoritme (algoritme 6.23) met een uniform

parallelle realisatie bekomen (figuur 6.4), dat een tijdscontinue versie is van het Moonen-McWhirter systolisch algoritme (figuren 6.5 en 6.6) [80]. Door het bijhouden van een LU-ontbinding in plaats van een Cholesky-ontbinding worden enkele varianten verkregen (algoritmen 6.19, 6.20 en 6.21) die enkele voordelen hebben voor analoge realisatie.

Hoofdstuk 7. Ontwerp en analyse van tijdscontinue deelruimtevolgers

In dit hoofdstuk bestuderen we het probleem van deelruimtevolgen, een ander adaptief signaalverwerkingsprobleem. Bij dit probleem is een signaal $x(t)$ gegeven, gegenereerd door $x(t) = Ms(t) + n(t)$, met $M \in \mathbb{R}^{n\times\kappa}$ een matrix van volle rang ($\kappa \leqslant n$), $s(t) \in \mathbb{R}^{\kappa}$ een bronsignaal met niet-singuliere correlatiematrix, en $n(t)$ een ruissignaal met $E\{nn^T\} = \sigma_n^2 I$. De bedoeling is de signaalruimte $\mathcal{S} = \text{col}(M)$ te schatten. We bestuderen een klasse van algoritmen, aangeduid met 'kruisprodukt-deelruimtevolgers'. We besteden vooral aandacht aan één algoritme van deze klasse, 'algoritme C'.

$$\dot{A} = \gamma x x^T A - 2A\,\text{upph}(A^T x x^T A).$$

Dit tijdscontinue algoritme is sterk verwant met verschillende tijdsdiscrete algoritmen zoals stochastische-gradiëntalgoritmen [86, 107, 85], sferische deelruimtevolgers[21], en andere [84, 105, 3, 108, 91, 68, 61]. Verder zijn ze verwant aan enkele autonome tijdscontinue algoritmen zoals de QR-flow, de double bracket flow en Riccativergelijkingen.

Algoritme C kan afgeleid worden naar analogie met de tijdscontinue kleinstekwadratenalgoritmen van hoofdstuk 6, door de eigenwaardenontbinding van de informatiematrix bij te houden. Dit levert een complexer algoritme, maar door een benadering die het tijdscontinu equivalent is van de idee achter sferische deelruimtevolgers, wordt algoritme C bekomen. Dit algoritme behoort tot de algoritmen van hoofdstuk 5 met een uniform parallelle realisatie (figuur 7.1).

De theorie en de meetkundige interpretaties van hoofdstuk 3 zijn ook van nut voor de *analyse* van algoritme C (en andere kruisprodukt-deelruimtevolgers). Een eerste aspekt hierbij is de analyse van de evolutie van de schatting van de signaalruimte. Dit gebeurt door de evolutie te analyseren van de canonieke hoeken (gedefinieerd op basis van een singuliere-waardenontbinding) tussen de geschatte en de te schatten deelruimte. Er wordt een eenvoudig interpreteerbare differentiaalvergelijking voor de evolutie van de hoeken bekomen. In het geval van ruisvrije invoer en een stilstaande deelruimte, kan aan de hand van deze differentiaalvergelijkingen worden aangetoond dat de geschatte deelruimte onder ruime voorwaarden naar de te schatten deelruimte convergeert.

Een tweede aspekt van de analyse is de aantrekking tot de verzameling van orthogonale matrices. Met een niet-orthogonale initiële matrix $A(0)$, evolueert A toch tot een orthogonale matrix (d.i. met orthogonale kolommen). Dit wordt bewezen door een analyse van de evolutie van de singuliere waarden van A.

Hoofdstuk 8. Afleiding van tijdsdiscrete systolische deelruimtevolgers

In dit hoofdstuk leiden we een tijdsdiscrete systolische versie van algoritme C (hoofdstuk 7) af, algoritme D genaamd. Dit algoritme heeft een signaalgrafe die gelijkaardig is aan die van algoritme C (figuren 8.1 en 8.2). De afleiding is geïnspireerd door het verband tussen systolische algoritmen voor kleinste-kwadratenschatting en hun tijdscontinue versies, zoals beschreven in hoofdstuk 6. Algoritme D wordt bekomen door exacte integratie van algoritme C voor een stuksgewijs constant invoersignaal. Het bekomen tijdsdiscrete algoritme kan horizontaal 'gepipelined' worden, en ook verticaal (figuren 8.5 en 8.6). Dit laatste gebeurt opnieuw naar analogie met het Moonen-McWhirter systolisch algoritme, maar geeft wel aanleiding tot een complexere structuur.

Algoritme D erft de belangrijkste eigenschappen van algoritme C, zoals besproken in hoofdstuk 7, waaronder de aantrekking tot een orthogonale matrix en de analyse van de evolutie van de geschatte deelruimte. Als voordelen ten opzichte van stochastische-gradiëntalgoritmen vermelden we de goedkope orthogonalisatie (exacte herorothogonalisatie indien vertrokken wordt van een orthogonale matrix) en de mogelijkheid van realisaties met hoge doorvoer. Een voordeel ten opzichte van Oja's neuraal stochastische-gradiëntalgoritme is ook de mogelijkheid van willekeurig grote stapgrootten. Met een daartoe gekozen stapgrootte, levert algoritme D dezelfde deelruimteschattingen als de oorspronkelijke sferische deelruimtevolger. Experimenten tonen aan dat er nog ruimte is voor betere keuzes van de stapgrootte.

Verder wordt ook nog een variant afgeleid, gebaseerd op de LU ontbinding van A. Hiermee wordt aangetoond dat de afleiding van de systolische versie van algoritme D geen alleenstaand geval is. Een automatische methode om parallelle tijdsdiscrete algoritmen af te leiden kunnen we echter niet geven.

Hoofdstuk 9. Tijdscontinue adaptatiewetten voor neurale netwerken voor adaptieve signaalverwerking

In dit en het volgende hoofdstuk bestuderen we neurale adaptiewetten binnen het kader van tijdscontinue matrixalgoritmen. In dit hoofdstuk geven we neurale interpretaties aan de algoritmen van hoofdstukken 5 tot 8. In hoofdstuk 10 bekijken we adaptatiewetten voor recurrente neurale netwerken voor geheugentoepassingen.

Twee algemene ideeën werden ontleend aan de biologie. Een eerste idee is dat complex gedrag kan bekomen worden door een aantal gelijkaardige, eenvoudige elementen (neuronen) te laten interageren. Neuronen wegen en sommeren informatie van andere neuronen en zenden het resultaat door naar andere neuronen. Het tweede idee is dat de gewichten, waarmee de neuronen de inkomende informatie wegen, tijdens het gebruik evolueren onder invloed van locale informatie. Het is niet de bedoeling dichter bij biologische netwerken aan te sluiten dan deze algemene gelijkenissen. Onze interesse in neurale netwerken ligt in het bereiken van interessant globaal gedrag vanuit eenvoudig locaal gedrag in neurale netwerken door middel van uniforme locale adaptatiewetten. Met uniform bedoelen we dat vershillende elementen dezelfde (of een beperkt aantal) wetten volgen. Met locale adaptatiewet bedoelen we een wet die op het netwerk zelf kan gerealiseerd worden.

Wetten met termen van type I (hoofdstuk 5) kunnen geïnterpreteerd worden als eenlagige lineaire voorwaartse netwerken met een Hebbiaanse adaptatiewet (figuur 9.5). Wetten van type II zijn hieraan verwant maar gebruiken locale tussentijdse resultaten van de sommeringen in de neuronen, in plaats van de volledige som om gewichten aan te passen (figuur 9.6). Wetten van type III vereisen daarenboven nog enkele andere afwijkingen zoals een deling in de neuronen. Figuur 9.7 toont een neurale-netwerkversie van het tijdscontinue Gentleman-Kung-algoritme. Figuur 9.8 toont een netwerkstructuur die kan gebruikt worden voor LCMV-bundelvorming (een techniek die gebaseerd is op recursieve kleinstekwadratenschatting) en ook voor een variant van algoritme C.

Hoofdstuk 10. Tijdscontinue adaptatiewetten voor neurale netwerken met geheugentoepassingen

We beschouwen drie verwante, volledig geïnterconnecteerde, recurrente neurale netwerkmodellen. Deze modellen kunnen als robuust patroongeheugen gebruikt worden. De stabiele equilibria van het netwerk zijn opgeslagen patronen. De

drie bestudeerde modellen zijn geschikt om binaire patronen op te slaan. Een patroon kan opgeroepen worden door een geperturbeerd (of onvolledig) patroon aan te leggen als initiële toestand en het netwerk te laten evolueren naar één van zijn stabiele toestanden. Het gedrag van het netwerk wordt bepaald door de gewichten van de verbindingen. Als de gewichtmatrix symmetrisch is, convergeert het netwerk naar een evenwicht vanuit gelijk welke initiële conditie. Een belangrijk probleem bestaat erin een geschikt stel gewichten te vinden om een gegeven verzameling patronen in het geheugen op te slaan. Een ontwerpalgoritme is een algoritme dat een dergelijk stel gewichten berekent. We spreken van een adaptatiewet als het algoritme een geleidelijke adaptatie van de gewichten voorschrijft, gestuurd door een leersignaal dat de patronen aangeeft.

We bestuderen een niet-autonome variant van de double bracket flow.

$$\dot{H} = [H, [H, xx^T]],$$

waarbij $x \in \mathbb{R}^n$ een tijdsafhankelijk vectorsignaal is. Deze wet is van type I (hoofdstuk 5) en kan gerealiseerd worden op het netwerk, waarbij H overeenkomt met de gewichtsmatrix. Als de initiële gewichtsmatrix $H(0)$ een m-voudige grootste eigenwaarde $\rho > 1$ heeft en de invoer x een m-dimensionale deelruimte $\mathcal{S}$ beschrijft, convergeert de principale eigenruimte van H naar $\mathcal{X}$ (terwijl de eigenwaarden van H behouden blijven). Omdat binaire patronen die voldoende dicht bij een eigenvector met eigenwaarde groter dan 1 komen te liggen, stabiele patronen van het geheugen worden, kan dit systeem gebruikt worden om binaire patronen in een deelruimte beschreven door x in het geheugen op te slaan. De matrix H convergeert naar een instelling die overeenkomt met een gekende (niet-adaptieve) ontwerpregel [72]. Door de aanwezigheid van sterk negatieve eigenvectoren kan ook het aantal ongewenste opgeslagen patronen beperkt worden.

Bij de hoger beschreven wet worden afwijkingen van het spectrum (b.v. door een benaderende realisatie) niet gecorrigeerd. Het spectrum kan gestabiliseerd worden door extra termen toe te voegen aan de wet. Mits invoering van een extra willekeurig invoersignaal (dat niet naar een deelruimte convergeert) kan deze stabilisatie opnieuw met termen van type I gebeuren. Tenslotte is het ook mogelijk de leerfaze en de operatiefaze van het geheugen te combineren.

Hoofdstuk 11. Optimisatie-algoritmen voor de berekening van een ondergrens voor de gestructureerde singuliere waarde

In dit laatste hoofdstuk worden gradiëntgebaseerde optimisatie-algoritmen over Lie-groepen van gestructureerde unitaire matrices beschreven voor de berekening

van een ondergrens voor de gestructureerde singuliere waarde, aangeduid met μ. Dit is een belangrijk instrument voor de robuustheidsanalyse van systemen met gestructureerde onzekerheden.

Het optimisatieprobleem kan alsvolgt geformuleerd worden.

$$\mu_{\mathbf{\Delta}}(M) = \max_{Q \in \mathcal{Q}} \rho(QM),$$

waarbij ρ de spectrale radius noteert en $\mathcal{Q}$ de groep van unitaire matrices met de structuur van de onzerkerheid, d.w.z blokdiagonale unitaire matrices met vaste blokgrootten en waarvan sommige blokken slechts (complexe) veelvouden van de eenheidsmatrix mogen zijn.

Het bestaande PIA algoritme (Power Iteration Algorithm) [87] is een iteratieve methode die in geval van convergentie aan de optimaliteitsvoorwaarden voldoet. Er bestaan echter gevallen waarin het algoritme niet convergeert en een triviale ondergrens als antwoord geeft. We geven nieuwe algoritmen die minder efficiënt zijn maar waarvoor ook de intermediaire resultaten kunnen geïnterpreteerd worden. De nieuwe algoritmen zijn gradiëntgebaseerde afdalingsmethoden (de methode van de steilste helling en de toegevoegde-gradiëntenmethode). Beide methoden houden met de beperking van het optimisatieprobleem rekening door op de Lie-groep $\mathcal{Q}$ te blijven. Waar in de gewone Euclidische versie van deze methoden lijnzoekalgoritmen langs rechte lijnen uitgevoerd worden bij iedere stap, worden deze nu vervangen door zoekalgoritmen langs geodeten op $\mathcal{Q}$. Raakvectoren in een punt worden aangepast voor gebruik in een ander punt door parallelle translatie [94]. Bij de afleiding van formules voor de gradiënt van de doelfunctie werd opnieuw gebruik gemaakt van concepten uit hoofdstuk 3.

Daarnaast worden ook tijdscontinue versies van deze algoritmen beschouwd. Opnieuw worden algoritmen bekomen die zich lenen tot parallellisatie. In dit geval komen we echter niet tot parallelle discrete versies van deze algoritmen.

Hoofdstuk 12. Conclusie

Tijdscontinue matrixalgoritmen bieden, aan wie dit apprecieert, een inspirerende meetkundige kijk op het ontwerp, de analyse en de parallelle realisatie van matrixalgoritmen, vaak met neurale netwerkinterpretaties. Hoewel analoge realisaties slechts realistisch lijken in eenvoudige gevallen en de omzetting in tijdsdiscrete algoritmen niet a priori voor de hand ligt, zijn hierbij verschillende aanpakken mogelijk en werden enkele succesvolle voorbeelden gegeven. We hebben getoond hoe via de tijdscontinue benadering enkele klassieke adaptieve signaalverwerkingsoperaties en leerregels voor associatief geheugen kunnen gevat worden in een klasse van locale adaptatiewetten voor neurale netwerken. Als meer concrete resultaten biedt dit werk een tijdscontinu algoritme voor recur-

sieve kleinste-kwadratenschatting, dat in aanmerking komt voor analoge realisatie, een nieuw efficiënt tijdsdiscreet systolisch algoritme voor deelruimtevolgen en een nieuw tijdsdiscreet algoritme voor de berekening van gestructureerde singuliere waarden.

Chapter 1

Introduction

The unifying theme of this work is the theory of **continuous-time matrix algorithms**. A number of recent papers have considered matrix problems and algorithms from the viewpoint of *continuous-time systems theory and differential geometry*. We will refer to this viewpoint as the 'continuous-time approach'. Next to offering an elegant framework, shedding new light on known algorithms, and often unifying seemingly different results, this approach has also inspired new theoretical results and new discrete-time algorithms.

We give an *introduction* to 'continuous-time matrix algorithms', we present *new contributions* with applications in adaptive signal processing, robust control, and neural network associative memories, and we try to *evaluate* the merits and the limitations of the approach, on the basis of the presented material.

One of the contributions of this work is to establish a link between continuous-time algorithms and discrete-time **systolic algorithms**. We show how known systolic arrays have simple continuous-time counterparts, with neural network interpretations. Exploiting the natural parallelism of some continuous-time algorithms, we also derive new systolic algorithms.

An important motivation for this work, was the study of **adaptation laws for neural networks**. Since by making generalizations our work has gradually grown out of this field, most results are first presented in a more general context, and only afterwards reinterpreted in the context of neural networks.

Section 1.1, gives a general overview of this work. Section 1.2 gives a chapter by chapter overview.

1.1 General overview

Figure 1.1 summarizes the main ideas from a theoretical point of view, and considers four aspects of the central topic of continuous-time matrix algorithms.

The *first* aspect is the use of the continuous-time approach, for the design of both continuous-time and discrete-time algorithms. For all applications considered, we will *derive several continuous-time algorithms*. Basic tools are the use of gradients of functions defined over a manifold, and the use of derivatives of linear algebra related maps, like matrix decompositions, to design, transform, and analyze algorithms.

Although the *analog realization* of these algorithms is appealing and has received much recent interest for instance in the area of analog adaptive signal processing, it is only feasible for simple algorithms. We will discuss high-level aspects of the analog realization of recursive least-squares algorithms.

However, continuous-time algorithms can also be useful for the *design of discrete-time algorithms*. A new contribution of this work lies in the design of *systolic algorithms* through the continuous-time approach. Many continuous-time algorithms turn out to have simple uniform parallel signal flow graphs (consisting of simple, often identical cells). We show how some of these algorithms are continuous-time limits of systolic algorithms and derive new systolic algorithms, inheriting properties from their continuous-time counterparts. However there is no automatic method to turn continuous-time algorithms and their simple signal flow graphs into parallel discrete-time algorithms.

Next to the design of parallel algorithms, the continuous-time view is useful for the design of *discrete-time optimization algorithms*, especially with regular equality constraints. To define algorithms that 'live' in a space defined by an equality constraint, differential geometric concepts are useful, and the continuous-time view is never far away. Moreover continuous-time gradient algorithms can be helpful for exploratory experiments, as often they are much easier to program (using standard integration packages) than discrete-time algorithms that preserve the constraint.

The *second* aspect is the analysis of algorithms. Concepts from differential geometry and systems theory can be helpful in many ways for the analysis of algorithms, like the analysis of the stability equilibrium points and the corresponding convergence behavior. We concentrate on the analysis of adaptive signal processing algorithms. We prove results on *convergence* and *tracking performance* by using continuous-time tools such as singular value tracking and canonical angle tracking. Similar ideas are also applied to the analysis of neural network adaptation laws for memory applications.

As a *third* aspect, we claim that the continuous-time approach is an insightful

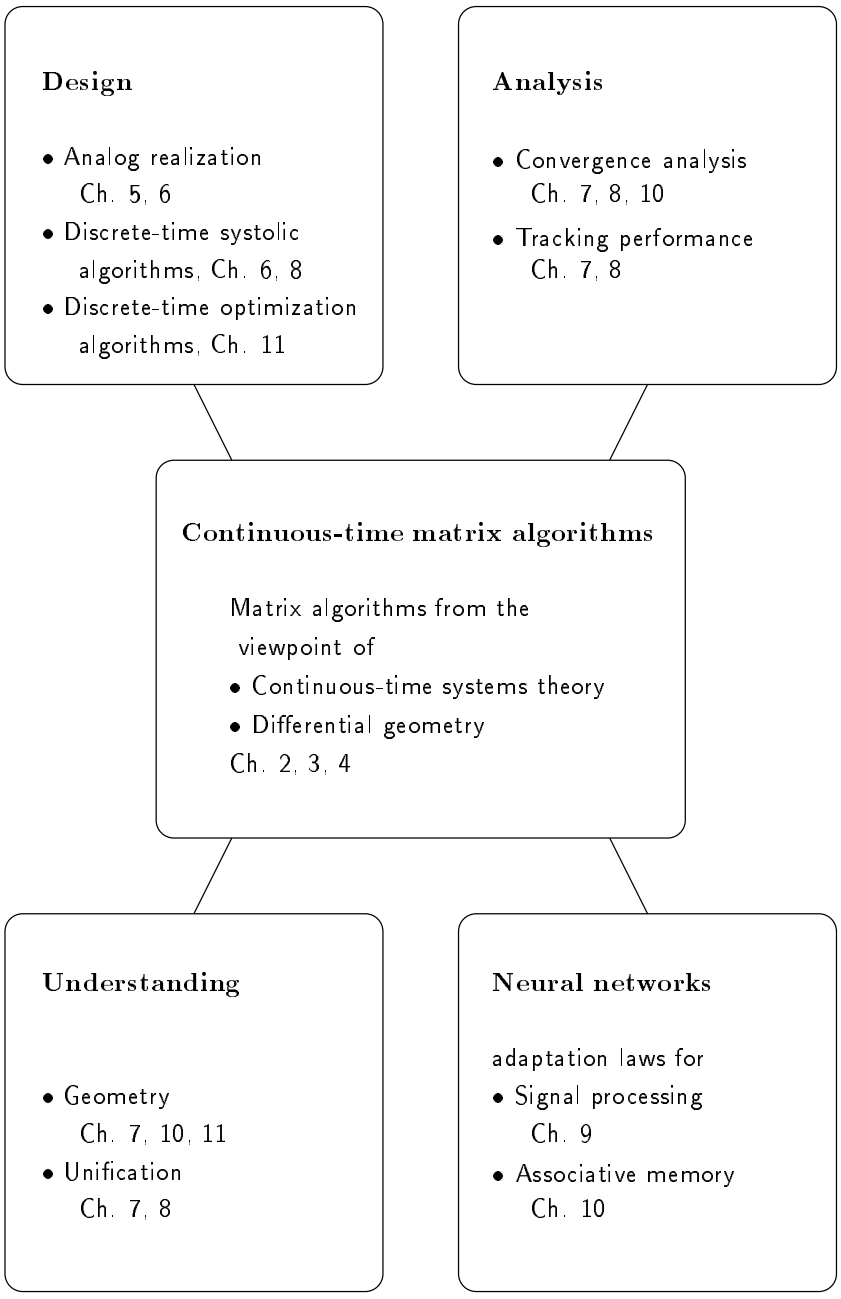


Figure 1.1: General overview of the main ideas of this work.

approach, leading to the *understanding* of algorithms and to inspiration in their design and analysis. We put a special emphasis on geometric interpretations. The appreciation of this more subjective aspect can of course depend on personal taste. Nevertheless it is clear, that continuous-time algorithms and their derivations often take surprisingly different and simpler forms than their discrete-time counterparts. And the mere fact that the viewpoint is different can only foster inspiration. The continuous-time viewpoint often offers a *unifying* viewpoint for different algorithms. We will see how seemingly different algorithms for subspace tracking take the same form in the continuous-time limit.

As a *fourth* aspect we consider neural network interpretations. As mentioned above, the study of adaptation laws for artificial neural networks has been a motivation since the beginning of our work, but as our research work has grown out of this field, we present some of these results as special cases and other interpretations.

The general idea borrowed from the field of neural networks is the following way of computing with a dynamical system. On a *local* scale (neurons, weights), the system shows simple and often uniform behavior, all elements evolving by the same law (or a small number of different laws) using information available through the network. On a *global* scale this results in some interesting behavior such as memory, control, or various types of information processing. One of the main problems is that the relation between even the simplest local behavior and the resulting global behavior can be very complex. In this work we study a class of systems, which can be realized with with simple and uniform local behavior on a network, and at the same time allow for a rigorous mathematical study of the global behavior. We consider applications in adaptive signal processing and neural associative memory.

Finally, figure 1.2 gives a general overview from the applications point of view. This corresponds more to the division into chapters, as discussed in the next section. The applications considered are diverse. It is impossible to go into details for all applications, and we will avoid deviating too far from the main topic of this work, being the application of ideas from continuous-time systems theory and differential geometry to the design and analysis of algorithms. To support the relevance of this viewpoint, in chapters 8, 10 and 11 we also pay attention to issues outside these domains.

1.2 Chapter by chapter overview

Figure 1.3 shows how the different chapters can be grouped together.

Chapters 2 to 4 are an introduction to continuous-time matrix algorithms. Although most of the results in this introduction are not new, we give some new

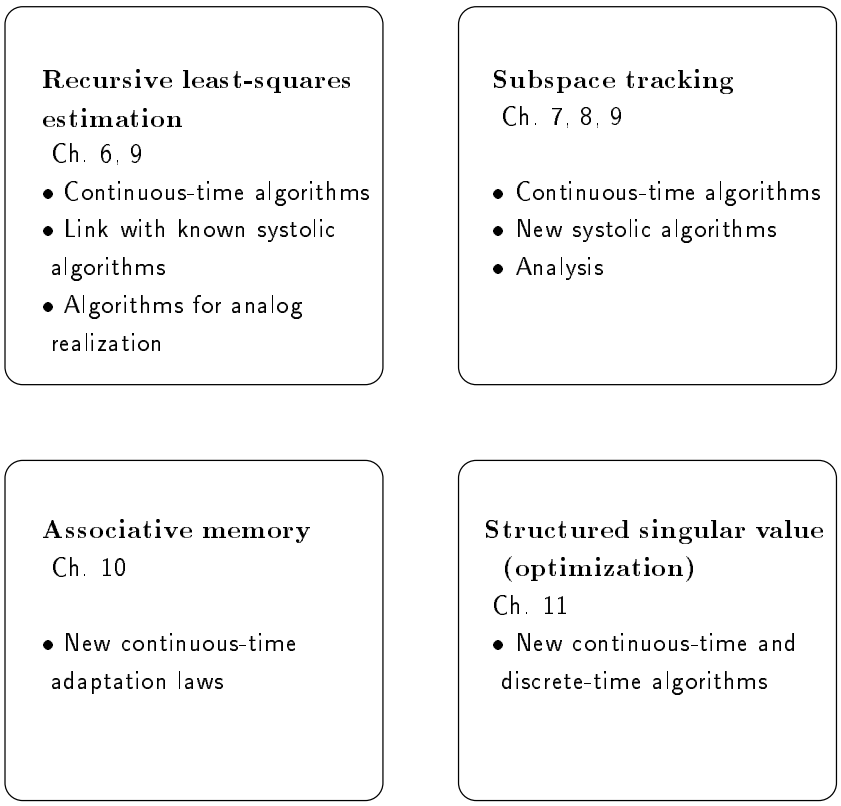


Figure 1.2: Overview from the applications point of view.

interpretations and put a special emphasis on the results of section 3.1. The organization of the text should allow readers who are only interested in concrete results of later chapters, to skip the introduction and only use results as they are needed.

Chapter 2 introduces the basic concepts. We give a more precise content to the notion of 'continuous-time algorithms'. We summarize some definitions and properties of matrix space as a Euclidean vector space, and we consider important manifolds in matrix space.

Chapter 3 introduces the basic tools for the design and analysis of matrix algorithms. We put the emphasis on a number of lemmas for tracking matrix decompositions, and give a geometric interpretation, considering matrix decompositions as nonlinear coordinate transformations of matrix space. Another important tool is the use of gradients of functions over Riemannian manifolds. We give definitions and examples and show the relevance of the decomposition

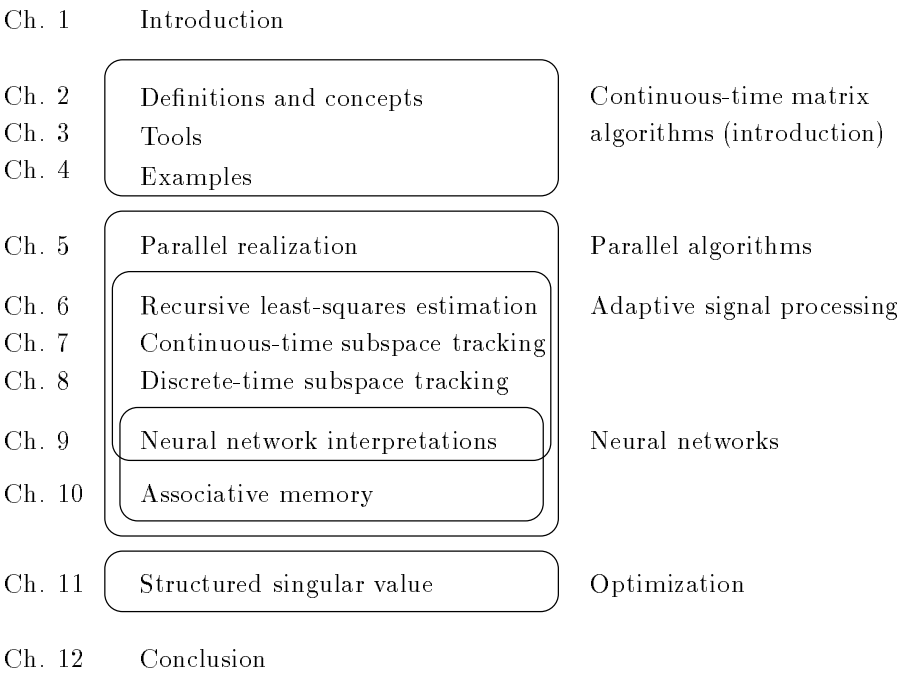


Figure 1.3: Chapter by chapter overview

tracking lemmas for deriving gradients. In this chapter we also consider generalizations to the case of complex matrices.

Chapter 4 discusses the QR flow and the double bracket flow. These are two continuous-time algorithms that have played a central part in the evolution of the field, and are also relevant in further chapters. We also introduce a new square root version of the QR flow.

Chapter 5 is a short chapter, describing a class of continuous-time matrix algorithms, with a uniform parallel signal flow graph that can be easily derived from the formal description of the algorithm. The main continuous-time algorithms of chapters 6 to 10 fit in this class. These algorithms have neural network interpretations (chapter 9) and some of them are continuous-time limits of systolic algorithms (chapters 6 and 8).

Chapters 6 to 9 consider adaptive signal processing applications. The signal processing operations considered are classical ones, with many well known applications.

Chapter 6 considers continuous-time recursive least-squares estimation. Three algorithms are derived using the tools of chapter 2 for matrix decomposition

tracking. The obtained algorithms fit in the class introduced in chapter 5, and can be interpreted as continuous-time versions of systolic algorithms. We also consider high-level aspects of analog realization. Neural network interpretations are discussed in chapter 9. The relations between these continuous-time algorithms and their systolic counterparts will be exploited to derive new systolic algorithms in chapter 8.

Chapter 7 considers the problem of subspace tracking. A class of continuous-time algorithms is considered, containing continuous-time versions of a number of known discrete-time algorithms. Three continuous-time algorithms fit in the class defined in chapter 5, and have a simple signal flow graph. Special attention is paid to one of them. A new analysis is given of the convergence and tracking behavior, using the tools of chapter 3 to track singular values and canonical angles between subspaces.

In chapter 8 one of the continuous-time subspace tracking algorithms of chapter 7 is converted into a discrete-time algorithm with a similar signal flow graph, that serves as a starting point for the derivation of high throughput systolic algorithms, inheriting the main properties of the continuous-time algorithms. The systolic realizations obtained are compared with existing related discrete-time algorithms.

Chapters 9 and 10 consider continuous-time algorithms as a framework for studying neural network adaptation laws.

Chapter 9 summarizes some elementary concepts of neural networks and discusses neural interpretations for the algorithms of chapters 5 to 7, and some related new ones.

In chapter 10 we consider continuous-time adaptation laws for recurrent neural networks, used as associative memories. We derive new adaptation laws which can be implemented on the network. They store patterns in the memory that are shown by a teaching signal. In an appendix, three closely related models of neural networks for associative memory applications are compared.

In **chapter 11** we derive continuous-time and discrete-time gradient based optimization algorithms for calculating lower-bounds of the structured singular value of a complex matrix. We derive a steepest ascent algorithm and a conjugate gradient algorithm, that live in a manifold of structured unitary matrix. Gradients are obtained using the theory of chapters 2 and 3. We compare our algorithm with an existing algorithm and illustrate our results by numerical experiments.

Finally, in **chapter 12** we conclude by summarizing our experience, trying to assess the merits and limitations of the continuous-time approach on the basis of the presented material. We also discuss open problems and further work and summarize the main contributions of this work.

Chapter 2

Definitions and concepts of continuous-time matrix algorithms

Chapters 2, 3 and 4 form an introduction to the field of continuous-time matrix algorithms. In this chapter, we introduce basic concepts of continuous-time algorithms and matrix space as a Euclidean vector space. In chapter 3, we discuss tools for the design and analysis of continuous-time algorithms, and give geometric interpretations. In chapter 4, we discuss the QR flow and the double bracket flow, two examples of autonomous continuous-time algorithms for eigenvector/eigenvalue calculation.

Continuous-time algorithms are continuous-time dynamical systems, whose behavior is interpreted as an algorithm. By attaching the label *matrix* algorithms, we focus on systems whose state (or part of it) is interpreted as a matrix. We do not consider interior point methods and homotopy methods, which are also often studied from a continuous-time viewpoint ([4, 5, 71, 45]).

A number of recent papers have shown that a continuous-time approach sheds new light on algorithms for problems in linear algebra and related fields. Without the intention to be complete we cite some references. For an overview and more references we refer to the recent book of Helmke and Moore [55]. One of the important early impulses in the field has come from work on the relation between the QR algorithm and isospectral flows by Symes, Deift, Nanda, Tomei, Chu, Watkins and others [96, 38, 15, 102, 82]. A (1988) survey article was written by Chu [16]. Another important impulse has come from the work of Brockett, Bloch, Ratiu, Chu, Driessel and others [8, 9, 11, 7, 17]. In [12] a more general

picture is sketched of the design of algorithms for numerical analysis in a Lie group theoretical framework. A very recent survey article is [54]. For a more general background on the differential geometric approach to dynamical systems theory we refer to Chillingworth [14]. For a standard work on Lie groups we refer to Helgason [52]. We also recommend Sattinger and Weaver [92]. An account, oriented towards continuous-time algorithms, is found in Helmke and Moore [55].

As this text is written for engineers and applied mathematicians, and since we want to convince the reader of the simplicity of the basic ideas, we have tried to avoid as much as possible the notations and jargon of differential geometry. Instead, we try to describe the main ideas in terms of linear algebra and calculus, with much emphasis on geometric interpretations, using such simple concepts as subspaces and circles. We have included proofs of known results if they can clarify the explained ideas. Furthermore, in several remarks we briefly mention more technical facts and terminology, in order to provide a link with the existing literature.

The choice of examples is oriented toward the contributions of later chapters, and the discussion of the QR flow and the double bracket flow in chapter 4.

This chapter is organized as follows. In section 2.1 we define continuous-time matrix algorithms. Section 2.2 stresses features of $\mathbb{R}^{n\times n}$ as a Euclidean space, and introduces some definitions. Section 2.3 discusses manifolds in $\mathbb{R}^{n\times n}$ that play a role in this work and briefly places them in the broader theoretical framework of Lie groups and homogeneous spaces. Section 2.4 extends the concept of continuous-time matrix algorithms to algorithms on manifolds.

2.1 Continuous-time matrix algorithms

The algorithms discussed in chapter 4 are *autonomous* continuous-time matrix algorithms.

Definition 2.1 (Autonomous continuous-time matrix algorithm)
An *autonomous continuous-time matrix algorithm* is a system

$$\begin{aligned} \dot{A}(t) &= F(A(t)), \\ A(0) &= A_0, \end{aligned} \quad \text{where } A(t) \in \mathbb{R}^{n\times m} \text{ and } F : \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times m}. \tag{2.1}$$

This is an autonomous dynamical system with a matrix state. The variable t denotes time. $\dot{A}$ denotes the time derivative of A. The input is specified by the initial condition A_0. For all autonomous algorithms in this text, the state $A(t)$ of the dynamical system converges to some fixed matrix A_∞, which is defined to be the output of the algorithm. We will not explicitly denote time dependence,

except when necessary to avoid ambiguity. Also the initial condition will not always be denoted explicitly. This results in the more concise form

$$\dot{A} = F(A). \tag{2.2}$$

In a more general framework, we consider *non*autonomous continuous-time matrix algorithms

Definition 2.2 (Nonautonomous continuous-time matrix algorithm)
A *nonautonomous continuous-time matrix algorithm* is a system of the form

$$\begin{array}{rcll} \dot{A}(t) & = & F(A(t), u(t), t), & A(0) = A_0, \\ y(t) & = & G(A(t)), & \end{array}$$

where $A(t) \in \mathbb{R}^{n \times m}$, $u(t) \in \mathbb{R}^p$, $y(t) \in \mathbb{R}^q$,
$F : \mathbb{R}^{n \times m} \times \mathbb{R}^p \times \mathbb{R} \to \mathbb{R}^{n \times m}$,
$G : \mathbb{R}^{n \times m} \to \mathbb{R}^q$.

The input is now specified by the initial condition $A_0 \in \mathbb{R}^{n \times m}$ and by $u(t)$. The output of the algorithm is $y(t)$.

Nonautonomous continuous-time algorithms can be regarded as continuous-time adaptive filters. We will study such systems in chapters 6 and 7.

An intuitive way to think of an *autonomous* continuous-time algorithm is to consider it as a continuous-time analogue of a discrete-time iterative algorithm

$$A^{[k+1]} = \tilde{F}(A^{[k]}), \tag{2.3}$$

where the index $^{[k]}$ denotes discrete-time. If no confusion is possible we will often drop the discrete-time index $^{[k]}$ and use a prime ($'$) for $^{[k+1]}$, giving $A' = \tilde{F}(A)$.

Writing the continuous-time algorithm (2.2), in the form

$$dA = F(A)dt,$$

intuitively, one can think of it as taking infinitely many infinitely small steps. Figure 2.1 gives a schematic picture.

Remark 2.3 (matrix flows)
In the literature, autonomous continuous-time matrix algorithms are sometimes referred to as *matrix flows*. Here we will use that name for the well known double bracket flow and QR flow, and for gradient flows. In other cases, we will prefer the term 'continuous-time algorithm'. Technically speaking, a flow on some subset $U \subseteq \mathbb{R}^{n \times n}$ is a one parameter family of transformations $\Phi_t : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}$, that form a group:

$$\begin{array}{rcl} \Phi_{t_1} \circ \Phi_{t_2} & = & \Phi_{t_1 + t_2}, \\ \Phi_t^{-1} & = & \Phi_{-t}. \end{array}$$

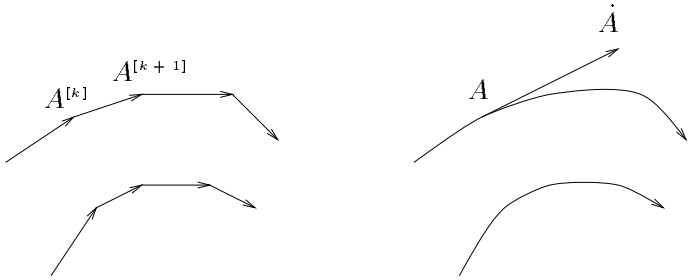


Figure 2.1: Schematic picture of a discrete-time iterative algorithm (left) and an autonomous continuous-time algorithm (right).

A flow Φ_t is said to solve the system $\dot{A} = F(A)$, if

$$A(t) = \Phi_t(A_0)$$

satisfies the system equation, that is, if

$$\frac{\partial}{\partial t}\Phi_t(A_0) = F(\Phi_t(A_0)).$$

This approach stresses the overall behavior of the system, rather than single trajectories for a given initial value.

Remark 2.4 (existence and uniqueness)
For most algorithms considered in this work, the existence and uniqueness of solutions to the differential equations will not be a big issue. Local existence will be ensured for almost all algorithms by the continuous differentiability of F. For nonautonomous systems we will also consider cases with piecewise continuous differentiability with respect to t, for which trajectories can be considered as concatenations of trajectories of systems with continuously differentiable F. Local existence of solutions does not yet exclude that the state $A(t)$ escapes to infinity (or off a noncompact manifold - see section 2.3) in finite time. Typically this will be prevented by the fact that the algorithm is defined on a compact manifold, or by the existence of a Lyapunov function with compact sublevel sets. For a more extensive discussion on existence and uniqueness of solutions of differential equations, we refer to [14, 57].

2.2 Matrix space as a Euclidean vector space

In this section we briefly recapitulate some features of matrix space $\mathbb{R}^{n\times m}$ as an nm-dimensional Euclidean vector space. We introduce some notations, define the standard inner product, and put emphasis on the linear nature of some operations and subsets.

The matrix space $\mathbb{R}^{n\times m}$ is identified with $\mathbb{R}^{nm}$ through the matrix reshaping operation $\mathrm{vec}(A)$, stacking the columns of A in one big column vector.

Definition 2.5

$$\mathrm{vec}(A) = \mathrm{vec}(\begin{bmatrix} A_{\cdot,1} & A_{\cdot,2} & \dots & A_{\cdot,m} \end{bmatrix}) = \begin{bmatrix} A_{\cdot,1} \\ A_{\cdot,2} \\ \dots \\ A_{\cdot,m} \end{bmatrix}.$$

The matrix space $\mathbb{R}^{n\times m}$ inherits the following standard basis and standard inner product from R^{nm}.

Definition 2.6 (Standard basis)
The *standard basis* for $\mathbb{R}^{n\times m}$ consists of the matrices $E^{(i,j)}$, $i = 1,\dots,n$, $j = 1,\dots,m$, defined by

$$E^{(i,j)}{}_{k,l} = \delta_{i,k}\delta_{j,l}.$$

Definition 2.7 (Standard inner product, norm)
The standard *inner product* of matrices $A, B \in \mathbb{R}^{n\times m}$ is defined by

$$\begin{aligned} \langle A, B\rangle &= \mathrm{vec}(A)^T\,\mathrm{vec}(B) \\ &= \textstyle\sum_{i=1}^{n}\sum_{j=1}^{m} a_{i,j}b_{i,j} \\ &= \mathrm{Tr}(A^TB) = \mathrm{Tr}(B^TA) \\ &= \mathrm{Tr}(AB^T) = \mathrm{Tr}(BA^T). \end{aligned}$$

The corresponding norm is the *Frobenius norm*

$$\|A\|_F = \sqrt{\langle A, A\rangle}.$$

Some important subspaces of $\mathbb{R}^{n\times n}$ are

Definition 2.8 (Subspaces)

$$
\begin{array}{lcl}
\mathcal{S}_{symm}(n) & = & \{X \in \mathbb{R}^{n\times n} | X = X^T\}, \\
 & & \text{symmetric matrices,} \\
\mathcal{S}_{skew}(n) & = & \{X \in \mathbb{R}^{n\times n} | X = -X^T\}, \\
 & & \text{skew symmetric matrices,} \\
\mathcal{S}_{upp}(n) & = & \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i > j, i, j = 1, \ldots, n\}, \\
 & & \text{upper triangular matrices,} \\
\mathcal{S}_{upps}(n) & = & \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i \geqslant j, i, j = 1, \ldots, n\}, \\
 & & \text{strictly upper triangular matrices,} \\
\mathcal{S}_{diag}(n) & = & \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i \neq j, i, j = 1, \ldots, n\}, \\
 & & \text{diagonal matrices,} \\
\mathcal{C}_A & = & \{X \in \mathbb{R}^{n\times n} | XA - AX = 0\}, \\
 & & \text{matrices commutable with } A \in \mathbb{R}^{n\times n}.
\end{array}
$$

Some linear transformations, that will be frequently used, are

Definition 2.9 (Linear transformations)

$$
\begin{array}{ll}
\text{id}: & \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times m} : X \to Y = X, \\
 & \text{identity transformation,} \\
l_A: & \mathbb{R}^{n\times m} \to \mathbb{R}^{p\times m} : X \to Y = AX, \\
 & \text{left multiplication,} \\
r_A: & \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times p} : X \to Y = XA, \\
 & \text{right multiplication,} \\
e_{R,S}: & \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times m} : X \to Y = RXS, \\
 & \text{equivalence transformation,} \\
s_R: & \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n} : X \to Y = R^{-1}XR, \\
 & \text{similarity transformation,} \\
d_A: & \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times m} : X \to Y = A.*X, \\
 & \text{elementwise multiplication (Hadamard product),} \\
 & \text{"diagonal" transformation,} \\
c_A: & \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n} : X \to Y = [X, A] = XA - AX, \\
 & \text{commutator,} \\
\text{trans}: & \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n} : X \to Y = X^T, \\
 & \text{transpose,}
\end{array}
$$

$$
\begin{array}{lll}
\mathsf{upp}: & \mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}: X\to Y, \text{ where} & y_{i,j}=x_{i,j} \quad \text{if } i\leqslant j,\\
 & & y_{i,j}=0 \quad \text{if } i>j,\\
 & \text{upper triangular part,} & \\
 & \text{orthogonal projection on } \mathcal{S}_{upp}(n), & \\
\mathsf{upps}: & \mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}: X\to Y, \text{ where} & y_{i,j}=x_{i,j} \quad \text{if } i<j,\\
 & & y_{i,j}=0 \quad \text{if } i\geqslant j,\\
 & \text{strictly upper triangular part,} & \\
 & \text{orthogonal projection on } \mathcal{S}_{upps}(n), & \\
\mathsf{upph}: & \mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}: X\to Y, \text{ where} & y_{i,j}=x_{i,j} \quad \text{if } i<j,\\
 & & y_{i,j}=\tfrac{1}{2}x_{i,j}, \quad \text{if } i=j,\\
 & & y_{i,j}=0, \quad \text{if } i>j,\\
 & \text{upper triangular half,} & \\
\mathsf{low}: & \mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}: X\to Y, \text{ where} & y_{i,j}=0 \quad \text{if } i<j,\\
 & & y_{i,j}=x_{i,j} \quad \text{if } i\geqslant j,\\
 & \text{lower triangular part,} & \\
 & \text{orthogonal projection on } \mathcal{S}_{low}(n), & \\
\mathsf{lows}: & \mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}: X\to Y, \text{ where} & y_{i,j}=0 \quad \text{if } i\leqslant j,\\
 & & y_{i,j}=x_{i,j} \quad \text{if } i>j,\\
 & \text{strictly lower triangular part,} & \\
 & \text{orthogonal projection on } \mathcal{S}_{lows}(n), & \\
\mathsf{lowh}: & \mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}: X\to Y, \text{ where} & y_{i,j}=0 \quad \text{if } i<j,\\
 & & y_{i,j}=\tfrac{1}{2}x_{i,j} \quad \text{if } i=j,\\
 & & y_{i,j}=x_{i,j} \quad \text{if } i>j,\\
 & \text{lower triangular half,} & \\
\mathsf{diag}: & \mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}: X\to Y, \text{ where} & y_{i,j}=0 \quad \text{if } i\neq j,\\
 & & y_{i,j}=x_{i,j} \quad \text{if } i=j,\\
 & \text{diagonal part,} & \\
\rho_{\mathsf{skew}}: & \mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}: X\to Y=\mathsf{low}(X)-\mathsf{low}(X)^T, & \\
 & \text{projection on } \mathcal{S}_{skew}(n), \text{ parallel to } \mathcal{S}_{upp}(n), & \\
\rho_{\mathsf{upp}}: & \mathbb{R}^{n\times n}\to\mathbb{R}^{n\times n}: X\to Y=\mathsf{upps}(X)+\mathsf{low}(X)^T, & \\
 & \text{projection on } \mathcal{S}_{upp}(n), \text{ parallel to } \mathcal{S}_{skew}(n), &
\end{array}
$$

where A, R and S are matrices of appropriate dimensions, and R and S are nonsingular.

Each of these linear transformations τ can of course be represented by an $nm\times nm$-matrix M_τ, such that $\mathrm{vec}(Y)=M_\tau\,\mathrm{vec}(X)$. The label "diagonal" for d_A refers to the fact that M_{d_A} is diagonal (with respect to the standard basis).

For the sake of clarity, we repeat the definition of the matrix bracket (Lie bracket), included above in the definition of c_A.

Definition 2.10

$$[X,Y]=XY-YX \qquad \text{for } X,Y\in\mathbb{R}^{n\times n}.$$

The following lemma gathers some useful relations between the above definitions. All relations are standard or easy to verify.

Lemma 2.11

(i)
$$\begin{array}{llll} (l_A)^T = l_{(A^T)}, & (l_A)^{-1} = l_{(A^{-1})}, \\ (r_A)^T = r_{(A^T)}, & (r_A)^{-1} = r_{(A^{-1})}, \\ (s_A)^T = s_{(A^T)}, & (s_A)^{-1} = s_{(A^{-1})}, \\ (c_A)^T = c_{(A^T)}, & \\ c_A = r_A - l_A, & \\ s_A = r_A \circ l_{A^{-1}}. & \end{array}$$

(ii) If $\Lambda \in \mathcal{S}_{diag}(n)$

$$\begin{array}{llll} c_\Lambda = d_C, & \text{where} & C_{i,j} = \Lambda_{j,j} - \Lambda_{i,i}, & \\ s_\Lambda = d_S, & & S_{i,j} = \Lambda_{j,j}/\Lambda_{i,i} & (\Lambda_{i,i} \neq 0). \end{array}$$

(iii) $\mathcal{C}_A = \mathsf{null}(c_A)$.

(iv) If $A = P\Lambda P^{-1}$ is an eigenvalue decomposition (EVD) of A, the EVD of s_A and c_A is given by

$$\begin{aligned} s_A &= s_P \circ s_\Lambda \circ s_{(P^{-1})}, \\ c_A &= s_P \circ c_\Lambda \circ s_{(P^{-1})}. \end{aligned}$$

(v) If $Q \in \mathbb{R}^{n\times n}$ is orthogonal l_Q, r_Q, and s_Q are orthogonal.

(vi) If $X \in \mathbb{R}^{n\times n}$ is symmetric

$$\begin{aligned} \rho_{\mathsf{upp}}(X) &= 2\,\mathsf{upph}(X), \\ \rho_{\mathsf{skew}}(X) &= X - 2\,\mathsf{upph}(X) = 2\,\mathsf{lowh}(X) - X. \end{aligned}$$

(vii)
$$\begin{array}{rcccl} \langle AB, C^T\rangle & = & \langle BC, A^T\rangle & = & \langle CA, B^T\rangle, \\ \langle [A,B], C^T\rangle & = & \langle [B,C], A^T\rangle & = & \langle [C,A], B^T\rangle. \end{array}$$

Finally, we recall the well known property that C_A contains all matrices that share (at least a possible choice of) their eigenvectors with A, (and possibly some more that are not diagonalizable). $C_A \cap \mathcal{S}_{symm}(n)$ is equal to the set of all symmetric matrices that share (at least a possible choice of) their eigenvectors with A.

2.3 Differentiable manifolds, Lie groups and homogeneous spaces

In this section we discuss some aspects of differentiable manifolds that will be important in the rest of this work. Most manifolds considered in this work will

be Lie groups or homogeneous spaces. However to understand this work it is not necessary to know the details of the theory of manifolds, nor of the theory of Lie groups and homogeneous spaces.

It will be sufficient to think of a *differentiable manifold* as a subset of a Euclidean space (in our case mostly a matrix space) with at every point a unique tangent space of a fixed dimension, equal to the dimension of the manifold. The *tangent space* $T_X\mathcal{M}$ to a manifold $\mathcal{M}$ at a point $X \in \mathcal{M}$ is the space of all possible velocity vectors at X of differentiable paths through X. The orthogonal complement of the tangent space is referred to as the *normal space*.

The main significance of *Lie groups and homogeneous spaces* for this work is that they are nicely behaved manifolds, with simple expressions for their tangent spaces, and that classical matrix decompositions decompose a matrix into a product of matrices belonging to some classical Lie groups. We first discuss three examples of such manifolds and then briefly situate them in a more general framework. The first example is the orthogonal group $\mathsf{O}(n)$, that is the Lie group of orthogonal $n \times n$ matrices. The second example is the isospectral manifold of symmetric matrices with fixed eigenvalues. This manifold is a homogeneous space. The third example is the Stiefel Manifold $\mathrm{St}(m, n)$, a homogeneous space consisting of the rectangular matrices with orthogonal columns. Because of the central position of these manifolds in this work, we also try to get some intuitive ideas about their geometry.

2.3.1 The orthogonal group $\mathsf{O}(n)$, as a regular manifold

In this section we consider the orthogonal group $\mathsf{O}(n)$, containing all orthogonal $n \times n$-matrices (definition 2.24, below).

The group $\mathsf{O}(n)$ is not a connected manifold, but consists of two connected submanifolds. One is the special orthogonal group $\mathsf{SO}(n)$ of all orthogonal matrices with determinant 1, consisting of rotation matrices. The other part is the set of orthogonal matrices with determinant -1, which can be written as the product of a rotation matrix and a reflection matrix. To aid visualization, figure 2.2 gives a schematic picture. Note however that $\mathsf{SO}(n)$ is not a torus. But it is a highly symmetric compact manifold.

Proposition 2.12
(i) The tangent space $T_Q\mathsf{O}(n)$ to $\mathsf{O}(n)$ at a point Q is given by

$$T_Q\mathsf{O}(n) = \{Q\Omega | \Omega \in \mathcal{S}_{skew}(n)\} = \{\tilde{\Omega}Q | \tilde{\Omega} \in \mathcal{S}_{skew}(n)\}. \tag{2.4}$$

(ii) The normal space at Q is given by

$$(T_Q\mathsf{O}(n))^{\perp} = \{QS | S \in \mathcal{S}_{symm}(n)\} = \{\tilde{S}Q | \tilde{S} \in \mathcal{S}_{symm}(n)\}.$$

(iii) $\dim \mathsf{O}(n) = \frac{1}{2}n(n-1)$.

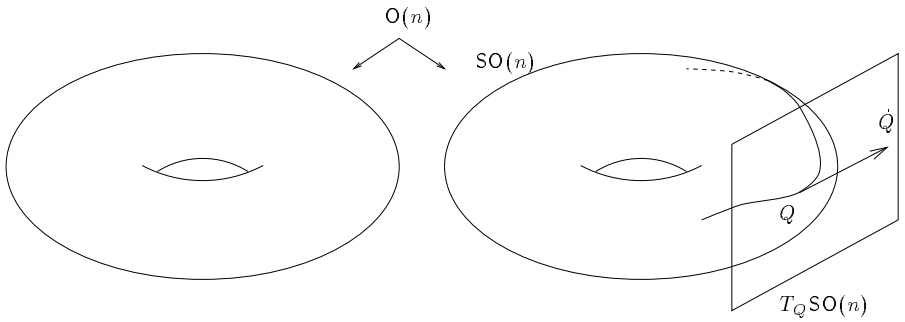


Figure 2.2: Schematic picture of $\mathsf{O}(n)$ and its tangent space $T_Q\mathsf{O}(n)$ at a point Q

Proof:
(i) A tangent vector T at Q is the velocity of a differentiable path $Q(t) \in \mathsf{O}(n)$ through Q. Say $Q(0) = Q$, $T = \dot{Q} = \dot{Q}(0)$. For a path $Q(t) \in \mathsf{O}(n)$ we have $Q^T(t)Q(t) = I$, and by differentiating at $t = 0$,

$$\frac{d}{dt}(Q^TQ)|_{t=0} = \dot{Q}^TQ + Q^T\dot{Q} = (Q^T\dot{Q})^T + Q^T\dot{Q} = 0.$$

That is, $Q^T\dot{Q}$ must be skew symmetric, or equivalently, $\dot{Q}$ must be of the form $Q\Omega$ where $\Omega \in \mathcal{S}_{skew}(n)$. The second identity is obtained in a similar way from $Q(t)Q(t)^T = I$.

Conversely every vector of the form $Q\Omega$ or $\tilde{\Omega}Q$ is a velocity vector of a path. Take for instance $Qe^{\Omega t}$ and $e^{\tilde{\Omega}t}Q$.

One can also see the one to one correspondence between vectors of the form $Q\Omega$ and vectors of the form $\tilde{\Omega}Q$ through $\tilde{\Omega} = Q\Omega Q^T$.

(ii) By lemma 2.11(vii), $\langle X, Q\Omega\rangle = 0$ for all $\Omega \in \mathcal{S}_{skew}(n)$ if and only if $\langle Q^TX, \Omega\rangle = 0$ for all $\Omega \in \mathcal{S}_{skew}(n)$. This holds if and only if $Q^TX \in \mathcal{S}_{symm}(n)$. This proves the first identity, the second identity is proved in a similar way.

(iii) $\dim\mathsf{O}(n) = \dim\mathcal{S}_{skew}(n) = \frac{1}{2}n(n-1)$.
□

Remark 2.13
A more intuitive explanation of the dimension $\frac{1}{2}n(n-1)$ of $\mathsf{O}(n)$ lies in the fact that any $Q \in \mathsf{O}(n)$ can be described uniquely by $\frac{1}{2}n(n-1)$ angles.

One can think of $Q\Omega$ (or $\tilde{\Omega}Q$) as an 'instantaneous rotation'. The entries $\omega_{i,j}$ are then angular velocities of 'planar components' as described by the following interpretation. The fact that $\Omega = Q^T\dot{Q}$ must be skew symmetric can be inter-

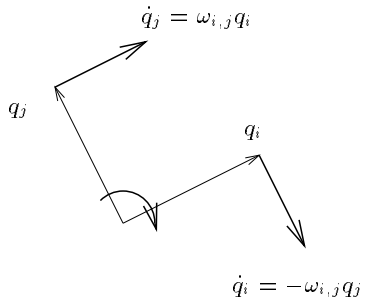


Figure 2.3: Rotation due to $\omega_{i,j}$ of columns q_i and q_j if $\dot{Q} = Q\Omega$

preted geometrically by considering the evolution of columns of Q for such a planar component. (Figure 2.3).

Interpretation 2.14
(i) If Q has a velocity $\dot{Q} = Q\Omega$, the component $\omega_{i,j}(E^{(i,j)} - E^{(j,i)})$ of Ω corresponds to an instantaneous planar rotation of two columns of Q.
(ii) If $\dot{q}_i$ has a component along q_j, $\dot{q}_j$ must have a corresponding component along $-q_i$ to preserve orthogonality. This explains the skew symmetry of Ω.

For $\dot{Q} = Q\omega_{i,j}(E^{(i,j)} - E^{(j,i)})$ implies

$$\begin{array}{rcl} \dot{q}_i & = & -\omega_{i,j} q_j, \\ \dot{q}_j & = & \omega_{i,j} q_i, \\ \dot{q}_k & = & 0, \text{ if } k \neq i \text{ and } k \neq j. \end{array}$$

That is, q_i and q_j rotate instantaneously as in figure 2.3.

$\mathsf{O}(n)$ is a highly symmetrical manifold. One can get an idea of this symmetry from the following observations. Every pair of orthogonal matrices $U, V \in \mathsf{O}(n)$ induce an orthogonal transformation of the matrix space $e_{U,V} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n} : X \to Y = UXV$, mapping $\mathsf{O}(n)$ onto itself.

A more concrete picture of the regularity of $\mathsf{O}(n)$ follows from the following lemma.

Lemma 2.15
Let $\Omega \in \mathcal{S}_{skew}(n)$ be any skew symmetric matrix of rank 2. Then
(i) Ω can be written as $pq^T - qp^T$, $p, q \in \mathbb{R}^n$ $(p, q \neq 0)$ and has eigenvectors

$$\begin{array}{ll} x + iy & \text{with eigenvalue } i\omega, \qquad (i^2 = -1), \\ x - iy & \text{with eigenvalue } -i\omega, \end{array}$$

where $\omega = \|p\|\|q\| \sin(\angle(p, q))$ and $\{x, y\}$ is an orthonormal basis of $\text{span}\{p, q\}$.
The null space of Ω is the orthogonal complement $\{x, y\}^\perp$.

(ii) $\Omega = pq^T - qp^T$ generates a circle through I in $\mathsf{O}(n)$, through the linear system

$$\dot{Q} = Q\Omega, \qquad Q(0) = I, \tag{2.5}$$

yielding

$$Q(t) = e^{\Omega t} = (I-(xx^T+yy^T))+\cos(\omega t)(xx^T+yy^T)+\sin(\omega t)(xy^T-yx^T). \tag{2.6}$$

The points of this circle are planar rotation matrices.

Proof:
(i) is easy to verify, taking $x = p/\|p\|$ and $y = (I-xx^T)q/\|(I-xx^T)q\|$, yielding $\Omega = \|p\|\|(I-xx^T)q\|(xy^T - yx^T)$.
(ii) $Q = e^{\Omega t}$ from linear systems theory. The exponentiation is done using the EVD of Ω given in (i). The result (2.6) is a circle because $\langle xx^T + yy^T, xy^T - yx^T\rangle = 0$. The obtained matrices are planar rotation matrices, as follows from their eigenvalues $e^{i\omega t}$, $e^{-i\omega t}$ and $n-2$ times 1.
□

For $\Omega = \omega \cdot (E^{(i,j)} - E^{(j,i)})$ one obtains so called Givens rotations $e^{\Omega t} = G^{i|j}(\omega t)$.

Definition 2.16 (Givens rotation)
The Givens rotation $G^{i|j}(\phi) \in \mathbb{R}^{n\times n}$ is the planar rotation with angle ϕ affecting only components i and j of a vector it is operating on. It is given by

$$\begin{aligned} G^{i|j}(\phi) &= (I - (E^{(i,i)} + E^{(j,j)})) + \cos(\phi)(E^{(i,i)} + E^{(j,j)}) \\ &\quad + \sin(\phi)(E^{(i,j)} - E^{(j,i)}) \\ &= \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix}. \end{aligned}$$

2.3.2 Isospectral manifolds

As a second example we consider the *isospectral manifold* $\mathcal{I}_A$ of symmetric matrices with the same spectrum (the same eigenvalues with the same multiplicity) as $A \in S_{symm}$.

Definition 2.17 (Isospectral manifold)

$$\mathcal{I}_A = \{XAX^T | X \in \mathsf{O}(n)\}.$$

If the EVD of A is given by $A = Q\Lambda Q^T$, we have

$$\mathcal{I}_A = \mathcal{I}_\Lambda.$$

This manifold is the image of $\mathsf{O}(n)$ under the (nonlinear) smooth map ϕ_A (or ϕ_Λ), defined as follows.

Definition 2.18

$$\phi_A : \mathsf{O}(n) \to \mathbb{R}^{n\times n} : X \to Y = XAX^T.$$

Note that the mapping ϕ_A is not one to one. The inverse image $X = \phi_A^{-1}(Y)$ is the set of possible orthogonal eigenvector matrices in an EVD of $Y \in \mathcal{I}_\Lambda$ (definition 3.6). These matrices are determined up to orthogonal transformations of the eigenvectors in their eigenspaces. Even if A has distinct eigenvalues, one can still choose the sign of each eigenvector. It also follows that $\mathcal{I}_\Lambda$ unlike $\mathsf{O}(n)$ is a connected manifold, as it can also be seen as the image of the connected manifold $\mathsf{SO}(n)$ under the continuous map ϕ_A.

Proposition 2.19
Let $A = Q\Lambda Q^T \in \mathcal{S}_{symm}(n)$, $Q \in \mathsf{O}(n)$, $\Lambda \in \mathcal{S}_{diag}(n)$, then
(i) The tangent space $T_A\,\mathcal{I}_\Lambda$ to the isospectral manifold $\mathcal{I}_\Lambda$ at $A \in \mathcal{I}_\Lambda$ is given by

$$T_A\,\mathcal{I}_\Lambda = \{[\tilde{\Omega}, A] | \tilde{\Omega} \in \mathcal{S}_{skew}(n)\}.$$

(ii) The normal space to $\mathcal{I}_\Lambda$ in $\mathcal{S}_{symm}(n)$, is given by

$$(T_A\,\mathcal{I}_\Lambda)^\perp \cap \mathcal{S}_{symm}(n) = \mathcal{C}_A \cap \mathcal{S}_{symm}(n).$$

(iii) If A has r distinct eigenvalues with multiplicities $m_1, \ldots, m_r$,

$$\dim \mathcal{I}_\Lambda = \dim T_A\,\mathcal{I}_\Lambda = \frac{1}{2}(n^2 - \sum_{i=1}^{r} m_i^2).$$

Proof:
(i) The path $A(t)$ through $A = A(0) = Q\Lambda Q^T$ stays on $\mathcal{I}_\Lambda$ if there exists some path $Q(t)$ on $\mathsf{O}(n)$ such that $A(t) = \phi_\Lambda(Q(t)) = Q(t)\Lambda Q(t)^T$. For $\dot{A} = \dot{A}(0)$, this means that

$$\dot{A} = \dot{Q}\Lambda Q^T + Q\Lambda \dot{Q}^T.$$

With $\dot{Q} = \tilde{\Omega}Q$, we obtain

$$\dot{A} = \tilde{\Omega}A - A\tilde{\Omega} = [\tilde{\Omega}, A].$$

(ii) Let $X \in \mathcal{S}_{symm}(n)$ then $\langle X, [\tilde{\Omega}, A]\rangle = 0$ for all $\tilde{\Omega} \in \mathcal{S}_{skew}(n)$ if and only if $\langle \tilde{\Omega}, [A, X]\rangle = 0$ for all $\tilde{\Omega} \in \mathcal{S}_{skew}(n)$ (lemma 2.11(vii)). Since $[A, X] \in \mathcal{S}_{skew}(n)$, $[A, X]$ must be 0, which proves (ii). Conversely every vector of the given form is the velocity vector of a path. Take for instance $e^{\tilde{\Omega}t} A e^{-\tilde{\Omega}t}$.
(iii) The tangent space $T_Q\mathsf{O}(n)$ is taken to the tangent space $T_\Lambda \mathcal{I}_\Lambda$ by the linear transformation $c_\Lambda = [., \Lambda] = d_C$, with $C_{i,j} = \lambda_{i,i} - \lambda_{j,j}$ as in lemma 2.11(ii). The rank of d_C, restricted to $\mathcal{S}_{skew}(n)$, is equal to the number of index pairs $\{i, j\}$ for which $\lambda_{i,i} \neq \lambda_{j,j}$. This yields (iii).
□

Remark 2.20
As for lemma 2.12(iii) a more intuitive proof consists in counting the number of angles needed to describe the eigenspaces.

Every similarity transformation s_P with $P \in \mathsf{O}(n)$ is an orthogonal transformation, mapping $\mathcal{I}_\Lambda$ onto itself.

2.3.3 The Stiefel manifold of rectangular orthogonal matrices

Definition 2.21 (Stiefel manifold)
The Stiefel manifold $\mathsf{St}(m, n)$ is defined by $\mathsf{St}(m, n) = \{A \in \mathbb{R}^{n\times m} | A^T A = I\}$.

Proposition 2.22
(i) The tangent space $T_A\,\mathsf{St}(m, n)$ tangent to $\mathsf{St}(m, n)$ at a point A is given by

$$\begin{aligned} T_A\,\mathsf{St}(m,n) &= \{T \in \mathbb{R}^{n\times m} | A^T T \in \mathcal{S}_{skew}(m)\} \\ &= \{\tilde{\Omega} A | \tilde{\Omega} \in \mathcal{S}_{skew}(n)\} \\ &= \{A\Omega + B\Psi | \Omega \in \mathcal{S}_{skew}(m),\ \Psi \in \mathbb{R}^{(n-m)\times m}\}. \end{aligned}$$

where $B \in \mathsf{St}(n - m, n)$ satisfies $B^T A = 0$.

(ii) $\dim \mathsf{St}(m, n) = m(n - m) + \frac{1}{2}m(m - 1)$.

Proof:
(i) The first description follows by proceeding as in the proof of proposition 2.12(i). The second and third description follow by regarding $\mathrm{St}(m, n)$ as the image of $\mathsf{O}(n)$ under the linear transformation r_{E_m} (definition 2.9) where $E_m = I_{.,\leqslant m}$, selecting the first m columns of a matrix. A is the image of $Q = [A|B]$ where B spans the orthogonal complement of $\mathrm{col}(A)$. If $\frac{d}{dt}[A|B] = \tilde{\Omega}[A|B] = [A|B]\bar{\Omega}$ we obtain $\dot{A} = \tilde{\Omega}[A|B]E_m = \tilde{\Omega}A$, yielding the second description. Also, $\dot{A} = [A|B]\bar{\Omega}E_m = A\bar{\Omega}_{\leqslant m,\leqslant m} + B\bar{\Omega}_{>m,\leqslant m}$. Setting $\Omega = \bar{\Omega}_{\leqslant m,\leqslant m}$ and $\Psi = \bar{\Omega}_{>m,\leqslant m}$, we obtain the third description.

(ii) One can easily verify that the third description in (i) gives a unique characterization of tangent vectors with $m(n-m) + \frac{1}{2}m(m-1)$ parameters.
□

In the interpretation as instantaneous rotations as in interpretation 2.14, the entries of Ω correspond to rotations of the columns of A inside the column space $\mathrm{col}(A)$, while the entries Ψ represent rotations of $\mathrm{col}(A)$ "toward its orthogonal complement".

2.3.4 Lie groups and homogeneous spaces

In this section we introduce other Lie groups that will play a role in this work. To establish a link with the literature on related topics, we also briefly situate the above examples in a more theoretic framework. For a more extensive account we refer to [55, 94, 12].

Definition 2.23 (Lie group)
A *Lie group* $\mathcal{G}$ is an analytic manifold that is also an algebraic group for which the group operation and the inversion are analytic functions.

In this text, we will only consider matrix groups with the matrix multiplication as group operation.

To every Lie group is also associated a *Lie algebra* $\mathfrak{g}$, which can be identified with the tangent space $T_I\mathcal{G}$ to the group $\mathcal{G}$ at the identity element. On the Lie algebra a bracket operation is defined, called the *Lie bracket*. For matrix Lie groups the Lie bracket can be defined as $[A, B] = AB - BA$ (as in definition 2.10). A Lie algebra is an algebraic structure that can be defined without referring to groups, but we will not exploit many properties of Lie algebras in this work, except for their *relation with the tangent spaces to a Lie group* (lemma 2.27).

Lie groups in matrix space form highly regular manifolds, with simple descriptions of their tangent spaces. Some examples that play a role in this work are

Definition 2.24

$$
\begin{aligned}
\mathsf{GL}(n) &= \{X \in \mathbb{R}^{n\times n} | \det X \neq 0\},\\
&\quad \text{general linear group,}\\
\mathsf{SL}(n) &= \{X \in \mathbb{R}^{n\times n} | \det X = 1\},\\
&\quad \text{special linear group,}\\
\mathsf{O}(n) &= \{X \in \mathbb{R}^{n\times n} | X^T X = I\},\\
&\quad \text{orthogonal group,}\\
\mathsf{SO}(n) &= \{X \in \mathbb{R}^{n\times n} | X^T X = I, \det X = 1\},\\
&\quad \text{special orthogonal group,}\\
\mathcal{G}_{upp}(n) &= \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i > j, x_{i,i} \neq 0, i, j = 1, \ldots, n\},\\
&\quad \text{group of nonsingular upper triangular matrices,}\\
\mathcal{G}_{upp}(n)^+ &= \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i > j, x_{i,i} > 0, i, j = 1, \ldots, n\},\\
&\quad \text{group of upper triangular matrices with positive diagonal,}\\
\mathcal{G}_{upps}(n) &= \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i > j, x_{i,i} = 1, i, j = 1, \ldots, n\},\\
&\quad \text{group of upper triangular matrices with unit diagonal,}\\
\mathcal{G}_{low}(n) &= \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i < j, x_{i,i} \neq 0, i, j = 1, \ldots, n\},\\
&\quad \text{group of nonsingular lower triangular matrices,}\\
\mathcal{G}_{lows}(n) &= \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i < j, x_{i,i} = 1, i, j = 1, \ldots, n\},\\
&\quad \text{group of lower triangular matrices with unit diagonal,}\\
\mathcal{G}_{diag}(n) &= \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i \neq j, x_{i,i} \neq 0, i, j = 1, \ldots, n\},\\
&\quad \text{group of nonsingular diagonal matrices,}\\
\mathcal{G}_A &= \{\exp(At) | t \in \mathbb{R}\}\\
&\quad \text{one-parameter group generated by } A \in \mathbb{R}^{n\times n}.
\end{aligned}
$$

In all cases, the group operation is matrix multiplication. (And as a consequence, the inversion operation is the matrix inversion).

The corresponding Lie algebras are

Definition 2.25

$$
\begin{aligned}
\mathfrak{gl}(\mathsf{n}) &= \mathbb{R}^{n\times n},\\
\mathfrak{sl}(\mathsf{n}) &= \{X \in \mathbb{R}^{n\times n} | \,\mathsf{Tr}X = 0\},\\
\mathfrak{o}(\mathsf{n}) &= \mathcal{S}_{skew}(n) = \{X \in \mathbb{R}^{n\times n} | X^T = -X\},\\
\mathfrak{so}(\mathsf{n}) &= \mathcal{S}_{skew}(n) = \{X \in \mathbb{R}^{n\times n} | X^T = -X\},\\
\mathfrak{g}_{upp}(n) &= \mathcal{S}_{upp}(n) = \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i > j, i, j = 1, \ldots, n\},\\
\mathfrak{g}_{upp}(n)^+ &= \mathfrak{g}_{upp}(n) = \mathcal{S}_{upp}(n),\\
\mathfrak{g}_{upps}(n) &= \mathcal{S}_{upps}(n) = \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i \geqslant j, i, j = 1, \ldots, n\},\\
\mathfrak{g}_{low}(n) &= \mathcal{S}_{low}(n) = \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i < j, i, j = 1, \ldots, n\},\\
\mathfrak{g}_{lows}(n) &= \mathcal{S}_{lows}(n) = \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i \leqslant j, i, j = 1, \ldots, n\},\\
\mathfrak{g}_{diag}(n) &= \mathcal{S}_{diag}(n) = \{X \in \mathbb{R}^{n\times n} | x_{i,j} = 0 \text{ if } i \neq j, i, j = 1, \ldots, n\},\\
\mathfrak{g}_A &= \{rA | r \in \mathbb{R}\}.
\end{aligned}
$$

Remark 2.26
The Lie algebra of $\mathsf{O}(n)$ is mostly referred to as $\mathfrak{so}(n)$ (Note that $\mathfrak{o}(n) = \mathfrak{so}(n)$. $\mathsf{SO}(n)$ is the connected component of $\mathsf{O}(n)$ containing the identity).

Lemma 2.27
The tangent space to a Lie group $\mathcal{G}$ at an element $G \in \mathcal{G}$, with Lie algebra $\mathfrak{g}$ (identified with the tangent space at the identity element) is given by

$$T_G\mathcal{G} = \{GS | S \in \mathfrak{g}\} = \{\tilde{S}G | \tilde{S} \in \mathfrak{g}\}.$$

The highly regular structure of a Lie group $\mathcal{G}$ considered as a manifold is due to the group structure. Linear operations $e_{U,V}$ with $U, V \in \mathcal{G}$ always map the manifold on itself, although in general they are not orthogonal transformations as in the above example of $\mathsf{O}(n)$. The circles in $\mathsf{O}(n)$ correspond to the subgroup generated by $(E^{(1,2)} - E^{(2,1)})$ (isomorphic to $\mathsf{SO}(2)$), its images under the automorphisms $s_P = e_{P^T,P}$ and their cosets (found by left or right multiplication). More generally submanifolds, diffeomorphic to any subgroup can be found at any point.

To establish a link with literature on related topics, we situate the above examples of homogeneous spaces in a more theoretic framework in the following. Knowledge of this theory is not a prerequisite to read the remainder of this work.

Definition 2.28 (Homogeneous space, group action, orbit)
Homogeneous spaces are the orbits of Lie group actions. A *Lie group action* of a group $\mathcal{G}$ on a smooth manifold M, is a smooth map $\psi : \mathcal{G} \times \mathcal{M} \rightarrow \mathcal{M}$ satisfying

$$\begin{aligned} \psi(g, \psi(h, x)) &= \psi(gh, x), \\ \psi(e, x) &= x, \end{aligned}$$

where $g, h \in \mathcal{G}$, $x \in \mathcal{M}$ and $e \in \mathcal{G}$ is the identity element of $\mathcal{G}$.

The *orbit* $\mathcal{O}(a)$ of an element $a \in \mathcal{M}$ is the image under $\psi_a = \psi(., a)$ of $\mathcal{G}$,

$$\mathcal{O}(a) = \psi_a(\mathcal{G}) = \{\psi(g, a) \in \mathcal{M} | g \in \mathcal{G}\}.$$

Note that, if $b \in \mathcal{O}(a)$, $\mathcal{O}(a) = \mathcal{O}(b)$.

In the example of the isospectral manifold $\mathcal{I}_A$, discussed above, we had $\mathcal{G} = \mathsf{O}(n)$ and $\mathcal{M} = \mathcal{S}_{symm}(n)$, and $\psi_A = \phi_A(P) = PAP^T$ for $A \in \mathcal{G}$. Then, $\mathcal{O}(A) = \phi_A(\mathsf{O}(n)) = \mathcal{I}_A$. If $B \in \mathcal{I}_A$, $\mathcal{I}_A = \mathcal{I}_B$. In particular we had $\mathcal{I}_A = \mathcal{I}_\Lambda$.

Every $g \in \mathcal{G}$ induces a transformation $\psi(g, .)$ of the orbit $\mathcal{O}(a)$. For $\mathcal{I}_\Lambda$, we have $\psi(P, .) = s_P$.

The map ψ_a may not be 1 to 1. The set $\psi_a^{-1}(a)$ of elements $g \in \mathcal{G}$ that (like the identity element $e \in \mathcal{G}$) are mapped to a is a subgroup of $\mathcal{G}$, denoted $\mathrm{Stab}(a)$

and called the stabilizer of a. Sets of group elements that (like some element $g \in \mathcal{G}$) are mapped to the same point $b = \psi_a(g)$ of $\mathcal{M}$ are left cosets $g\,\mathrm{Stab}(a)$ of the stabilizer. That is, $\mathcal{M}$ can be identified with $\mathcal{G}/Stab(a)$, denoting the set of left cosets of $\mathrm{Stab}(a)$. In the example of $\mathcal{I}_\Lambda$, $\mathrm{Stab}(A)$ is the group of orthogonal transformations in the eigenspaces of A. $\mathrm{Stab}(A)$ can be identified with $\mathsf{O}(m_1) \times \ldots \times \mathsf{O}(m_r)$, ($m_1, \ldots, m_r$ as in lemma 2.19(iii)). (This observation also implies an alternative proof for lemma 2.19).

The derivative $\mathrm{D}\psi_a|_e$ of ψ_a at the identity $e \in \mathcal{G}$ maps tangent vectors $t \in \mathfrak{g}$, tangent to $\mathcal{G}$ at e to tangent vectors to $\mathcal{O}(a)$ at a. The kernel of $\mathrm{D}\psi_a|_e$ is the Lie algebra of $\mathrm{Stab}(A)$, a subalgebra of $\mathfrak{g}$. For $\mathcal{I}_\Lambda$, we have $\mathrm{D}\phi_A(I) = c_A$. (The kernel of $D\phi_\Lambda$ played a role in the proof of lemma 2.19). The tangent space $T_a\mathcal{O}(a)$ can be identified with the orthogonal complement of the kernel of $\mathrm{D}\psi_a|_e$. This identification generalizes the identification between tangent vectors to Lie groups and vectors of the Lie algebra.

Also the Stiefel manifold $\mathrm{St}(m,n)$ is a homogeneous space. We have $\psi(Q,A) = QA$, with $Q \in \mathsf{O}(n)$, $A \in \mathbb{R}^{n\times m}$ and $\mathrm{St}(m,n) = \mathcal{O}(E_m)$, where $E_m = I_{:,\leqslant m}$. Also $\psi_A = r_A$ (right multiplication). The derivative $\mathrm{D}\psi_A|_I = r_A$, maps a tangent vector $\tilde{\Omega}$ at I onto $\tilde{\Omega}A$. Hence the second description of the tangent space in proposition 2.22(i). The kernel of $\mathrm{D}\psi_A|_I$ is given by the matrices for which $\Omega = Q^T\tilde{\Omega}Q$ has only nonzero entries in the lower right block $\Omega_{>m,>m}$. This way, the third description in the same proposition identifies the tangent space to $\mathrm{St}(m,n)$ with the orthogonal complement of the kernel of $\mathrm{D}\psi_A$.

Other examples of homogeneous spaces are the left (or right) cosets of a subgroup $\mathcal{G}$ of $\mathsf{GL}(n)$. In that case $\psi(G,A) = AG$ (or $\psi(G,A) = GA$) giving $\psi_A = l_A$ (or $\psi_A = r_A$). The tangent vectors to $\mathcal{O}(A)$ at A have the form AL (or LA) where $L \in \mathfrak{g}$.

2.4 Continuous-time algorithms on manifolds

In this section, we extend the picture of continuous-time matrix algorithms, sketched in section 2.1 to algorithms on manifolds.

A continuous-time algorithm on a differentiable manifold $\mathcal{M}$ takes the form

$$\dot{X} = F(X,t), \tag{2.7}$$

where $F(X,t) \in T_X\mathcal{M}$ for every $X \in \mathcal{M}$ and for all time instants t.

If F is continuously differentiable and $\mathcal{M}$ is compact, solutions $X(t) \in \mathcal{M}$ exist for all t [14].

A continuous-time algorithm on the Lie group $\mathsf{O}(n)$, takes the general form

$$\dot{Q} = Q\Omega(Q,t),$$

where $\Omega(Q,t) \in \mathfrak{so}(n)$, or equivalently

$$\dot{Q} = \tilde{\Omega}(Q,t)Q.$$

A continuous-time algorithm on $\mathcal{I}_\Lambda$ takes the general form

$$\dot{A} = [\tilde{\Omega}(A,t), A]. \tag{2.8}$$

This is the image under ϕ_Λ of an algorithm

$$\dot{Q} = \tilde{\Omega}(A,t)Q. \tag{2.9}$$

The examples of the QR flow and the double bracket flow, discussed in chapter 4 will take this form.

Autonomous versions of (2.8) and (2.9) are the continuous-time counterpart of discrete-time algorithms of the form

$$\begin{array}{rcl} Q' & = & Z(A)Q, \\ A' & = & Z(A)AZ(A)^T. \end{array} \tag{2.10}$$

where $Z(A) \in \mathsf{O}(n)$. This is the form of many algorithms for computing the EVD of $A^{[0]}$.

2.5 Conclusion

We have defined continuous-time matrix algorithms, introduced some elementary concepts and recalled properties of $\mathbb{R}^{n\times m}$ as a Euclidean vector space, and we have discussed some important manifolds in matrix space. The most important results are the formulas for the tangent spaces of these manifolds, propositions 2.12, 2.19 and 2.22.

Chapter 3

Tools for the design and analysis of continuous-time matrix algorithms

A major tool for the design of continuous-time algorithms is the use of gradients. Almost every publication in this field makes use of gradients in some way. Gradients and gradient flows will be treated in section 3.3 of this chapter. However first, we want to put special emphasis on formulas for tracking decompositions of a time-varying matrix, assembled in section 3.1. These formulas play a central role in this work, and we like to present them as a toolbox for the design and analysis of algorithms. Section 3.2 places them in a geometrical framework, by interpreting some matrix decompositions as nonlinear coordinate transformations of matrix space. In section 3.4 we address generalizations for complex matrices.

3.1 Tracking matrix decompositions

We consider common nonlinear functions of matrices, like the inverse of a matrix and the factors of some classical matrix decompositions such as QR decomposition, LU decomposition, eigenvalue decomposition (EVD), singular value decomposition (SVD) or Cholesky decomposition. The derivatives of these functions link the evolution of the inverse of a matrix $X(t)$ or of a factor in some matrix decomposition, to the evolution of the matrix X itself. All derivatives are compositions of simple linear operations as the ones defined in definition 2.9, and depend on X in a simple way.

Related formulas can be found at different places throughout literature, sometimes in the context of continuous-time algorithms and often in a context of perturbation theory. Formulas, similar to lemma 3.4 are used in the derivation of continuous-time QR-type algorithms in [96]. Tracking matrix factors along a path, is also done in [13, 69]. A recent publication, with a perturbation analysis of QR-like factorizations, with derivations in a differential geometric context is [6]. A perturbation analysis of the eigenvalue decomposition, yielding formulas similar to lemma 3.9 for the case of distinct eigenvalues, can be found in [48, p.346]. However, these formulas are rarely put together as below and used for the design and analysis of algorithms as is done further in this work.

3.1.1 Tracking the inverse and decompositions of the form $X = YZ$

The following definition introduces notations and recalls existence conditions for the inverse and three matrix decompositions of the form $X = YZ$.

Definition 3.1
If $X \in \mathbb{R}^{n\times n}$ is nonsingular, we denote

$$\mathsf{inv}(X) \quad = \quad X^{-1} \quad \text{(inverse)}.$$

If $X \in \mathcal{S}_{symm}(n)$ is positive definite and symmetric, there exists a unique upper triangular matrix $C \in \mathcal{G}_{upp}(n)^+$ such that $X = C^T C$. We denote

$$\begin{array}{rcll} \mathsf{cf}(X) & = & C & \text{(Cholesky factor)}, \\ \mathsf{icf}(X) & = & C^{-1} & \text{(inverse Cholesky factor)}. \end{array}$$

If every principal submatrix $X_{\leqslant i,\leqslant i}$, $i = 1, \ldots, n$ of $X \in \mathbb{R}^{n\times n}$ is nonsingular, there exist a unique $L \in \mathcal{G}_{lows}(n)$ and $U \in \mathcal{G}_{upp}(n)$, such that $X = LU$. We denote

$$\begin{array}{rcll} \mathsf{lf} & = & L & \text{(L-factor)}, \\ \mathsf{uf} & = & U & \text{(U-factor)}, \\ \mathsf{ilf} & = & L^{-1} & \text{(inverse L-factor)}, \\ \mathsf{iuf} & = & U^{-1} & \text{(inverse U-factor)}. \end{array}$$

If $X \in \mathbb{R}^{n\times n}$ is nonsingular, there exist a unique $Q \in \mathsf{O}(n)$ and $R \in \mathcal{G}_{upp}(n)^+$ such that $X = QR$. Similarly, if $X \in \mathbb{R}^{n\times m}$ (rectangular) has full rank there exist a unique $Q \in \mathsf{St}(m,n)$ and $R \in \mathcal{G}_{upp}(m)^+$ such that $X = QR$. We denote

$$\begin{array}{rcll} \mathsf{qf} & = & Q & \text{(Q-factor)}, \\ \mathsf{rf} & = & R & \text{(R-factor)}, \\ \mathsf{irf} & = & R^{-1} & \text{(inverse R-factor)}. \end{array}$$

Lemma 3.2
All functions defined in definition 3.1 are differentiable over their domain of definition.

Proof:
Standard algorithms for calculating these functions are compositions of multiplications, additions, division by nonzero numbers, and square roots of nonzero numbers.
□

Below, we give the derivatives of the functions defined above. The results are given in terms of velocities of matrices and matrix factors. If $\mathrm{D}\Phi|_X$ denotes the derivative of the map Φ at X, and if $Y(t) = \Phi(X(t))$, then $\dot{Y} = \mathrm{D}\Phi|_X(\dot{X})$. We will mainly use these formulas for the purpose of *tracking* matrix factors of a path $X(t)$ described by a differential equation $\dot{X} = F(X)$.

Lemma 3.3
Let $X(t) \in \mathbb{R}^{n\times n}$ describe a continuously differentiable path.
If $Y = Y(t) = \mathsf{inv}(X(t))$,

$$\dot{Y} = \mathrm{D}\,\mathsf{inv}|_X(\dot{X}) = -Y\dot{X}Y. \tag{3.1}$$

Proof:
$\dot{Y}$ exists from lemma 3.2. Since $X(t)Y(t) = I$ for all t, we have $\dot{X}Y + X\dot{Y} = 0$. Solving for $\dot{Y}$ yields (3.1).
□

Note that this lemma is the matrix counterpart of the well known scalar formula

$$\frac{d}{dt}\frac{1}{x} = -\frac{1}{x^2}\dot{x}.$$

Lemma 3.4
Let $X(t) \in \mathbb{R}^{n\times n}$ describe a continuously differentiable path,
(i) If $C = C(t) = \mathsf{cf}(X(t))$ and $D = D(t) = \mathsf{icf}(X(t))$

$$\begin{array}{rclcl} \dot{C} & = & \mathrm{D}\,\mathsf{cf}|_X(\dot{X}) & = & \mathsf{upph}(C^{-T}\dot{X}C^{-1})C, \\ \dot{D} & = & \mathrm{D}\,\mathsf{icf}|_X(\dot{X}) & = & -D\,\mathsf{upph}(D^T\dot{X}D). \end{array}$$

(ii) If $L = L(t) = \mathsf{lf}(X(t))$, $U = U(t) = \mathsf{uf}(X(t))$, $M = M(t) = \mathsf{ilf}(X(t))$ and $V = V(t) = \mathsf{iuf}(X(t))$,

$$\begin{array}{rclcl} \dot{L} & = & \mathrm{D}\,\mathsf{lf}|_X(\dot{X}) & = & L\,\mathsf{lows}(L^{-1}\dot{X}U^{-1}), \\ \dot{U} & = & \mathrm{D}\,\mathsf{uf}|_X(\dot{X}) & = & \mathsf{upp}(L^{-1}\dot{X}U^{-1})U, \\ \dot{M} & = & \mathrm{D}\,\mathsf{ilf}|_X(\dot{X}) & = & -\mathsf{lows}(M\dot{X}V)M, \\ \dot{V} & = & \mathrm{D}\,\mathsf{iuf}|_X(\dot{X}) & = & -V\,\mathsf{upp}(M\dot{X}V). \end{array}$$

(iii) If $Q = Q(t) = \mathsf{qf}(X(t))$, $R = R(t) = \mathsf{rf}(X(t))$, and $S = S(t) = \mathsf{irf}(X(t))$,

$$\begin{array}{rcccl} \dot{Q} & = & \mathsf{D}\,\mathsf{qf}|_X(\dot{X}) & = & Q\rho_{\mathsf{skew}}(Q^T\dot{X}R^{-1}), \\ \dot{R} & = & \mathsf{D}\,\mathsf{rf}|_X(\dot{X}) & = & \rho_{\mathsf{upp}}(Q^T\dot{X}R^{-1})R, \\ \dot{S} & = & \mathsf{D}\,\mathsf{irf}|_X(\dot{X}) & = & -S\rho_{\mathsf{upp}}(Q^T\dot{X}S). \end{array}$$

with upph, ρ_{skew}, ρ_{upp}, lows and upp as in definition 2.9.

Proof:
All matrix decompositions involved have the form $X = YZ$. Since lemma 3.2 guarantees the existence of all derivatives involved, a simple proof consists in verifying that $\frac{d}{dt}(YZ) = \dot{Y}Z + Y\dot{Z}$ yields $\dot{X}$ after substituting the given formulas for $\dot{Y}$ and $\dot{Z}$.

Below, we give a more constructive proof. Since in all cases Y and Z belong to some Lie group, the derivatives $\dot{Y}$ and $\dot{Z}$ can be written as a product $\dot{Y} = YG_Y$ and $\dot{Z} = \tilde{H}_Z Z$, where G_Y and $\tilde{H}_Z$ belong to the corresponding Lie algebras. From $\dot{X} = \dot{Y}Z + Y\dot{Z}$ one finds $\dot{X} = Y(G_Y + \tilde{H}_Z)Z$. To find $\dot{Y}$ and $\dot{Z}$ we have to solve this equation for G_Y and $\tilde{H}_Z$. Now, G_Y and $\tilde{H}_Z$ can be recovered from

$$Y^{-1}\dot{X}Z^{-1} = G_Y + \tilde{H}_Z, \tag{3.2}$$

exploiting the fact that they belong to given Lie algebras.
More specifically, with

$$\begin{array}{rclcrcl} \dot{C} & = & CG_C, & & \dot{C}^T & = & \tilde{H}_{C^T}C^T = {G_C}^T C^T, \\ \dot{L} & = & LG_L, & & \dot{U} & = & \tilde{H}_U U, \\ \dot{Q} & = & QG_Q, & & \dot{R} & = & \tilde{H}_R R. \end{array} \tag{3.3}$$

where $G_C, \tilde{H}_R, \tilde{H}_U \in \mathfrak{g}_{upp}(n)$, $G_L \in \mathfrak{g}_{lows}(n)$, and $G_Q \in \mathfrak{so}(\mathrm{n})$,
equation (3.2) yields

$$\begin{array}{rcl} C^{-T}\dot{X}C^{-1} & = & G_C + {G_C}^T, \\ L^{-1}\dot{X}U^{-1} & = & G_L + \tilde{H}_U, \\ Q^T\dot{X}R^{-1} & = & G_Q + \tilde{H}_R. \end{array}$$

Therefore

$$\begin{array}{rcll} G_C & = & \mathrm{upph}(C^{-T}\dot{X}C^{-1}), & \\ G_L & = & \mathrm{lows}(L^{-1}\dot{X}U^{-1}) & \tilde{H}_U = \mathrm{upp}(L^{-1}\dot{X}U^{-1}), \\ G_Q & = & \rho_{\mathrm{skew}}(Q^T\dot{X}R^{-1}) & \tilde{H}_R = \rho_{\mathrm{upp}}(Q^T\dot{X}R^{-1}). \end{array}$$

Together with (3.3) this proves the formulas for $\dot{C}$, $\dot{L}$, $\dot{U}$, $\dot{Q}$ and $\dot{R}$. The other formulas are proved by applying lemma 3.3.
□

Corollary 3.5 (Rectangular QR decomposition)
Let $X(t) \in \mathbb{R}^{n\times m}$. If $Q(t) = \mathsf{qf}(X(t))$ and $R(t) = \mathsf{rf}(X(t))$,

$$\begin{aligned}\dot{Q} &= Q\rho_{\mathsf{skew}}(Q^T\dot{X}R^{-1}) + (I - QQ^T)\dot{X}R^{-1},\\ \dot{R} &= \rho_{\mathsf{upp}}(Q^T\dot{X}R^{-1})R.\end{aligned}$$

Proof:
One can easily verify that these formulas imply $\frac{d}{dt}(QR) = \dot{X}$. With $I - QQ^T = PP^T$ where $P \in \mathrm{St}(n-m,n)$ and $P^TQ = 0$, one also sees that $\dot{Q} \in T_Q\,\mathrm{St}(m,n)$ (proposition 2.22). A more constructive proof follows directly from lemma 3.4(iii) by considering $X = \bar{X}_{\leqslant m,.}$ where $\bar{X} \in \mathbb{R}^{n\times n}$. If $\bar{X} = \bar{Q}\bar{R}$ then $Q = \bar{Q}_{\leqslant m,.}$ and $R = \bar{R}_{\leqslant m,\leqslant m}$.
□

3.1.2 Tracking the symmetric eigenvalue decomposition and the singular value decomposition

Definition 3.6
(i) If $X \in \mathcal{S}_{symm}(n)$ a symmetric eigenvalue decomposition of X with orthonormal eigenvectors is given by

$$X = P\Lambda P^T,$$

where $P \in \mathsf{O}(n)$ and $\Lambda \in \mathcal{S}_{diag}(n)$.
The columns of P, $p_i = P_{.,i}$ $i = 1,\ldots,n$, are eigenvectors of X.
the diagonal entries of Λ, $\lambda_i = \Lambda_{i,i}$ are the corresponding eigenvalues.

(ii) If $X \in \mathbb{R}^{n\times m}$, the singular value decomposition of X is given by

$$X = U\Sigma V^T,$$

where $U \in \mathsf{O}(n)$, $V \in \mathsf{O}(m)$, $\Sigma \in \mathbb{R}^{n\times m}$, $\Sigma_{i,j} = 0$ iff $i \neq j$, $\Sigma_{i,i} \geqslant 0$, $i = 1,\ldots,m$.
The columns of V, $v_i = V_{.,i}$, $i = 1,\ldots,n$ are the right singular vectors,
the columns $u_j = U_{.,j}$ of U, $j = 1,\ldots,m$ are the left singular vectors,
The diagonal entries of Σ, $\sigma_i = \Sigma_{i,i}$. $i = 1,\ldots,\min(m,n)$ are the singular values.

For the symmetric EVD (with orthonormal eigenvectors) and the SVD we have to take a slightly different approach than for the decompositions of definition 3.1, discussed above. The discussion below is in terms of the EVD, but the same arguments can be given for the SVD. As defined above, the symmetric EVD is not unique. One still has the freedom to choose the order of the eigenvalues. Furthermore the eigenspaces are only defined up to an orthogonal transformation. Even in the case of distinct eigenvalues one can still choose the sign of the corresponding eigenvector.

Any attempt to define a unique decomposition breaks the differentiability of P as a function of X. Consider for instance the differentiable path

$$A(t) = \begin{bmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} \cos(t) & -\sin(t) \\ \sin(t) & \cos(t) \end{bmatrix},$$

where $t \in [0, \pi]$. Clearly $A(0) = A(\pi)$ but the eigenvectors have changed sign. Any uniquely defined EVD would necessarily be discontinuous at some point of this path.

Nevertheless, as is clear from the example, it is still possible to define differentiable paths $P(t)$ and $\Lambda(t)$, defining an EVD at any t. However, the resulting paths $P(t)$ and $\Lambda(t)$ are not the image of $X(t)$ under some globally defined 'eigenvector function' and 'eigenvalue function'. Lemma 3.9 describes how to track $P(t)$ and $\Lambda(t)$, with formulas similar to the ones given for the Cholesky, LU and QR decompositions in lemma 3.4. First, lemma 3.7 gives a necessary condition for the existence of a differentiable symmetric EVD or differentiable SVD along a path $X(t)$.

Lemma 3.7
(i) If $X(t) \in \mathcal{S}_{symm}(n)$ is continuously differentiable in an interval $[a, b]$ and analytic in a neighborhood of any t_0 for which the eigenvalues of X are not distinct, there exists a continuously differentiable EVD

$$X(t) = P(t)\Lambda(t)P(t)^T,$$

where $P(t) \in \mathsf{O}(n)$ and $\Lambda(t) \in \mathcal{S}_{diag}(n)$ describe continuously differentiable paths.

(ii) If $X(t) \in \mathbb{R}^{n \times m}$ is continuously differentiable in an interval $[a, b]$ and analytic in a neighborhood of any t_0 for which the singular values of X are not distinct or where a singular value is 0, there exists a continuously differentiable SVD

$$X(t) = U(t)\Sigma(t)V(t)^T,$$

where $U(t) \in \mathsf{O}(n)$, $V(t) \in \mathsf{O}(m)$ and $\Sigma(t) \in \mathbb{R}^{n \times m}$ with $\Sigma_{i,j}(t) = 0$ if $i \neq j$, $i = 1, \ldots, n$, $j = 1, \ldots, m$, describe continuously differentiable paths.

(iii) If $X(t) \in \mathcal{S}_{symm}(n)$ is continuously differentiable, an EVD $X(t) = P(t)\Lambda(t)P(t)^T$ exists for which $\Lambda(t)$ is continuously differentiable.

(iv) If $X(t) \in \mathbb{R}^{n \times m}$ is continuously differentiable, an SVD $X(t) = U(t)\Sigma(t)V(t)^T$ exists for which $\Sigma(t)$ is continuously differentiable.

Remark 3.8
Note that we do not require the singular values to be nonnegative anymore. As this enhances the possibilities of obtaining a differentiable SVD.

Proof:
(i) The proof follows from [65, Ch.II, Par. 6].
(ii) The proof for the SVD follows from applying part (i) of this lemma to the EVD of

$$Z = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}.$$

A singular value σ of A corresponds to eigenvalues σ and $-\sigma$ for Z. Note that a singular value 0 gives rise to a multiple eigenvalue. (Although working with A^TA or AA^T instead of Z a weaker condition can be obtained for the case $\sigma = 0$ if A is square).
(iii) This follows from [65, Ch.II, Th. 6.8] again considering the EVD of Z for the SVD.
□

Lemma 3.9
(i) Let $X(t) \in \mathbb{R}^{n\times n}$ satisfy the conditions of lemma 3.7(i), and let

$$X(t) = P(t)\Lambda(t)P(t)$$

be a continuously differentiable EVD. Then,

$$\begin{aligned} \dot{\Lambda} &= \mathrm{diag}(M), \\ \dot{P} &= P\Omega, \end{aligned}$$

where $M = P^T\dot{X}P$, and Ω can be derived from M by

$$\begin{aligned} \omega_{i,j}(t) &= 0 \\ &\quad \text{if } i = j, \\ \omega_{i,j}(t) &= m_{i,j}(t)/(\lambda_j(t) - \lambda_i(t)) \\ &\quad \text{if } \lambda_i(t) \neq \lambda_j(t), && i = 1,\ldots,n \\ \omega_{i,j}(t) &= \lim_{\tau\to t} m_{i,j}(\tau)/(\lambda_j(\tau) - \lambda_i(\tau)) && j = 1,\ldots,n. \\ &\quad \text{if } \lambda_i(t) = \lambda_j(t) \text{ for an isolated value of } t, \\ \omega_{i,j}(t) &= -\omega_{j,i} = \text{ arbitrary} \\ &\quad \text{if } \lambda_i(\tau) = \lambda_j(\tau) \text{ for all } \tau \in [a,b] \text{ and } i \neq j. \end{aligned}$$

(ii) Let $X(t) \in \mathbb{R}^{n\times m}$ satisfy the conditions of lemma 3.7(ii), and let

$$X(t) = U(t)\Sigma(t)V(t)^T$$

be continuously differentiable SVD. Assume $m \leqslant n$ (if not, work with X^T, exchanging the role of Ω and Ψ below). Then

$$\begin{aligned} \dot{\Sigma} &= \mathrm{diag}(N), \\ \dot{U} &= U\Psi, \\ \dot{V} &= V\Omega, \end{aligned}$$

where $N = U^T \dot{A} V$, and Ω and Ψ can be derived from N by

$$
\begin{aligned}
\omega_{i,j}(t) &= 0 \\
&\quad \text{if } i = j, \\
\omega_{i,j}(t) &= \frac{\sigma_i(t) n_{i,j}(t) + \sigma_j(t) n_{j,i}(t)}{\sigma_j^2(t) - \sigma_i^2(t)} \\
&\quad \text{if } \sigma_i(t) \neq \sigma_j(t), \\
\omega_{i,j}(t) &= \lim_{\tau \to t} \frac{\sigma_i(\tau) n_{i,j}(\tau) + \sigma_j(\tau) n_{j,i}(\tau)}{\sigma_j^2(\tau) - \sigma_i^2(\tau)} \\
&\quad \text{if } \sigma_i(t) = \sigma_j(t) \text{ for an isolated value of } t, \\
\omega_{i,j}(t) &= -\omega_{j,i}(t) = \text{ arbitrary} \\
&\quad \text{if } \sigma_i(\tau) = \sigma_j(\tau) \text{ for all } \tau \text{ and } i \neq j,
\end{aligned}
$$

$$
\begin{aligned}
\psi_{i,j}(t) &= 0 \\
&\quad \text{if } i = j, \\
\psi_{i,j}(t) &= \frac{\sigma_j(t) n_{i,j}(t) + \sigma_i(t) n_{j,i}(t)}{\sigma_j^2(t) - \sigma_i^2(t)} \\
&\quad \text{if } \sigma_i(t) \neq \sigma_j(t) \text{ and } i, j \leqslant n, \\
\psi_{i,j}(t) &= \frac{n_{i,j}(t)}{\sigma_j(t)} \\
&\quad \text{if } \sigma_j(t) \neq 0 \text{ and } j \leqslant n < i, \\
\psi_{i,j}(t) &= -\frac{n_{j,i}(t)}{\sigma_i(t)} \\
&\quad \text{if } \sigma_i(t) \neq 0 \text{ and } i \leqslant n < j, \\
\psi_{i,j}(t) &= \lim_{\tau \to t} \frac{\sigma_j(\tau) n_{i,j}(\tau) + \sigma_i(\tau) n_{j,i}(\tau)}{\sigma_j^2(\tau) - \sigma_i^2(\tau)} \\
&\quad \text{if } \sigma_i(t) = \sigma_j(t) \text{ for an isolated value of } t \text{ and } i, j \leqslant n, \\
\psi_{i,j}(t) &= \lim_{\tau \to t} \frac{n_{i,j}(\tau)}{\sigma_j(\tau)} \\
&\quad \text{if } \sigma_j(t) = 0 \text{ for an isolated value of } t \text{ and } j \leqslant n < i, \\
\psi_{i,j}(t) &= -\psi_{j,i}(t) = -\lim_{\tau \to t} \frac{n_{j,i}(\tau)}{\sigma_i(\tau)} \\
&\quad \text{if } \sigma_i(t) = 0 \text{ for an isolated value of } t \text{ and } i \leqslant n < j, \\
\psi_{i,j}(t) &= \text{ arbitrary} \\
&\quad \text{if } \sigma_i(\tau) = \sigma_j(\tau) \text{ for all } \tau \text{ and } i, j \leqslant n, \\
\psi_{i,j}(t) &= \text{ arbitrary} \\
&\quad \text{if } \sigma_j(\tau) = 0 \text{ for all } \tau \text{ and } j \leqslant n < i, \\
\psi_{i,j}(t) &= -\psi_{j,i}(t) \\
&\quad \text{if } \sigma_i(\tau) = 0 \text{ for all } \tau \text{ and } i \leqslant n < j, \\
\psi_{i,j}(t) &= \text{ arbitrary} \\
&\quad \text{if } i > n \text{ and } j > n \text{ and } i \neq j.
\end{aligned}
$$

Remark 3.10
Note that the analyticity condition implies that if $\lambda_i(t) = \lambda_j(t)$ in a nonzero interval, this must necessarily be the case in the whole interval $[a, b]$. A similar remark holds for the SVD.

Remark 3.11
Note that the lemma seems more complicated than it is, at least for the (general) case of distinct eigenvalues or singular values. In the case of distinct *eigenvalues*, for instance, Ω is obtained as an elementwise scaling of $M = P^T \dot{X} P$.

Proof:
(i) The proof is similar to the proof of lemma 3.4. Lemma 3.7 guarantees the existence of $P(t)$ and $\Lambda(t)$. $\dot{P}$ can be written as $\dot{P} = P\Omega$, where $\Omega \in \mathfrak{so}(n)$. It follows that

$$\begin{aligned} \dot{X} &= \tfrac{d}{dt}(P\Lambda P^T) \\ &= \dot{P}\Lambda P^T + P\dot{\Lambda}P^T + P\Lambda\dot{P}^T \\ &= P([\Omega,\Lambda] + \dot{\Lambda})P^T. \end{aligned} \tag{3.4}$$

Let $T = [\Omega, \Lambda] = d_C(\Omega)$ with $C_{i,j} = \lambda_j - \lambda_i$ as in lemma2.11(ii). Since T has zeros on the diagonal, T and $\dot{\Lambda}$ can be found as

$$\begin{aligned} T &= \text{offdiag}(M) = \Omega\Lambda - \Lambda\Omega, \\ \dot{\Lambda} &= \text{diag}(M), \end{aligned} \tag{3.5}$$

where offdiag $=$ id $-$ diag, and $M = P^T\dot{X}P$.

The exceptions for multiple eigenvalues arise when we want to extract Ω from T (or M):

$$m_{i,j} = t_{i,j} = \omega_{i,j}(\lambda_j - \lambda_i) \text{ for } i \neq j. \tag{3.6}$$

If the eigenvalues $\lambda_i, i = 1, \ldots, n$ are distinct, Ω can be found from $\omega_{i,j} = m_{i,j}/(\lambda_j - \lambda_i)$ if $i \neq j$, and $\omega_{i,i} = 0$ (since Ω is skew symmetric).

For the case of $\lambda_i = \lambda_j$ at an isolated time instant t_0, since lemma 3.7 guarantees the existence of a continuous $\Omega(t)$, $\omega_{i,j}(t_0)$ can be found by continuous extension of $\omega_{i,j}(t)$.

If $\lambda_i = \lambda_j$ over some time interval, P is no longer uniquely defined. It follows from interpretation 2.14 that the coefficients $\omega_{i,j}$ (with $\lambda_i = \lambda_j$) now correspond to rotations of the corresponding eigenvectors within the corresponding eigenspace. Any (continuously differentiable) choice of $\omega_{i,j} = -\omega_{j,i}$ results in a suitable $P(t)$.

(ii) It is well known that the right singular vectors of A are the eigenvectors of A^TA, the left singular vectors are the eigenvectors of AA^T, and the singular values squared are the eigenvalues of both AA^T and A^TA (from $A^TA = V\Sigma^T\Sigma V^T$ and $AA^T = U\Sigma\Sigma^TU^T$). Consequently, $\dot{V}$ can be found from lemma 3.9, by setting $D = \Sigma^T\Sigma$, $P = V$ and $M = V^T\frac{d}{dt}(A^TA)V = V^T(\dot{A}^TA + A^T\dot{A})V = N^T\Sigma + \Sigma^TN$, where $N = U^T\dot{A}V$. This results immediately in the given formula for $\dot{V}$. Similarly, setting $D = \Sigma\Sigma^T$ (yielding $m - n$ extra eigenvalues 0), $P = U$ and $M = U^T\frac{d}{dt}(A^TA)U = U^T(\dot{A}\dot{A}^T + A\dot{A}^T)U = N\Sigma^T + \Sigma N^T$, results in the formulas for $\dot{U}$.

To obtain the formula for $\dot{\Sigma}$, it is easier to follow directly the same approach as for lemmas 3.4 and 3.9. Writing $\dot{U}$ as $\dot{U} = U\Psi$ and $\dot{V}$ as $\dot{V} = V\Omega$, where Ψ and Ω are skew symmetric (like Ω in lemma 3.9), it follows that $\dot{A} = \dot{U}\Sigma V^T +$

$U\dot{\Sigma}V^T + U\Sigma\dot{V}^T = U(\Psi\Sigma - \Sigma\Omega + \dot{\Sigma})V^T$. Finally, $\dot{\Sigma} = \mathrm{diag}(U^T\dot{A}V) = \mathrm{diag}(N)$. □

3.2 Matrix decompositions as nonlinear coordinate transformations

In this section we consider the QR decomposition and the symmetric EVD as nonlinear coordinate transformations of matrix space, that can be regarded as generalizations of polar coordinates in the plane. A first motivation is that it gives an intuitive geometric picture of lemmas 3.4 and 3.9, placing them in a well understood context, and clarifying the sometimes troublesome issue of multiple eigenvalues (in the symmetric case). A second motivation is that the pictures below should be helpful in understanding the central ideas behind some results, presented in later chapters.

3.2.1 The polar coordinate picture

For this purpose, we define polar coordinates of a point $x \in \mathbb{R}^2$ as the tuple (q, r), satisfying $x = qr$ where the unit vector $q = [\cos\theta \ \sin\theta]^T$ represents the *angular part* and $r \in \mathbb{R}^+$ is the *radial part* (figure 3.1). Note that q is left undetermined if $x = 0$. Lines of constant q are *rays* $\mathcal{R}_q$, that is half straight lines through q and bounded by the origin. Lines of constant r are *circles* $\mathcal{Q}_r$ with radius r and centered at the origin.

If $x(t)$ is a path through x, one can find back $\dot{q}$ and $\dot{r}$ from $\dot{x}$. For, in $\dot{x} = q\dot{r} + \dot{q}r$, the component $\dot{q}r$ is tangent to $\mathcal{Q}_r$, and the component $q\dot{r}$ is tangent to the $\mathcal{R}_q$. They can be found back by projection onto the tangent spaces $T_x\mathcal{Q}_r = \mathcal{R}_q^\perp$ and $T_x\mathcal{R}_q = \mathcal{R}_q$.

$$\begin{aligned} \dot{q}r &= (I - qq^T)\dot{x}, \\ q\dot{r} &= qq^T\dot{x}. \end{aligned}$$

Therefore

$$\begin{aligned} \dot{r} &= q^T\dot{x}, \\ \dot{q} &= \tfrac{1}{r}(I - qq^T)\dot{x}. \end{aligned}$$

It is common practice to introduce a new coordinate system at every point x for tangent vectors (velocities) at x, with one basis vector u_t tangent to the circle $\mathcal{Q}_r$ and one basis vector u_r tangent to the ray $\mathcal{R}_q$. The corresponding velocity components $v_t = \dot{q}r$ and $v_r = q\dot{r}$ are called tangential and radial (or normal).

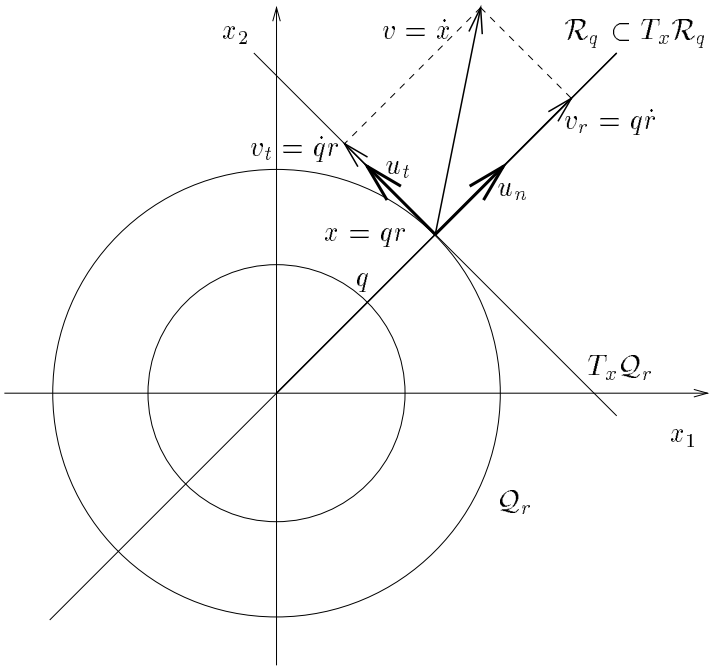


Figure 3.1: Polar coordinates in the plane, rays, circles, tangential and radial velocity components

3.2.2 The QR decomposition as a coordinate transformation

In a similar way the QR decomposition can be regarded as a nonlinear coordinate transformation of $\mathsf{GL}(n)$ (figure 3.2).

For the QR decomposition, we will refer to $Q = \mathrm{qf}(X)$ as the *angular part* and $R = \mathrm{rf}(X)$ as the *radial part* of X. Manifolds $\mathcal{R}_Q$ of constant Q (rays) are isometric to $\mathcal{G}_{upp}(n)^+$, as they are the image of $\mathcal{G}_{upp}(n)$ under the orthogonal transformation l_Q. That is, they are the sum of a subspace of dimension $\frac{1}{2}n(n-1)$ and n open half spaces of dimension 1 (corresponding to the positive diagonal elements of $R \in \mathcal{G}_{upp}(n)^+$). Manifolds $\mathcal{Q}_R$ of constant R are the images of $\mathsf{O}(n)$ under the nonsingular (but generally not orthogonal) linear transformation r_R. If $X(t)$ is a path in $\mathsf{GL}(n)$, $\dot{Q}$ and $\dot{R}$ are found from $\dot{X}$ by lemma 3.4(iii). The geometrical interpretation is as follows. In $\dot{X} = \dot{Q}R + Q\dot{R}$, the component $\dot{Q}R$ is tangent to $\mathcal{Q}_R$, and the component $Q\dot{R}$ is tangent to $\mathcal{R}_Q$. The component $\dot{Q}R$ can be found by projection onto $T_X\mathcal{Q}_R$ parallel to $T_X\mathcal{R}_Q$, whereas $Q\dot{R}$ is found by projection onto $T_X\mathcal{R}_Q$ parallel to $T_X\mathcal{Q}_R$. By differentiation of paths $X(t) = Q(t)R$ on $\mathcal{Q}_R$ and $X(t) = QR(t)$ on $\mathcal{R}_Q$ one finds $T_X\mathcal{Q}_R = \{QG_QR | G_Q \in \mathfrak{so}(n)\}$

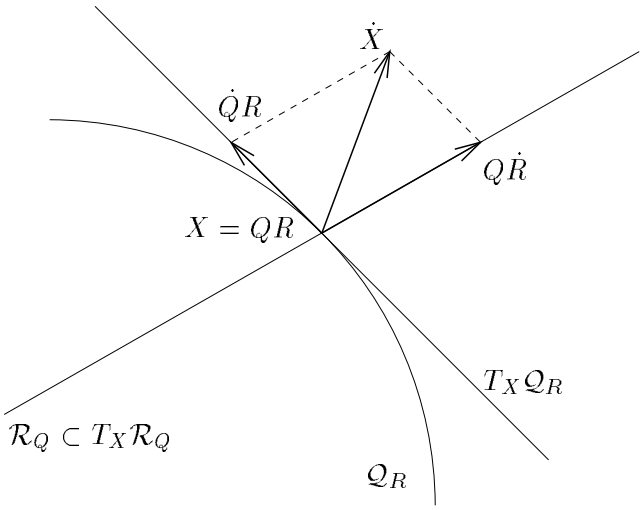


Figure 3.2: Schematic picture of the QR decomposition as a coordinate transformation. Manifold $\mathcal{R}_Q$ of constant Q, manifold $\mathcal{Q}_R$ of constant R, their tangent spaces, and tangential and radial velocity components.

and $T_X\mathcal{R}_Q = \{Q\tilde{H}R | \tilde{H} \in \mathfrak{g}_{upp}(n)\}$. And from lemma 3.4(iii), we have

$$\begin{array}{rclcl} \dot{Q}R & = & Q\rho_{\text{skew}}(Q^T\dot{X}R^{-1})R & = & (e_{Q,R} \circ \rho_{\text{skew}} \circ e_{Q,R}^{-1})(\dot{X}), \\ Q\dot{R} & = & Q\rho_{\text{upp}}(Q^T\dot{X}R^{-1})R & = & (e_{Q,R} \circ \rho_{\text{upp}} \circ e_{Q,R}^{-1})(\dot{X}), \end{array} \tag{3.7}$$

with $e_{Q,R}$ as in definition 2.9. One can easily check that these are indeed the projections mentioned above (note that $e_{Q,R}$ maps $\mathfrak{so}(n)$ to $T_X\mathcal{Q}_R$ and $\mathfrak{g}_{upp}(n)$ to $T_X\mathcal{R}_Q$).

As for the polar coordinates we introduce a local coordinate system at X for velocities $\dot{X} \in T_X\mathsf{GL}(n) = \mathbb{R}^{n\times n}$. It is possible to get a finer picture than just a tangential and a radial component. The following lemma gives a local basis with a natural interpretation.

Lemma 3.12
Let $X(t) \in \mathsf{GL}(n)$ describe a path through $X = X(0)$, $Q = \mathsf{qf}(X)$, $R = \mathsf{rf}(X)$. Let $\dot{X} = \dot{X}(0)$ be the velocity at X. Let

$$\begin{array}{rcl} U_t^{(i,j)} & = & q_j r_i^T - q_i r_j^T, \\ U_r^{(i,j)} & = & q_i r_j^T, \end{array}$$

where $q_i = Q_{:,i}$ and $r_j^T = R_{j,:}$
and let $Z = e_{Q,R}^{-1}(\dot{X}) = Q^T\dot{X}R^{-1}$.

(i) The matrices $U_t^{(i,j)}$, $i = 1, \ldots, n$, $j = i+1, \ldots n$ and $U_r^{(i,j)}$, $i = 1, \ldots, n$, $j = i, \ldots, n$ form a basis of $T_X \mathsf{GL}(n) = \mathbb{R}^{n \times n}$.

(ii) $\dot{X} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} -z_{j,i} U_t^{(i,j)} + \sum_{i=1}^{n} \sum_{j=i}^{n} (z_{i,j} + z_{j,i}) U_r^{(i,j)}$ (with $z_{i,j} = Z_{i,j}$).

(iii) A (tangential) velocity $\omega_{i,j} U_t^{(i,j)} \in T_X \mathcal{Q}_R$ ($\omega_{i,j} \in \mathbb{R}$) corresponds to an instantaneous planar rotation of Q affecting only the columns q_i and q_j according to

$$\begin{array}{rcl} \dot{q}_i & = & -\omega_{i,j} q_j, \\ \dot{q}_j & = & \omega_{i,j} q_i. \end{array} \tag{3.8}$$

(iv) A (radial) velocity $\tilde{h}_{i,j} U_r^{(i,j)} \in T_x \mathcal{R}_Q$ ($\tilde{h}_{i,j} \in \mathbb{R}$) only affects the row r_i^T according to

$$\dot{r}_i^T = \tilde{h}_{i,j} r_j^T. \tag{3.9}$$

Proof:
(i) The given basis is the image under the nonsingular transformation $e_{Q,R}$ of the basis consisting of $E^{(i,j)} - E^{(j,i)}$, $i = 1, \ldots, n$, $j = i+1, \ldots, n$ and $E^{(i,j)}$, $i = 1, \ldots, n$, $j = i, \ldots, n$, of $\mathbb{R}^{n \times n}$.
(ii) $\dot{X} = e_{Q,R}(Z) = e_{Q,R}(\rho_{\mathrm{skew}}(Z) + \rho_{\mathrm{upp}}(Z)) =$
$\sum_{i=1}^{n} \sum_{j=i+1}^{n} (-z_{j,i}) e_{Q,R}(E^{(i,j)} - E^{(j,i)}) + \sum_{i=1}^{n} \sum_{j=i}^{n} (z_{i,j} + z_{j,i}) e_{Q,R}(E^{(i,j)})$.
(iii,iv) this follows by application of lemma 3.4(iii)
□

Remark 3.13
Note the double role of $e_{Q,R}$. In equation (3.7) it is a linear transformation mapping I to X and the tangent spaces $\mathfrak{so}(n)$ and $\mathfrak{g}_{upp}(n)$ at I to the tangent spaces $T_X \mathcal{Q}_R$ and $T_X \mathcal{R}_Q$ at X. In lemma 3.12 $e_{Q,R}^{-1}$ plays the role of a coordinate transformation of the tangent space $T_X \mathsf{GL}(n)$, (transforming velocities, without taking them to another point).

Remark 3.14
The QR decomposition in the matrix Lie group

$$\{ \begin{bmatrix} x & y \\ -y & x \end{bmatrix} | x, y \in \mathbb{R} \setminus \{0\} \}$$

corresponds exactly to polar coordinates, by identifying

$$\begin{bmatrix} x & y \\ -y & x \end{bmatrix}$$

with $[x \ y]^T \in \mathbb{R}^2$.

3.2.3 The symmetric EVD as a coordinate transformation

Figure 3.4 shows the 'real' picture of the 3-dimensional space $\mathcal{S}_{symm}(2)$. The EVD in $\mathcal{S}_{symm}(2)$ corresponds to a cylindrical coordinate transformation. A general matrix $X \in \mathcal{S}_{symm}(2)$ with eigenvalues λ_1, λ_2 has the form

$$\begin{aligned} X &= P\Lambda P^T \\ &= \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} \\ &= \tfrac{\lambda_1+\lambda_2}{2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \tfrac{\lambda_1-\lambda_2}{2} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cos(2\phi) + \tfrac{\lambda_1-\lambda_2}{2} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \sin(2\phi). \end{aligned}$$

The isospectral manifold $\mathcal{I}_\Lambda$ (fixed λ_1, λ_2, variable ϕ) is a circle with radius $\frac{1}{\sqrt{2}}|\lambda_1 - \lambda_2|$ centered around the vertical symmetry axis $\mathbb{R}I$, and becoming a point on the symmetry axis if $\lambda_1 = \lambda_2$.

Manifolds of constant P (fixed ϕ, variable λ_1, λ_2) are planes through the symmetry axis. The similarity transformation s_P is the rotation around the symmetry axis. It takes the plane of diagonal matrices to the manifold of constant P.

In planes orthogonal to the symmetry axis we get the polar coordinate picture again. If $X(t)$ is a path in $\mathcal{S}_{symm}(2)$, $\dot{P}$ and $\dot{\Lambda}$ are found from $\dot{X}$ by lemma 3.9. The rotational velocity $\omega_{i,j}$ of the eigenvectors p_1 and p_2 (see interpretation 2.14) turns out to correspond, up to a constant factor 2, to the rotational velocity of the path $X(t)$ around the symmetry axis, and $m_{1,2}$ in lemma 3.9 corresponds (up to a factor $\sqrt{2}$) to the tangential velocity component in the coordinate picture. The relation between both is given by $m_{1,2} = \omega_{1,2}(\lambda_2 - \lambda_1)$ (cf. equation (3.6)).

The more general picture is given schematically in figure 3.3, by analogy with figures 3.1 and 3.2.We consider P as the angular part and Λ as the radial part (figure 3.3).

Manifolds of constant Λ are $\mathcal{I}_\Lambda$, as described in section 2.3. If X has distinct eigenvalues, manifolds $\mathcal{L}_P$ of constant P are the n-dimensional subspaces $\mathcal{L}_P = \mathcal{C}_X = s_{P^T}(\mathcal{S}_{diag}(n))$. If X has multiple eigenvalues, different choices of P lead to different sets of constant P through X. They are given by $\mathcal{L}_P = \mathcal{C}_{PDP^T} = s_{P^T}(\mathcal{S}_{diag}(n))$, where D is a diagonal matrix with distinct entries, say $D = \mathrm{diag}\{1, \ldots, n\}$. This situation is comparable to the situation of the origin in polar coordinates.

If $X(t)$ is a path in $\mathcal{S}_{symm}(n)$, $\dot{P}$ and $\dot{\Lambda}$ are found from $\dot{X}$ by lemma 3.9. If $\dot{P} = P\Omega$ we know from equation (3.4) in lemma 3.9 that

$$\dot{X} = P[\Omega, \Lambda]P^T + P\dot{\Lambda}P^T.$$

The component $P[\Omega, \Lambda]P^T = [P\Omega P^T, X]$ lies in $T_X\,\mathcal{I}_\Lambda$, whereas the component $P\dot{\Lambda}P^T$ lies in $T_X\mathcal{L}_P = \mathcal{L}_P$.

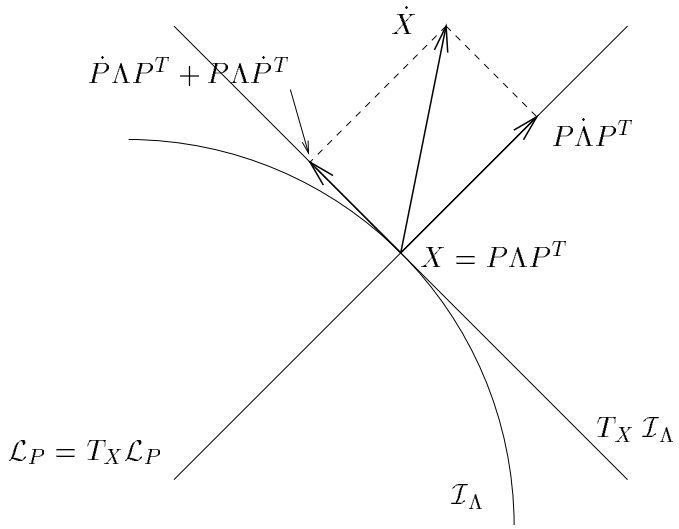


Figure 3.3: Schematic picture of the EVD as a coordinate transformation, manifold $\mathcal{L}_P$ of constant P, manifold $\mathcal{I}_\Lambda$ of constant Λ, their tangent spaces, and tangential and radial velocity components.

From equation (3.5) in lemma 3.9, we have

$$\begin{array}{rclcl} P[\Omega,\Lambda]P^T & = & P\,\mathrm{offdiag}(P^T\dot{X}P)P^T & = & (s_P^{-1}\circ\mathrm{offdiag}\circ s_P)(\dot{X}), \\ P\dot{\Lambda}P^T & = & P\,\mathrm{diag}(P^T\dot{X}P)P^T & = & (s_P^{-1}\circ\mathrm{diag}\circ s_P)(\dot{X}), \end{array} \tag{3.10}$$

with s_P as in definition 2.9. These are clearly the orthogonal projections of $\dot{X}$ onto the tangent spaces $T_X\,\mathcal{I}_\Lambda$ and $T_X\mathcal{L}_P = \mathcal{L}_P$. (We do not go into details for the case of repeated eigenvalues - see lemma 3.15(v,vi), statement 3.18).

Like for the polar coordinates and the QR decomposition we introduce a local orthogonal coordinate system at X for velocities in $\mathcal{S}_{symm}(n)$.

Lemma 3.15
Let $X(t) \in \mathcal{S}_{symm}(n)$ describe a path through $X = X(t_0)$ with EVD $X = PDP^T$. Let $\dot{X} = \dot{X}(t_0)$ be the velocity at X. Let

$$\begin{array}{rcl} W_t^{(i,j)} & = & p_i p_j^T + p_j p_i^T, \\ W_r^{(i)} & = & p_i p_i^T, \end{array}$$

where $p_i = P_{\cdot,i}$
and let $M = s_P(\dot{X}) = P^T\dot{X}P$

(i) The matrices $\frac{1}{\sqrt{2}}W_t^{(i,j)}$, $i = 1,\ldots,n$, $j = i,\ldots,n$, and $W_r^{(i)}$, $i = 1,\ldots,n$ form an orthonormal basis of $\mathcal{S}_{symm}(n)$.

(ii) $\dot{X} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} m_{i,j} W_t^{(i,j)} + \sum_{i=1}^{n} m_{i,i} W_r^{(i)}$ (with $m_{i,j} = M_{i,j}$).

(iii) A (radial) velocity $l_i W_r^{(i)} \in T_X \mathcal{L}_P$ ($l_i \in \mathbb{R}$) only affects the eigenvalue λ_i according to

$$\dot{\lambda}_i = l_i.$$

(iv) If $\lambda_i \neq \lambda_j$, a (tangential) velocity $(\lambda_j - \lambda_i)\omega_{i,j} W_t^{(i,j)} \in T_X \mathcal{I}_\Lambda$ ($\omega_{i,j} \in \mathbb{R}$) corresponds to an instantaneous planar rotation of P affecting p_i and p_j according to

$$\begin{aligned} \dot{p}_i &= -\omega_{i,j} p_j, \\ \dot{p}_j &= \omega_{i,j} p_i. \end{aligned} \tag{3.11}$$

(It is of course also possible that eigenvectors other than p_i and p_j evolve inside their eigenspace if they have multiplicity > 1 in a neighborhood of t_0).
If $\lambda_i = \lambda_j$, a velocity $m_{i,j} W_t^{(i,j)}$ cannot always be interpreted as an evolution of eigenvectors p_i and p_j (see (v,vi)), and it is impossible to recover $\omega_{i,j}$ (if existent at all) from knowledge of $\dot{X}$ at t_0 only.

(v) a necessary condition for P to have a differentiable evolution is

$$m_{i,j} = 0 \text{ if } \lambda_i = \lambda_j. \tag{3.12}$$

(vi) Let $\tilde{P}\Lambda\tilde{P}^T$ be a symmetric EVD of X, and let $\tilde{M} = s_{\tilde{P}}(\dot{X}) = \tilde{P}^T \dot{X} \tilde{P}$. Let α be the set of indices corresponding to a multiple eigenvalue with multiplicity κ ($\lambda_i = \lambda_j$ for $i, j \in \alpha$, $\sharp\alpha = \kappa$) and let QLQ^T be a symmetric EVD of $\tilde{M}_{\alpha,\alpha}$.
Then the columns of the matrix P, defined by

$$\begin{aligned} P_{:,\alpha} &= \tilde{P}_{:,\alpha} Q, \\ P_{:,\bar{\alpha}} &= \tilde{P}_{:,\bar{\alpha}}, \end{aligned} \tag{3.13}$$

form a set of eigenvectors of X with $M_{\alpha,\alpha} = L$, satisfying the conditions (3.12) for $i, j \in \alpha$. ($\bar{\alpha}$ denotes the complement of α in $\{1, \ldots, n\}$).

Proof:
(i) The given basis is the image under the orthogonal linear transformation s_{P^T} of the orthonormal basis $\frac{1}{\sqrt{2}}(E^{(i,j)} + E^{(j,i)})$, $i = 1, \ldots, n$, $j = i+1, \ldots, n$, $E^{(i,i)}$, $i = 1, \ldots, n$ of $\mathcal{S}_{symm}(n)$.
(ii) $\dot{X} = s_{P^T}(M) = s_{P^T}(\text{offdiag}(M) + \text{diag}(M)) =$
$\sum_{i=1}^{n} \sum_{j=i+1}^{n} m_{i,j} s_{P^T}(E^{(i,j)} - E^{(j,i)}) + \sum_{i=1}^{n} m_{i,i} s_{P^T}(E^{(i,i)})$.
(iii,iv) This follows by application of lemma 3.9.
(v) It follows from equation (3.5) in lemma 3.9, that $m_{i,j} = \omega_{i,j}(\lambda_j - \lambda_i) = 0$ if $\lambda_i = \lambda_j$.
(vi) Since Q in (3.13) performs an orthogonal transformation on eigenvectors corresponding to the same eigenvalue, the columns of P are still orthonormal eigenvectors of X. And, $M_{\alpha,\alpha} = P_{:,\alpha}^T \dot{X} P_{:,\alpha} = Q^T \tilde{P}_{:,\alpha}^T \dot{X} \tilde{P}_{:,\alpha} Q = Q^T \tilde{M} Q = L$.

Since L is diagonal the conditions 3.12 are satisfied.
□

Remark 3.16
Note that s_P, like $e_{Q,R}$ in remark 3.13, plays a double role. In equation (3.7) it is the orthogonal transformation mapping I to X and the tangent space $T_\Lambda \mathcal{I}_\Lambda$ and the ray $\mathcal{L}_I = \mathcal{S}_{diag}(n)$ at Λ to the tangent space $T_X \mathcal{I}_\Lambda$ and the ray $\mathcal{L}_P$ at X. In lemma 3.12 s_P^T plays the role of a coordinate transformation of the tangent space $T_X \mathbb{R}^{n \times n}$.

Remark 3.17
In a similar way, also the SVD can be seen as a decomposition of a matrix into two angular parts and a radial part. And a natural local basis can be identified, corresponding to changes to one singular value or a pair of singular vectors only.

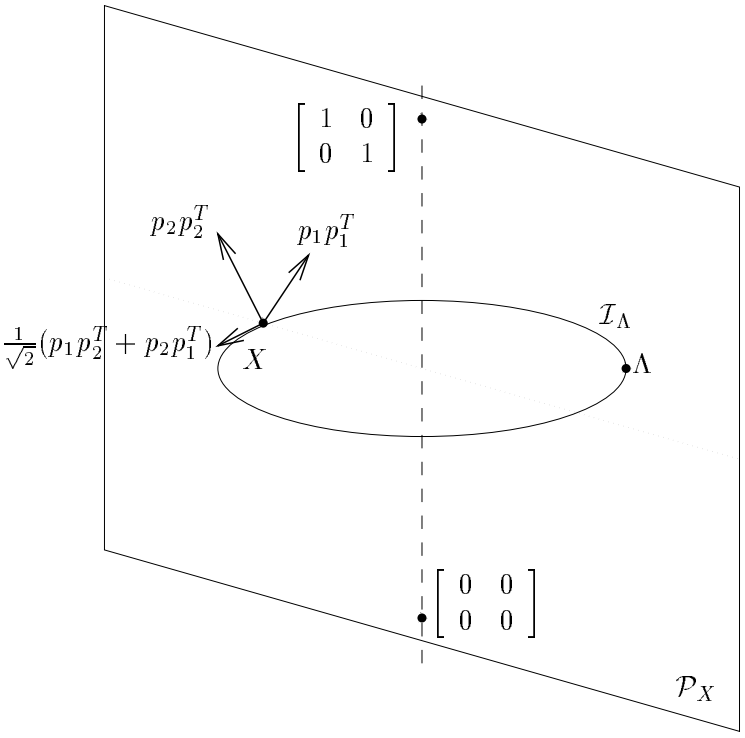


Figure 3.4: The EVD as a cylindrical coordinate transformation in $\mathcal{S}_{symm}(2)$. The symmetry axis, consisting of the multiples of the identity matrix, is pictured vertically. Manifolds of constant P are planes through the symmetry axis. Isospectral manifolds are circles in horizontal planes, centered around the symmetry axis.

Finally, the cylindrical coordinate picture (figure 3.4) can also bring some clarification in the case of multiple eigenvalues, and the analyticity condition of lemma 3.9. Although we do not want to overemphasize its importance for this work, this case does occur as a side aspect more than once. The following discussion has a more intuitive character.

Statement 3.18
A two times continuously differentiable path $X(t)$ has a continuously differentiable symmetric EVD except for contrived examples.

First, the dimension of the manifold of matrices with eigenvalues with multiplicity 2 (the symmetry axis in figure 3.4) is two less than the dimension of the total space $\mathcal{S}_{symm}(n)$. Therefore 'almost all' paths in the space simply avoid this manifold. In figure 3.4, this manifold is the symmetry axis, containing the multiples of the identity matrix. Consider two matrices $\mathrm{diag}\{a, b\}$ and $\mathrm{diag}\{b, a\}$ in $\mathcal{S}_{symm}(2)$. The straight path through these matrices will intersect the symmetry axis at $\mathrm{diag}\{\frac{a+b}{2}, \frac{a+b}{2}\}$. But for small deviations the eigenvalues will not cross. Instead the eigenvectors will rotate rather suddenly over about 90 degrees in the neighborhood of the symmetry axis.

But even among paths, intersecting the symmetry axis in figure 3.4, problems are exceptional. Interpreting the issue in terms of cylindrical coordinates, eigenvectors correspond to angles and gaps between eigenvalues to radii (moduli). Differentiability of eigenvectors corresponds to differentiability of angles. But first consider the issue of continuity of eigenvectors (angles). Discontinuity means that the path should leave the origin with a different angle than it arrived with. Assuming that the path is continuously differentiable (its 'Cartesian' coordinates are continuously differentiable to time), this can only happen if the radial velocity goes to 0. Similarly, one can prove that if the path is twice continuously differentiable, the angular *velocity* can only be discontinuous if the radial velocity goes to 0. In conclusion, for the eigenvectors to be no longer continuously differentiable, the (twice continuously differentiable) path does not only have to cross the symmetry axis, but it has to make second order contact with it (including crossing at 0 speed), and even that is far from sufficient.

The higher-dimensional case is of course more complicated but for a double eigenvalue one always gets the polar coordinate picture in the subspace spanned by $p_i p_j^T + p_j p_i^T$ and $p_i p_i^T - p_j p_j^T$, while evolutions in the orthogonal complement of this space preserve (at least locally) the multiplicity of the double eigenvalue. For eigenvalues with higher multiplicity the situation can of course be more complicated, but it is also even more contrived.

Of course the notion 'contrived' is not well defined, and in specific cases there can always be structural reasons making such 'contrived' examples more 'natural'.

3.3 Gradients and gradient flows

A central position in the field of continuous-time algorithms is taken by gradients and gradient flows. A gradient flow can be thought of as the continuous-time counterpart of a discrete-time gradient ascent or descent algorithm (steepest ascent/descent algorithm). In this work, gradient flows do not play the same central role, and we will pay less attention to them than is usual. In this section, we put emphasis on the use of the previously exposed theory (of the present and previous chapter) for obtaining gradients for linear algebra related optimization problems. Even for the design of *discrete-time* algorithms, the continuous-time view is never far away if one thinks of tangent vectors as velocities of paths on a manifold.

First we define gradients and gradient flows. To this end we need inner products on Riemannian manifolds. Then we give examples of derivations of gradients.

In order to define gradients of functions over a manifold $\mathcal{M}$, one has to define an inner product between tangent vectors at any point.

Definition 3.19 (Riemannian manifold)
A *Riemannian manifold* is a differentiable manifold, endowed with a family of inner products $\langle .,. \rangle_x$ between tangent vectors at x, depending smoothly on x.

As most manifolds considered in this text are embedded in a matrix space, a natural choice for the inner products in the tangent spaces consists in considering the tangent vectors as vectors in the embedding matrix space, and borrowing the standard inner product. However, other inner products are often useful too. The following lemma gives useful forms for the standard inner product on $\mathsf{O}(n)$ and the Stiefel Manifold.

Lemma 3.20
(i) The standard inner product in $T_Q\mathsf{O}(n)$ between tangent vectors $T_1 = Q\Omega = \tilde{\Omega}Q$ and $T_2 = Q\Omega_2 = \tilde{\Omega}_2 Q$ can be cast as

$$\langle T_1, T_2 \rangle = \langle \Omega_1, \Omega_2 \rangle = \langle \tilde{\Omega}_1, \tilde{\Omega}_2 \rangle.$$

(ii) The standard inner product in $T_A\,\mathsf{St}(m,n)$ between tangent vectors $T_1 = A\Omega_1 + B\Psi_2$ and $T_2 = A\Omega_2 + B\Psi_2$ can be cast as

$$\langle T_1, T_2 \rangle = \langle \Omega_1, \Omega_2 \rangle + \langle \Psi_1, \Psi_2 \rangle.$$

Proof:
(i,ii) This follows directly by application of lemma 2.11(vii), and $Q^TQ = I$, $A^TA = I$, $B^TB = I$ and $A^TB = 0$.

□

Definition 3.21 (gradient)
The gradient $\mathrm{grad}_x f$ of a differentiable function f over a Riemannian manifold $\mathcal{M}$ at a point x is the unique *tangent* vector in $T_x\mathcal{M}$ satisfying

$$\mathrm{D}f|_x(T) = \langle \mathrm{grad}_x f, T \rangle,$$

for all $T \in T_x\mathcal{M}$, where $Df|_x(T)$ is the derivative of f at x in the direction of T. We will often use an equivalent condition, thinking of the tangent vector T as a velocity $\dot{x} = \dot{x}(0)$ of a path $x(t) \in \mathcal{M}$ through $x = x(0)$,

$$\dot{f} = \langle \mathrm{grad}_x f, \dot{x} \rangle, \tag{3.14}$$

where $\dot{f}$ denotes $\frac{d}{dt} f(x(t))|_{t=0}$.

Definition 3.22 (gradient flow)
The *gradient ascent flow* of a differentiable function f on a Riemannian manifold $\mathcal{M}$ is generated by the system

$$\dot{x} = \mathrm{grad} f.$$

The *gradient descent flow* is generated by

$$\dot{x} = -\mathrm{grad} f.$$

The key property of a gradient (ascent) flow is that the evolution of f along a trajectory is given by

$$\dot{f} = \langle \mathrm{grad} f, \mathrm{grad} f \rangle \geqslant 0. \tag{3.15}$$

That is, f is steadily non-decreasing along any trajectory. Similarly, for a gradient descent flow, one finds from $\dot{f} = -\langle \mathrm{grad} f, \mathrm{grad} f \rangle$, that f is steadily non-increasing. Several conditions can be set up under which (3.15) guarantees convergence, by using f as a Lyapunov function. The following proposition is slightly adapted from [55].

Proposition 3.23
Let $f : \mathcal{M} \to \mathbb{R}$ be a smooth function on a Riemannian manifold with compact sublevel sets, i.e. for all $c \in \mathbb{R}$ $\{x \in \mathcal{M} | f(x) \leqslant c\}$ is a (possibly empty) compact subset of $\mathcal{M}$. Then every solution $x(t) \in \mathcal{M}$ of the gradient descent flow of f exists for all $t \geqslant 0$. Furthermore, $x(t)$ converges to a connected component of the set of critical points of f as $t \to +\infty$.

Below we discuss the relevance of the theory of the present and previous chapter for the derivation of gradients and the design of optimization algorithms, we distinguish the following three aspects.

Statement 3.24
(i) The geometrical interpretations provide insight in some optimization problems. As examples, below we discuss the orthogonal approximation problem 3.26 and we refer to the derivation of the double bracket flow in section 4.2.

(ii) The derivation of gradients, tangent to equality constraints, that are regular manifolds as in section 2.3, is achieved by simple application of the ideas of chapter 2. We refer to the same examples as for (ii) and to lemma 7.17 and theorem 11.5.

(iii) Gradients of functions defined in terms of matrix decompositions can be obtained using the lemmas of section 3.1. As an example we consider the gradient of two distance measures between subspaces. A simpler example, in combination with (ii), is found in theorem 11.5.

Remark 3.25
Some results of continuous-time matrix algorithms are also useful in the context of optimization problems with *inequality* constraints, like linear programming [8].

Example 1: orthogonal approximation

As a first simple example we consider classical problem of approximating a matrix by an orthogonal matrix (related to the Procrustes problem([48, p.582])).

Definition 3.26 (Orthogonal approximation problem)
Given $A \in \mathbb{R}^{n\times n}$ find

$$Q^{opt} = \arg \min_{Q\in \mathsf{O}(n)} \|Q - A\|_F.$$

First, note that this problem can be solved *geometrically*, by recalling proposition 2.12. $\|Q - A\|_F$ is (locally) minimal for $Q \in \mathsf{O}(n)$ if $Q - A$ lies in the normal space of $\mathsf{O}(n)$ at Q, that is if $Q^T(Q - A) \in \mathcal{S}_{symm}(n)$, or simply if $Q^T A \in \mathcal{S}_{symm}(n)$. Therefore Q^{opt} is the Q factor of a 'QS decomposition' of A, decomposing a matrix A in the product of an orthogonal matrix Q and a symmetric matrix S. Figure 3.5 shows a picture, analogous to the polar coordinate interpretation of the QR decomposition. The main difference with the QR decomposition is that now the set $\mathcal{S}_Q$ of constant Q is orthogonal to the (tangent space of) the set $\mathcal{Q}_S$ of constant S. Actually, if S is positive definite this decomposition is also called polar decomposition [48],[56]. It is related to the SVD in the following sense. If $A = U\Sigma V^T$ is an SVD of A, a QS decomposition is given by $A = (UV^T)(V\Sigma V^T)$. The solution with positive singular values yields the polar decomposition, but choices with negative singular values also result in critical points of the optimization problem. One can see that $Q^{opt} = UV^T$ (resulting from a choice of positive singular values), minimizes $\|Q - A\|_F$, by observing $\|Q-A\|_F = \|I-S\|_F$ (figure 3.5) or even $\|Q-A\|_F = \|I-\Sigma\|_F$. This last transformation actually uses the coordinate transformation corresponding to the EVD (or SVD), and corresponds to the classical solution of the problem [48].

Of course, most problems are not that simple. However, the following *formal* account, based on the gradient of $\|Q-A\|_F^2$, is more susceptible to generalization. The derivation of the gradient, is based on formula (3.14) of definition 3.21. We have to give Q a velocity $\dot{Q}$ and write $\frac{d}{dt}\|Q - A\|_F^2$ as an inner product between

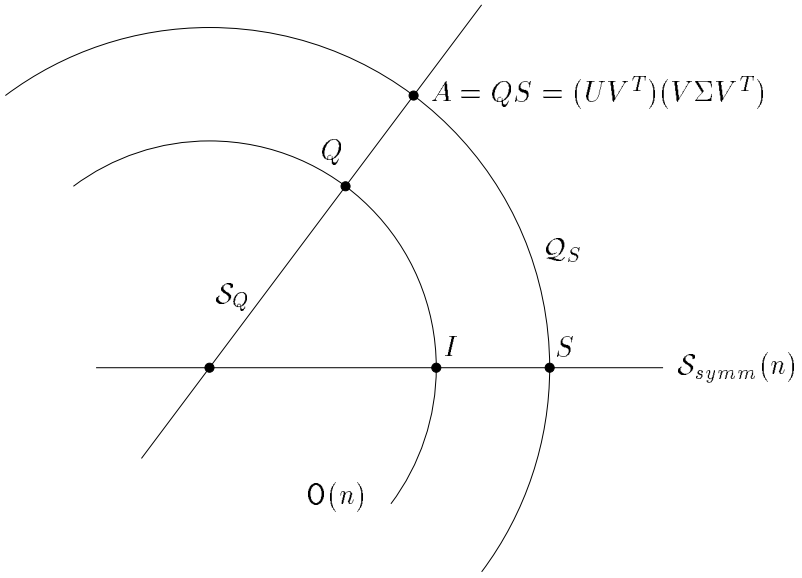


Figure 3.5: Geometric interpretation of problem 3.26 with a polar decomposition.

$\dot{Q}$ and another tangent vector. Setting $\dot{Q} = Q\Omega$, with $\Omega \in \mathcal{S}_{skew}(n)$ arbitrary, we find

$$\begin{aligned} \tfrac{d}{dt}\|Q-A\|_F^2 &= \tfrac{d}{dt}\langle Q-A, Q-A\rangle \\ &= 2\langle Q-A, \dot{Q}\rangle \\ &= 2\langle Q-A, Q\Omega\rangle \\ &= 2\langle I - Q^TA, \Omega\rangle \\ &= \langle A^TQ - Q^TA, \Omega\rangle \\ &= \langle Q(A^TQ-Q^TA), Q\Omega\rangle. \end{aligned}$$

In the one but last step we have taken the skew symmetric part $\frac{1}{2}(A^TQ - Q^TA)$ of $I - Q^TA$ (orthogonal projection onto $\mathcal{S}_{skew}(n)$). As a result

$$\mathrm{grad}_Q \|Q-A\|_F^2 = Q(A^TQ - Q^TA).$$

It is zero if Q^TA is symmetric, which corresponds to the results obtained above.

The corresponding gradient descent flow is given by

$$\dot{Q} = Q(Q^TA - A^TQ) = A - QA^TQ.$$

This can be regarded as a continuous-time algorithm for computing the polar decomposition. In section 4.2 we will show how the double bracket flow computes the EVD in a similar way. More general examples of this type are discussed in [9], [55], [17].

Example 2: distance measures between subspaces

In the examples below, illustrating statement 3.24(iii), we consider the gradient of functions defined in terms of matrix decompositions. We consider no constraint, although combination with a constraint as above is possible. (Actually, there is a constraint requiring that A (below) has full rank. However, as this constraint describes an nm-dimensional manifold in $\mathbb{R}^{n\times m}$, tangent vectors lie in $\mathbb{R}^{n\times m}$.)

We consider the following distance measures between a variable subspace $\mathrm{col}(A)$ and a fixed subspace $\mathrm{col}(M)$, where $A, M \in \mathbb{R}^{n\times m}$ have full rank.

$$\begin{aligned} \delta(A) &= \mathrm{Tr}(\,\mathrm{qf}(M)^T\,\mathrm{qf}(A)\,\mathrm{qf}(A)^T\,\mathrm{qf}(M)), \\ \theta(A) &= \mathrm{acos}(\bar{\sigma}(\,\mathrm{qf}(M)^T\,\mathrm{qf}(A))), \end{aligned}$$

where $\bar{\sigma}$ denotes the maximal singular value. The function $\theta(A)$ is the minimal angle between $\mathrm{col}(A)$ and $\mathrm{col}(M)$. The function $\delta(A)$ is the sum of the cosines squared of all canonical (or principal) angles [48].

To find δ or θ from A we need a QR decomposition (actually, any orthogonalization of the columns of A would do) and for θ also a singular value decomposition. Therefore, *to find the gradient we need the derivatives of these decompositions.* These are given in section 3.1.

Again A is given a velocity $\dot{A}$, and $\dot{\delta}$ is written as an inner product between $\dot{A}$ and another vector. Let $Q = \mathrm{qf}(A)$, $R = \mathrm{rf}(A)$ and $\bar{M} = \mathrm{qf}(M)$. With corollary 3.5, one finds

$$\dot{Q} = Q\Omega + PP^T\dot{A}R^{-1},$$

where $\Omega \in \mathcal{S}_{skew}(n)$, $P \in \mathrm{St}(n-m,n)$ and $P^TQ = 0$. (Ω is not further detailed as it will fall away - Recall that Ω corresponds to internal rotations in $\mathrm{col}(Q) = \mathrm{col}(A)$). For $\frac{d}{dt}\delta(A)$ this yields

$$\begin{aligned} \tfrac{d}{dt}\delta(A) &= \mathrm{Tr}\bar{M}^T\dot{Q}Q^T\bar{M} + \bar{M}^TQ\dot{Q}^T\bar{M} \\ &= 2\,\mathrm{Tr}\bar{M}^TPP^T\dot{A}R^{-1}Q^T\bar{M} \\ &= \langle \dot{A}, 2PP^T\bar{M}\bar{M}^TQR^{-T}\rangle. \end{aligned}$$

From the definition 3.21 of the gradient one finds

$$\mathrm{grad}_A\delta(A) = 2PP^T\bar{M}\bar{M}^TQR^{-T} = 2(I-QQ^T)\bar{M}\bar{M}^TQR^{-T}.$$

For the minimal angle one has to put some more steps using the derivative of a singular value. We only give the result. Similar ideas, although not explicitly directed towards obtaining the gradient, are worked out in more detail in section 7.4. One finds

$$\mathrm{grad}_A\theta(A) = -ba^TQR^{-T},$$

where a is the canonical (unit) vector in $\mathrm{col}(A)$ where the minimal angle with $\mathrm{col}(M)$ is reached, and b is the corresponding canonical (unit) vector in $\mathrm{col}(A)^\perp$ where the maximal angle with $\mathrm{col}(M)$ is reached.

3.4 The complex case

In this section we consider the case of complex matrices. In the major part of this work, we work with real matrices. One motivation is that the existing literature on continuous time matrix algorithms and certainly the literature on neural networks are almost completely in terms of real matrices. The main motivation is that directly giving the complex account entails quite a few extra details, spoiling the simplicity of the explanations. On the other hand, once one understands the real case, the results of this section make it a small step to the complex case. In chapter 11 we will consider an optimization problem over a manifold of complex matrices.

Below, we generalize some of the definitions of chapter 2, indicate the major points of difference for the lemmas of section 3.1 and finally discuss the general EVD with distinct eigenvalues.

Definition 3.27
The standard inner product of matrices $A, B \in \mathbb{C}^{n\times m}$ is defined by

$$\begin{aligned}\langle A, B\rangle &= \sum_{i=1}^{n}\sum_{j=1}^{m} \mathsf{Re}\,(a_{i,j})\mathsf{Re}\,(b_{i,j}) + \mathsf{Im}\,(a_{i,j})\mathsf{Im}\,(b_{i,j}),\\ &= \mathsf{Re}\ \mathsf{Tr}(A^H B) = \mathsf{Re}\ \mathsf{Tr}(B^H A),\\ &= \mathsf{Re}\ \mathsf{Tr}(A^H B) = \mathsf{Re}\ \mathsf{Tr}(B^H A).\end{aligned}$$

This inner product follows naturally by identifying $\mathbb{C}^{n\times n}$ with $\mathbb{R}^{2n^2}$.

The dominant role of the group of orthogonal matrices $\mathsf{O}(n)$ in the previous sections is taken over by the group of unitary matrices $\mathsf{U}(n)$ for the complex case. The role of the Lie algebra $\mathfrak{o}(n) = \mathfrak{so}(n)$ of skew symmetric matrices is played by the lie algebra $\mathfrak{u}(n)(= \mathfrak{su}(n))$ of skew Hermitian matrices. We also define one of the triangular complex groups, $\mathcal{C}_{upp}(n)^+$, and its Lie algebra.

Definition 3.28

$$\begin{aligned}\mathsf{U}(n) &= \{X \in \mathbb{C}^{n\times n} | X^H X = I\},\\ &\quad \text{orthogonal group},\\ \mathcal{C}_{upp}(n)^+ &= \{X \in \mathbb{C}^{n\times n} | x_{i,j} = 0 \text{ if } i > j, x_{i,i} \in \mathbb{R} \text{ and } x_{i,i} > 0, i, j = 1, \ldots, n\}.\end{aligned}$$

The corresponding Lie algebras are

Definition 3.29

$$\begin{aligned}\mathfrak{u}(n) &= \{X \in \mathbb{C}^{n\times n} | X^H = -X\},\\ \mathfrak{c}_{upp}(n)^+ &= \mathfrak{c}_{upp}(n) = \{X \in \mathbb{C}^{n\times n} | x_{i,j} = 0 \text{ if } i > j, x_{i,i} \in \mathbb{R}, i, j = 1, \ldots, n\}.\end{aligned}$$

The following proposition generalizes proposition 2.12. Also, the proofs are similar.

Proposition 3.30
(i) The tangent space $T_Q\mathsf{U}(n)$ to $\mathsf{U}(n)$ at a point Q is given by

$$T_Q\mathsf{U}(n) = \{Q\Omega | \Omega^H = -\Omega\} = \{\tilde{\Omega}Q | \tilde{\Omega}^H = -\tilde{\Omega}\}. \tag{3.16}$$

(ii) the normal space at Q is given by

$$(T_Q\mathsf{U}(n))^\perp = \{QS | S^H = S\} = \{\tilde{S}Q | \tilde{S}^H = \tilde{S}\}.$$

(iii) $\dim \mathsf{U}(n) = n^2$.

Generalizations of most linear operators of definition 2.9 are straightforward. The most useful generalization of upph for the complex case is given by

Definition 3.31

$$\mathsf{upph}: \quad \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n} : X \to Y, \text{ where } \begin{array}{ll} y_{i,j} = x_{i,j} & \text{if } i < j, \\ y_{i,j} = \frac{1}{2}\mathsf{Re}\, x_{i,j} & \text{if } i = j, \\ y_{i,j} = 0 & \text{if } i > j. \end{array}$$

Using the above generalizations and replacing transposes ^{T} by Hermitian transposes ^{H}, lemma 3.4 can be translated to the complex case, yielding the same formulas.

For the *symmetric* EVD $X = P\Lambda P^H$ one of the main differences with the real case, is that already in the case of distinct eigenvalues there is a (continuous) degree of freedom in the choice of each eigenvector. This yields arbitrary purely imaginary diagonal entries of Ω in $\dot{P} = P\Omega$ in lemma 3.9 for all cases (distinct or multiple eigenvalues). (Alternatively one can define P to belong to a homogeneous space obtained by 'dividing' $\mathsf{U}(n)$ through the subgroup of diagonal matrices with entries with modulus 1).

The following lemma considers the *general* (nonsymmetric) EVD of a matrix $X \in \mathbb{C}^{n\times n}$ with n distinct eigenvalues.

Lemma 3.32
Let $X(t) = P(t)\Lambda(t)P^{-1}(t) \in \mathbb{C}^{n\times n}$ be a continuously differentiable eigenvalue decomposition of a path $X(t) \in \mathbb{C}^{n\times n}$ with distinct eigenvalues. And let $V(t) = P^{-H}(t)$ (The columns of V are the *left* eigenvectors). Then

$$\begin{aligned} \dot{\Lambda} &= \mathsf{diag}(M), \\ \dot{P} &= PZ, \end{aligned}$$

where $M = P^{-1}\dot{X}P = V^H\dot{X}P$, and Z is given by

$$\begin{array}{rcl} z_{i,i} & = & \text{arbitrary}, \\ z_{i,j} & = & m_{i,j}/(\lambda_j - \lambda_i), \end{array} \quad i = 1, \ldots, n,\ j = 1, \ldots, n.$$

Proof:
The proof is analogous to the proof of lemma 3.9. Let $Z = P^{-1}\dot{P}$ then $\dot{P} = PZ$ and $\dot{V}^H = -Z\dot{V}$ by lemma 3.3 (complex version). It follows that

$$\begin{array}{rcl} \dot{X} & = & \frac{d}{dt}(P\Lambda V^H), \\ & = & \dot{P}\Lambda V^H + P\dot{\Lambda}V^H + P\Lambda\dot{V}^H, \\ & = & P([Z,\Lambda] + \dot{\Lambda})V^H. \end{array}$$

Solving for $T = [Z, \Lambda]$ and $\dot{\Lambda}$ one finds

$$\begin{array}{rcl} T & = & \text{offdiag}(M) = Z\Lambda - \Lambda Z, \\ \dot{\Lambda} & = & \text{diag}(M). \end{array}$$

The formula for $z_{i,j}$ follows from $t_{i,j} = z_{i,j}(\lambda_j - \lambda_i)$. The arbitrary diagonal entries $z_{i,i}$ reflect that the eigenvectors are undetermined up to a scalar multiple. □

Remark 3.33
For the case of multiple eigenvalues, the situation is much more complicated. In the (nongeneric) case that the geometric multiplicity equals the algebraic multiplicity an eigenvalue transformation of a submatrix of M as in lemma 3.9(vi) leads again to eigenvectors for which the analogue of the conditions of lemma 3.9(v) are satisfied. However in the (generic) case of lower geometric multiplicities (implying nondiagonal Jordan forms) eigenvalues may have infinite derivatives. See for instance [60, p.372]. This case will play a (small) role in chapter 11.

Finally observe that isospectral algorithms can be generalized for general (non-symmetric) matrices. They take the form

$$\dot{X} = [\tilde{K}, X],$$

as follows with $\dot{P} = \tilde{K}P$ and $\dot{\Lambda} = 0$. Such systems are also called *self-similar* [103] instead of isospectral, as the same form is also taken by systems on a manifold of matrices sharing a nondiagonal Jordan form.

3.5 Conclusion

We have considered the derivatives of some matrix decompositions, and presented them as tools for the design and analysis of continuous-time matrix algorithms. They link the evolution (velocity) of a factor of a matrix to the evolution

of the matrix itself. An interpretation of the QR decomposition, the symmetric EVD, (or the SVD and the polar decomposition, which received less attention) as coordinate transformations, results in pictures closely related to polar coordinates in the plane. Using these interpretations, the given formulas can be interpreted in terms of 'tangential' and 'radial' components. Next we have considered the central concept of gradients, and illustrated by some examples how they can be obtained. Finally, we have discussed generalizations to the case of complex matrices.

Chapter 4

The QR flow and the double bracket flow

In this chapter we consider two continuous-time algorithms for computing the EVD of a symmetric matrix. Section 4.1 discusses the QR flow and section 4.2 the double bracket flow.

The motivation to discuss these algorithms is threefold. First, they have played an important role in the development of the field of continuous-time matrix algorithms. Secondly, they illustrate how the theory and the geometric interpretations of the previous chapters can be used for the design and analysis of algorithms. The QR flow is derived using QR decomposition tracking lemma 3.4. Using the same lemma, we also derive a new square root version. The double bracket flow is derived using a derivation, analogous to the discussion of the orthogonal approximation problem of definition 3.26. Thirdly, both algorithms play a role in the remainder of this work.

For more about the QR flow we refer to [96, 38, 15, 82, 103, 55]. For more about the double bracket flow we refer to [11, 7, 12, 17, 55].

4.1 The QR flow

The QR flow is given by

Algorithm 4.1 (QR flow)

$$\dot{E} \quad = \quad [E, \rho_{\mathsf{skew}}(E)] \qquad E \in \mathbb{R}^{n \times n}.$$

To compute the eigenvalues of a matrix A the algorithm is started with $E(0) = A$ and converges to a diagonal matrix with the eigenvalues of A as diagonal entries. The eigenvectors can be obtained by algorithm 4.9. Recall from lemma 2.11(vi), that for symmetric matrices $\rho_{\text{skew}}(E) = E - 2\,\text{upph}(E)$. That is, application of ρ_{skew} only amounts to zeroing the diagonal and changing signs above the diagonal.

We will derive the QR flow by analogy with the discrete-time QR algorithm. This derivation 'by analogy' only includes an indirect proof of the continuous-time algorithm, via the connection with the discrete-time algorithm. A direct proof will be given at the end of section 4.2, discussing the double bracket flow.

The heart of the derivation is that whereas the discrete-time algorithm tracks $Q^{[k]} = qf(A^k)$ *to find the eigenvalues of* A*, the continuous-time algorithm is based on tracking* $Q(t) = qf(e^{At})$. The values of $Q(t)$ at integer times $t = k$ are equal to the discrete values $Q^{[k]}$ when the discrete algorithm is applied to e^A instead of A. It is also possible to keep track of $E(t) = Q(t)^T A Q(t)$ without explicitly keeping track of $Q(t)$. This yields the QR-flow. The diagonal entries of E converge to the eigenvalues of A. We will restrict our attention to symmetric matrices A.

The derivation of the QR flow is a simple application of lemma 3.4. The geometric interpretation of section 3.2 provides insight in the behavior of the continuous-time algorithm and the relation with the well known power algorithm for calculating the principal eigenvalue and corresponding eigenvector of a matrix. We will see that the QR flow like the power algorithm tracks the angular part of a linear system. To this end we start by a brief discussion of the discrete-time and continuous-time power algorithm in $\mathbb{R}^2$. This algorithm can be seen as a 'polar coordinate counterpart' of the QR flow. The relationship between the discrete-time power algorithm and the QR algorithm has been known for a long time. A clear account is found in [101].

4.1.1 The power algorithm

The power algorithm is a well known algorithm for calculating the principal eigenvalue and eigenvector of a matrix $X \in \mathbb{R}^{n \times n}$. Like for the QR algorithm, we will focus on symmetric matrices. For the polar coordinate interpretation we think in $\mathbb{R}^2$.

Algorithm 4.2 (Discrete-time power algorithm)

$$q^{[k+1]} = \frac{Aq^{[k]}}{\|Aq^{[k]}\|}.$$

Algorithm 4.3 (Continuous-time power algorithm)

$$\dot{q} = (I - qq^T)Aq.$$

Theorem 4.4
(i) The discrete-time power algorithm tracks the angular part $q^{[k]} = x^{[k]}/\|x^{[k]}\|$ of a vector $x^{[k]}$ obeying the linear system, $x^{[k+1]} = Ax^{[k]}$

$$q^{[k]} = \frac{A^k q^{[0]}}{\|A^k q^{[0]}\|}.$$

(ii) The continuous-time power algorithm tracks the angular part $q(t) = x(t)/\|x(t)\|$ of a vector x obeying the linear system, $\dot{x} = Ax$

$$q(t) = \frac{e^{At} q(0)}{\|e^{At} q(0)\|}.$$

(iii) Let p_1 be the principal eigenvector of A. Then if $p_1^T q^{[0]} \neq 0$, $q^{[k]}$ converges to p_1.

Proof:
See [48] for the discrete-time algorithm and [16] for the continuous-time algorithm.
□

Figure 4.1 shows the evolution of a linear system $\dot{x} = Ax$ for a symmetric matrix $A \in \mathcal{S}_{symm}(2)$ with eigenvectors p_1 with positive eigenvalue and p_2 with negative eigenvalue. The figure also shows the evolution of the angular part. The intuitive explanation behind the power algorithm is that for the linear system the component of x along the principal eigenvector p_1 grows faster than the other component(s), such that the angular part converges to that eigenvector (except when no component along p_1 was present in the initial condition). Finally note that (after convergence) the corresponding eigenvalue can be found as $q^T Aq$.

4.1.2 The QR algorithm

The discrete-time QR algorithm is given by

Algorithm 4.5 (Discrete-time QR algorithm I)

$$\begin{array}{rclrcl} E^{[k+1]} & = & \mathsf{rf}(E^{[k]})\,\mathsf{qf}(E^{[k]}), & E^{[0]} & = & A, \\ Q^{[k+1]} & = & Q^{[k]}\mathsf{qf}(E^{[k]}), & Q^{[0]} & = & I. \end{array}$$

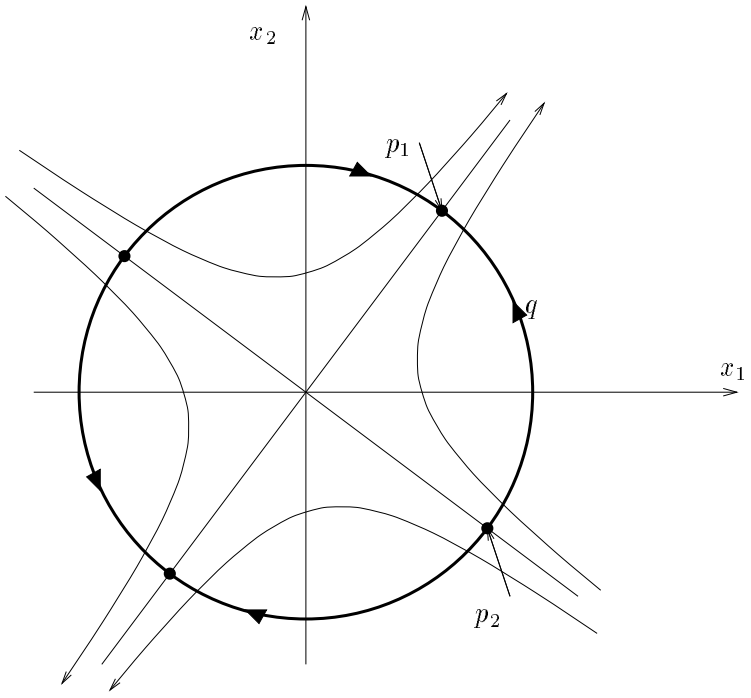


Figure 4.1: State space plot of a linear system $\dot{x} = Ax$, and its angular part, converging to the principal eigenvector p_1 of A.

An equivalent algorithm is

Algorithm 4.6 (Discrete-time QR algorithm II)

$$\begin{array}{rclrcl} E^{[k+1]} & = & \mathsf{qf}(E^{[k]})^T E^{[k]}\, \mathsf{qf}(E^{[k]}), & E^{[0]} & = & A, \\ Q^{[k+1]} & = & Q^{[k]}\, \mathsf{qf}(E^{[k]}), & Q^{[0]} & = & I. \end{array}$$

as follows directly from $\mathrm{rf}(E^{[k]}) = \mathrm{qf}(E^{[k]})^T E^{[k]}$. In this form the algorithm fits in the general scheme (2.10). It follows directly (by induction) that

$$E^{[k]} = {Q^{[k]}}^T A Q^{[k]}. \tag{4.1}$$

With this result, $Q^{[k+1]} = Q^{[k]}\,\mathrm{qf}(E^{[k]}) = Q^{[k]}\,\mathrm{qf}({Q^{[k]}}^T A Q^{[k]}) = \mathrm{qf}(AQ^{[k]})$. This leads to a third form of the algorithm, similar to the power algorithm, and also known as 'simultaneous iteration'.

Algorithm 4.7 (Discrete-time QR algorithm III)

$$\begin{array}{rclrcl} E^{[k+1]} & = & \mathsf{qf}(E^{[k]})^T E^{[k]}\, \mathsf{qf}(E^{[k]}), & E^{[0]} & = & A, \\ Q^{[k+1]} & = & \mathsf{qf}(AQ^{[k]}), & Q^{[0]} & = & I. \end{array}$$

Theorem 4.8
(i) $Q^{[k]}$ tracks the angular part $Q^{[k]} = \mathrm{qf}(X^{[k]})$ of a matrix $X^{[k]}$ obeying the linear system $X^{[k+1]} = AX^{[k]}$,

$$Q^{[k]} = \mathrm{qf}(X^{[k]}) = \mathrm{qf}(A^k X^{[0]}). \tag{4.2}$$

(ii) The columns of $Q^{[k]}$ converge to the eigenvectors of A, while the diagonal entries of $E^{[k]}$ converge to the corresponding eigenvalues.

Proof:
(i) Since $\mathrm{qf}(X^{[k+1]}) = \mathrm{qf}(AX^{[k]}) = \mathrm{qf}(A\,\mathrm{qf}(X^{[k]})\,\mathrm{rf}(X^{[k]})) = \mathrm{qf}(A\,\mathrm{qf}(X^{[k]}))$ the given solution (4.2) satisfies the third form of the QR algorithm (algorithm 4.7).

(ii) See [48] and references therein.
□

We are now in a position to derive the continuous-time QR algorithm. The matrix Q will track the angular part $Q = \mathrm{qf}(X)$ of a matrix X obeying the linear system $\dot{X} = AX$, while E keeps track of $Q^T AQ$. From lemma 3.4 this yields

$$\dot{Q} = Q\rho_{\mathrm{skew}}(Q^T AX\,\mathrm{rf}(X)^{-1}) = Q\rho_{\mathrm{skew}}(Q^T AQ) = Q\rho_{\mathrm{skew}}(E)$$

and

$$\dot{E} = \dot{Q}^T AQ + Q^T A\dot{Q} = -\rho_{\mathrm{skew}}(E)Q^T AQ + Q^T AQ\rho_{\mathrm{skew}}(E) = [E, \rho_{\mathrm{skew}}(E)].$$

Algorithm 4.9 (Continuous-time QR algorithm)

$$\begin{array}{rclcrcl} \dot{E} & = & [E, \rho_{\mathsf{skew}}(E)], & & E(0) & = & A, \\ \dot{Q} & = & Q\rho_{\mathsf{skew}}(E), & & Q(0) & = & I. \end{array}$$

The differential equation for E generates the QR flow (algorithm 4.1).

4.1.3 A square-root QR algorithm

Finally, we derive a new algorithm, the square-root QR algorithm. The derivation again illustrates the use of lemma 3.4 in the design of continuous-time algorithms.

Instead of keeping track of E as in the QR flow, the square-root algorithm keeps track of the Cholesky factor $C = \mathrm{cf}(E)$. With $\dot{E} = [E, \rho_{\mathrm{skew}}(E)]$ and $E = C^T C$, lemma 3.4 yields

$$\dot{C} = \mathrm{upph}(C\rho_{\mathrm{skew}}(C^T C)C^{-1})C - \mathrm{upph}(C^{-T}\rho_{\mathrm{skew}}(C^T C)C^T)C.$$

Because C and C^{-1} are upper triangular and C^T and C^{-T} are lower triangular, this formula can be simplified substituting $\rho_{\text{skew}} = \text{id} - 2\,\text{upph}$ in the first term and $\rho_{\text{skew}} = 2\,\text{lowh} - \text{id}$ in the second term (lemma 2.11(vi)). One finds

$$\begin{aligned} \dot{C} &= \text{upph}(CC^T)C - 2\,\text{upph}(C\,\text{upph}(C^TC)C^{-1})C \\ &\quad + \text{upph}(CC^T)C - 2\,\text{upph}(C^{-T}\,\text{lowh}(C^TC)C^T). \end{aligned}$$

Substituting $2\,\text{upph} = 2\,\text{upp} - \text{diag}$ and $\text{diag}(XY) = \text{diag}(X)\,\text{diag}(Y)$ for $X, Y \in \mathcal{S}_{upp}(n)$ or $X, Y \in \mathcal{S}_{low}(n)$ one finds

Algorithm 4.10 (Continuous-time square-root QR algorithm)

$$\dot{C} = 2\,\text{upph}(CC^T)C - 2C\,\text{upph}(C^TC).$$

4.2 The double bracket flow

The double bracket flow is another well known continuous-time algorithm that can be used to compute the EVD of a symmetric matrix, but also for other applications, like sorting numbers and solving linear programming problems [11].

It is given by

Algorithm 4.11 (Double bracket flow)

$$\dot{H} = [H,[H,N]], \qquad H, N \in \mathcal{S}_{symm}(n).$$

The matrix N parameterizes the system. For use as an EVD algorithm, N is chosen to be diagonal. But for further use, we consider the more general case with $N \in \mathcal{S}_{symm}(n)$. In that case *$H$ converges to a matrix H^{eq} with the eigenvalues of $H(0) = H_0$ and eigenvectors of N*. Theorem 4.12 gives a more precise statement. Below we give a simple derivation, by analogy with the discussion of the orthogonal approximation problem (definition 3.26).

Consider again the picture of the EVD as a coordinate transformation, reproduced in figure 4.2, and think of the circle as the manifold $\mathcal{I}_{H_0}$ of matrices with the same spectrum as H_0, and of the plane as the space $\mathcal{C}_N$ of matrices commutable with N (or rather as $\mathcal{C}_{H^{eq}}$ but this only makes a difference in the case of repeated eigenvalues). Because the plane is orthogonal to the (tangent space of the) circle (lemma 2.19), *finding a matrix H^{eq} with the eigenvalues of H_0 and eigenvectors of N, can be regarded as the following optimization problem.*

$$\min_{H \in \mathcal{I}_{H_0}} \|H - N\|_F^2.$$

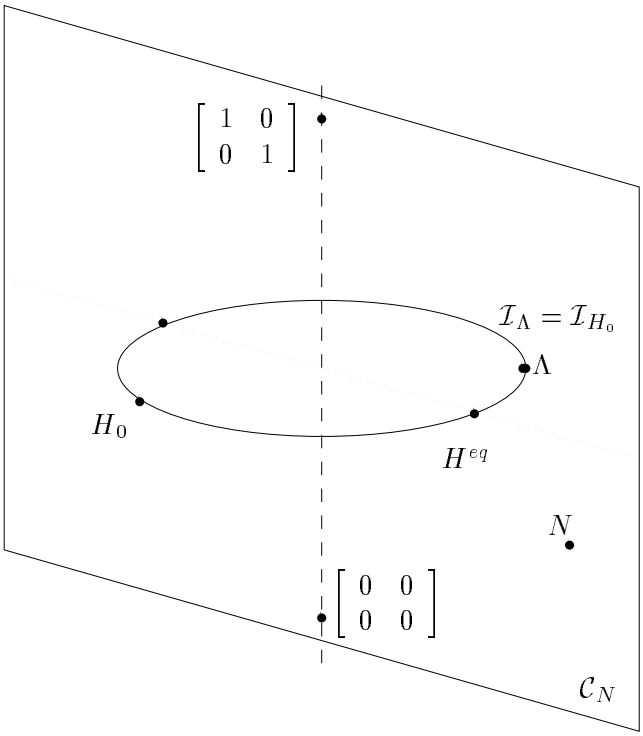


Figure 4.2: Picture of $\mathcal{S}_{symm}(2)$ with the cylindrical coordinate interpretation of the EVD (see figure 3.4). The matrix H^{eq} with the eigenvalues of H_0 and closest to N takes the eigenvectors of N.

That is, *find the point on the circle closest to* N. Because $\|H\|^2 = \mathrm{Tr}(H^T H)$ is constant on $\mathcal{I}_{H_0}$, $\|H - N\|_F^2 = \langle H - N, H - N \rangle$ is equal up to a constant to $-2\langle H, N \rangle$. Therefore the problem can be reformulated as

$$\max_{H \in \mathcal{I}_{H_0}} \langle H, N \rangle .$$

Rather than defining a gradient flow on $\mathcal{I}_{H_0}$, it is convenient to regard $\mathcal{I}_{H_0}$ as the image of $\mathsf{O}(n)$ under ϕ_{H_0} and consider a gradient flow on $\mathsf{O}(n)$ solving the following problem

$$\max_{Q \in \mathsf{O}(n)} \langle Q H_0 Q^T, N \rangle .$$

Actually the resulting algorithm on $\mathcal{I}_{H_0}$ (this will be the double bracket flow) can be considered as a gradient flow on $\mathcal{I}_{H_0}$ in an appropriate metric.

The gradient of $\langle Q H_0 Q^T, N \rangle$ is found by giving Q a velocity $\dot{Q} = \tilde{\Omega} Q$ and writing

$\frac{d}{dt}\langle QH_0Q^T, N\rangle$ as the inner product of $\dot{Q}$ and another tangent vector.

$$\begin{aligned}\frac{d}{dt}\langle QH_0Q^T, N\rangle &= \langle \dot{Q}H_0Q^T + QH_0\dot{Q}^T, N\rangle \\ &= \langle [\tilde{\Omega}, QH_0Q^T], N\rangle \\ &= \langle [N, QH_0Q^T], \tilde{\Omega}\rangle,\end{aligned}$$

where the last identity is an application of lemma 2.11(vii). By definition 3.21 of the gradient and proposition 2.12(iv) for the inner product of tangent vectors to $\mathsf{O}(n)$, the gradient is given by

$$\mathrm{grad}_{Q\in\mathsf{O}(n)}\langle QH_0Q^T, N\rangle = [N, QH_0Q^T]Q.$$

The corresponding gradient flow on $\mathsf{O}(n)$ is

$$\dot{Q} = [N, QH_0Q^T]Q. \tag{4.3}$$

The image under ϕ_{H_0}, denoting $H = \phi_{H_0}(Q)$, is $\dot{H} = [[N, H], H]$ or more classically,

$$\dot{H} = [H, [H, N]],$$

which is the double bracket flow.

One can see on figure 3.4 that the optimization problem, has more than one critical point (in $\mathbb{R}^{2\times 2}$). Therefore the double bracket flow has more than one equilibrium. They correspond to different orderings of the eigenvalues of H^{eq}.

However, in a *stable* equilibrium H^{st}, the k-th largest eigenvalue of H_0 ($k = 1, \ldots, n$) is attached to an eigenvector of H^{st} that is also an eigenvector of N with the k-th largest eigenvalue of N. This stable equilibrium is unique if N has distinct eigenvalues.

This behavior of the double bracket flow is summarized more accurately by the following three properties.

Theorem 4.12
(i) The double bracket flow is isospectral: $H(t)$ preserves its eigenvalues during the evolution of the system and has a differentiable EVD $H(t) = P(t)\Lambda P(t)^T$ where

$$\dot{P} = -[H, N]P = NP\Lambda - P\Lambda P^T NP.$$

(ii) If N has distinct eigenvalues, H converges to an isolated equilirium H^{eq} satisfying $[H^{eq}, N] = 0$. If N has multiple eigenvalues, it converges to a connected component of $\{H | H \in \mathcal{I}_{H_0}[H, N] = 0\}$.

(iii) Let H^{st} be a stable equilibrium of the double bracket flow. Let $\hat{P} \in \mathsf{O}(n)$ denote a matrix whose columns $\hat{p}_i = \hat{P}_{\cdot,i}$ are eigenvectors of H^{st} with eigenvalues λ_i and at the same time eigenvectors of N with eigenvalues μ_i, then

$$\begin{array}{lll}\lambda_i \geqslant \lambda_j & \text{if} & \mu_i > \mu_j, \\ \lambda_i \leqslant \lambda_j & \text{if} & \mu_i < \mu_j,\end{array}, \; i = 1, \ldots, n, \; j = 1, \ldots, n.$$

Proof:
(i) Since $H, N \in \mathcal{S}_{symm}(n)$, $[H,N] \in \mathfrak{so}(n) = \mathcal{S}_{skew}(n)$ and therefore $\dot{P} \in T_P\mathsf{O}(n)$ and P preserves its orthogonality. Assuming $H(t) = P(t)\Lambda P(t)^T$ one obtains, $\dot{H} = \dot{P}\Lambda P^T + P\Lambda\dot{P}^T = [H,[H,N]]$. The result follows from the uniqueness of H.
(ii) This is a direct consequence of the derivation above. Consider the inner product $\langle H,N\rangle$ as a Lyapunov function for the system (steadily increasing instead of decreasing). For the evolution of $\langle H,N\rangle$ we find

$$\frac{d}{dt}\langle H,N\rangle = \langle \dot{H},N\rangle = \langle [H,[H,N]],N\rangle = \langle [H,N],[H,N]\rangle,$$

where the last identity follows from lemma 2.11(vii). Consequently, $\frac{d}{dt}\langle H,N\rangle = \langle [H,N],[H,N]\rangle$ is always nonnegative and equals 0 only if $[H,N]=0$. Because $\mathcal{I}_\Lambda = \mathcal{I}_{H_0}$ is a compact manifold, this implies that H converges to a connected component of $\{H | H \in \mathcal{I}_\Lambda, [H,N] = 0\}$. If N has distinct eigenvalues, this implies convergence to an isolated equilibrium.

(iii) The existence of the matrix $\hat{P}$ with columns that are eigenvectors of both H^{eq} and N follows from (ii). To study the stability, we first transform the flow under the orthogonal transformation $s_{\hat{P}}$ (Recall that this transformation is a rotation around the symmetry axis in figure 3.4, and that it maps H^{st} to Λ and $T_{H^{st}}\mathcal{I}_\Lambda$ to $T_\Lambda\mathcal{I}_\Lambda$ - remark 3.16). Let $\hat{H} = s_{\hat{P}}(H) = \hat{P}^T H \hat{P}$, $\hat{H}^{st} = s_{\hat{P}}(H^{st}) = \hat{P}^T H^{st}\hat{P} \in \mathcal{S}_{diag}(n)$ and $\hat{N} = s_{\hat{P}}(N) = \hat{P}^T N\hat{P} \in \mathcal{S}_{diag}(n)$. Then,

$$\dot{\hat{H}} = [\hat{H},[\hat{H},\hat{N}]].$$

A perturbation of $\hat{H}$ with $\epsilon\Delta$ yields a right hand side $[\hat{H}+\epsilon\Delta,[\hat{H}+\epsilon\Delta,\hat{N}]] = [\hat{H},[\hat{H},\hat{N}]] + \epsilon[\hat{H},[\Delta,\hat{N}]] + \epsilon[\Delta,[\hat{H},\hat{N}]] + o(\epsilon)$. Because $[\hat{H}^{eq},\hat{N}]=0$, this results in the following linearized system at $\hat{H}^{eq}$

$$\dot{\Delta} = [\hat{H}^{eq},[\Delta,\hat{N}]].$$

Because $\hat{H}^{eq}$ and $\hat{N}$ are diagonal, with lemma 2.11(ii) one finds

$$\dot{\delta}_{i,j} = (\hat{h}_i^{st} - \hat{h}_j^{st})(\hat{n}_j - \hat{n}_i)\delta_{i,j}.$$

The equilibrium can be stable only if

$$(\hat{h}_i^{st} - \hat{h}_j^{st})(\hat{n}_i - \hat{n}_j) \geqslant 0.$$

That is, if the entries of $\hat{H}^{eq}$ are ordered in the same way as the entries of $\hat{N}$. This proves the lemma.
□

Remark 4.13
Note that if N has distinct eigenvalues, the stable equilibrium is unique, since its eigenvectors and the corresponding eigenvalues are fixed. However, if N has repeated eigenvalues, any rotation of two eigenvectors of H, having the same eigenvalue for N but a different eigenvalue for H, results in a new equilibrium.

To use the double bracket flow as an algorithm for computing eigenvalues, N can be chosen as $N = \mathrm{diag}\{1, \ldots, n\}$. The equilibrium H^{eq}, commuting with N, is diagonal and shows the eigenvalues of H_0 on its diagonal. The eigenvectors of H_0 can be obtained by

$$\dot{P} = P[H, N], \qquad P(0) = I,$$

which follows from 4.3 by setting $P = Q^T$.

Figure 4.3 shows the double bracket flow in $\mathcal{S}_{symm}(2)$ in the cylindrical coordinate interpretation of figure 3.4. The space of diagonal matrices $\mathcal{S}_{diag}(2)$ is one of the spaces with constant eigenvectors. The matrix H is attracted towards the diagonal matrix $N \in \mathcal{S}_{symm}(2)$, while staying on the isospectral circle. If $N = \mathrm{diag}\{n_1, n_2\}$ is on the right hand side of the symmetry axis, $n_1 > n_2$ and H converges to $\mathrm{diag}\{\lambda_1, \lambda_2\}$, with $\lambda_1 > \lambda_2$. If N is on the left hand side of the symmetry axis, H converges to $\mathrm{diag}\{\lambda_2, \lambda_1\}$.

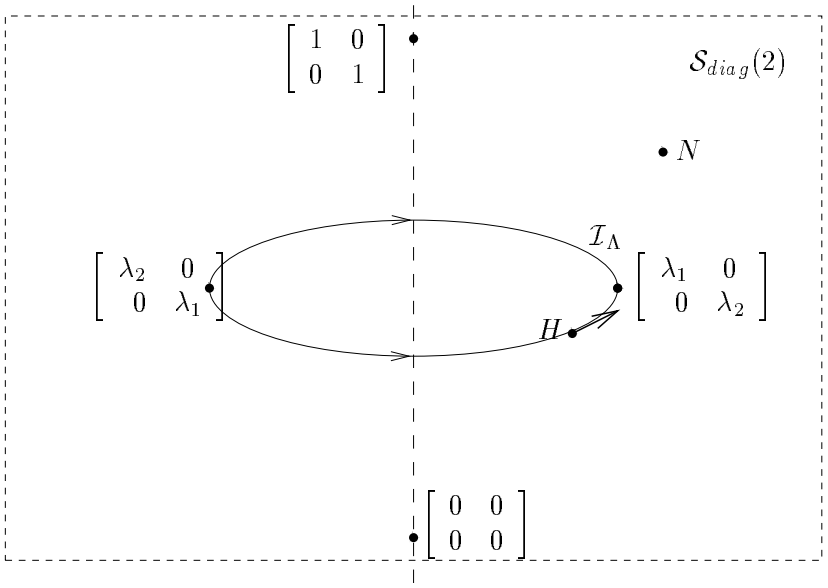


Figure 4.3: The double bracket flow for eigenvalue computation in $\mathcal{S}_{symm}(2)$, in the cylindrical coordinate interpretation of figure 3.4

Finally, we prove that also the QR flow converges using the same Lyapunov function as for the double bracket flow with $N = \mathrm{diag}\{1, \ldots, n\}$.

Theorem 4.14
Let E satisfy the QR flow (algorithm 4.1), then E converges to a diagonal matrix showing the eigenvalues of $E_0 = E(0)$ on the diagonal.

Proof:
Let $N = \text{diag}\{1, \ldots, n\}$. We can give almost the same proof as for theorem 4.12(ii), because the entries of $\rho_{\text{skew}}(E)$ have the same sign as the entries of $[E, N]$.

$$\frac{d}{dt}\langle E, N\rangle = \langle \dot{E}, N\rangle = \langle [E, \rho_{\text{skew}}(E)], N\rangle = \langle [E, N], \rho_{\text{skew}}(E)\rangle.$$

Now $[E, N]_{i,j} = e_{i,j}(n_{j,j} - n_{i,i}) = e_{i,j}(j - i)$, (lemma 2.11(ii)) and $\rho_{\text{skew}}(E)_{i,j} = e_{i,j}$ if $i < j$ and $\rho_{\text{skew}}(E)_{i,j} = -e_{i,j}$ if $i > j$. One finds that

$$\frac{d}{dt}\langle E, N\rangle \geqslant 0$$

and is only equal to 0 if $[E, N] = \rho_{\text{skew}}(E) = 0$. Because $\mathcal{I}_{E_0}$ is compact, E converges to a matrix E^{eq} that commutes with N (there are no unisolated equilibria). Therefore, E is diagonal and shows the eigenvalues of E_0 on the diagonal.
□

4.3 Conclusion

We have discussed the QR flow and the double bracket flow for computation of a symmetric EVD, and derived a square-root version of the QR flow. These examples illustrate the use of the lemmas of section 3.1 and gradients in the design of continuous-time matrix algorithms. Using the geometric interpretations of the previous chapter, we have found some pictures synthesizing the main properties of these algorithms.

Chapter 5

A class of uniform parallel continuous-time matrix algorithms

In this short chapter we introduce a class of continuous-time algorithms, that are linked with the main themes of this work in the following way.

1. They are (nonautonomous) continuous-time matrix algorithms, that can be analyzed and manipulated using the mathematical theory of chapters 2 and 3.

2. Some are continuous-time limits of systolic algorithms. (Chapters 6, 7 and 8).

3. They are generalizations of Hebbian adaptation laws for neural networks. (Chapters 9 and 10).

We will sometimes refer to these algorithms as continuous-time 'systolic' algorithms, although the original notion of systolic, referring to the way in which information travels through the array, is no longer appropriate in the continuous-time limit. The main characteristic of the considered signal flow graphs is that they consist of simple, (mostly) identical cells.

5.1 Definition

We consider a class of algorithms, whose right hand side consists of three types of terms.

Definition 5.1 (A class of uniform parallel continuous-time matrix algorithms) Consider a class of continuous-time algorithms of the form

$$\dot{A} = \sum_{i=1}^{N} T_i(A, x, y),$$

with state $A(t) \in \mathbb{R}^{n \times m}$ and possible inputs $x(t) \in \mathbb{R}^n$, $y(t) \in \mathbb{R}^m$ and where the *terms* $T_i(A, x, y)$ can be of three types, defined below.

Let $c, e \in \mathbb{R}^n$ be vectors of the form $(AA^T)^k x$ or $(AA^T)^k Ay$ for some integer $k \geqslant 0$. Let $d, f \in \mathbb{R}^m$ be vectors of the form $(A^T A)^k y$ or $(A^T A)^k A^T x$ for some integer $k \geqslant 0$.

Then, terms of type I are of the form

$$T = cd^T.$$

Terms of type II are of the form

$$T = \mathrm{tri}(ce^T)A \text{ or } T = A\,\mathrm{tri}(df^T),$$

where tri stands for upp, upph, upps, low, lowh or lows as defined in definition 2.9. Terms of type III are of the form

$$T = \mathrm{tri}(cd^T A^{-1})A \text{ or } T = A\,\mathrm{tri}(A^{-1}cd^T),$$

where either $A \in \mathcal{S}_{upp}(n)$ and tri stands for upp, upph or upps, or $A \in \mathcal{S}_{low}(n)$ and tri stands for low, lowh or lows.

5.2 Signal flow graphs

The signal flow graph of these systems is a rectangular array of cells. Cell (i, j) on row i and column j keeps track of the entry $a_{i,j}$ of A. Vectors are realized as signals in the rows or the columns of the array.

The systems are called *uniform* parallel because all cells in the signal flow graph have the same functionality. However, we will not be very strict about this requirement and also allow signal flow graphs with a limited number of different

cell types. For terms of type III for instance, there will be a difference between diagonal and nondiagonal cells.

The central idea behind the parallel realization is that by calculating signals like c, d, e, f, $A^T c$, Ad, $A^{-T}d$ *or* $A^{-1}c$ *from* A *and* x *or* y *by a straightforward implementation of matrix vector multiplications (or back substitutions for* $A^{-T}d$ *and* $A^{-1}c$*), each cell* (i,j) *possesses enough information to calculate* $\dot{a}_{i,j}$.

To this end each component x_i, c_i or e_i, $i = 1, \ldots, n$, of a vector x, c or e is realized as a signal, available to all cells (i,k) in row i of the array. The vectors x, c and e, are said to be *available in the rows*. Similarly, each component y_j, d_j or f_j, $j = 1, \ldots, m$, of a vector y, d or f is realized as a signal, available to all cells (k,j) in column j of the array. The vectors y, d and f are said to be *available in the columns*.

The inputs x or y are supplied externally (Most examples will have either an input x or an input y). The vectors c, d, e, f are obtained by matrix vector multiplication in a straightforward way. As an example, figure 5.1 shows the realization of $d = A^T c$. If needed (e.g. for the calculation of Ad) the components d_j obtained at the bottom cells in figure 5.1 are supplied to the other cells in the column. (The sums can of course also be accumulated from bottom to top. See for instance figure 5.2).

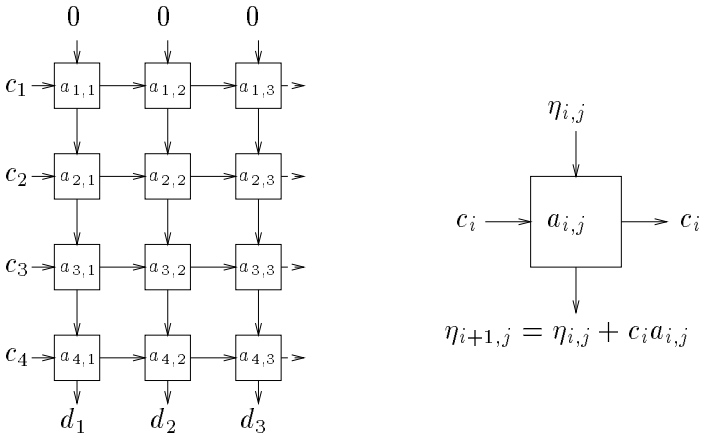


Figure 5.1: Signal flow graph for the calculation of $d = A^T c$. Left: global picture and boundary conditions. Right: cell (i,j).

For a term of type I, $T_{i,j} = c_i d_j$, both c_i and d_j are available at cell (i,j), through the realization of the matrix vector products, described above. As an example figure 5.2 shows the realization of $\dot{A} = xx^T A$

For terms of type II, consider the example $T = A\,\mathrm{upph}(df^T)$. Its signal flow graph is given in figure 5.3. For this term, cell (i,j) has to calculate $T_{i,j} =$

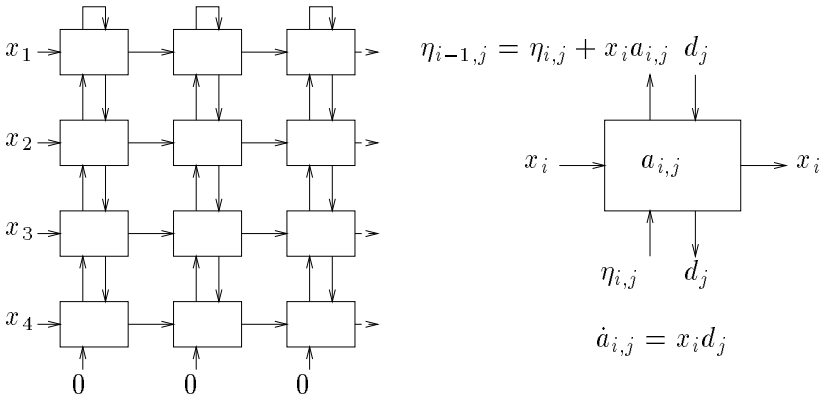


Figure 5.2: Signal flow graph for $\dot{A} = xx^T A$ of type I. Left: global picture and boundary conditions. Right: cell (i, j).

$\frac{1}{2}(\zeta_{i,j} + \zeta_{i,j+1}) f_j$, where $\zeta_{i,j} = \sum_{k=1}^{j-1} a_{i,k} d_k$. This only differs from the corresponding component $(Ad)_i f_j$ of the term Adf^T of type I, in that the partial sums $\zeta_{i,j}$ and $\zeta_{i,j+1}$ are needed instead of the whole sum $(Ad)_i = \zeta_{i,m+1}$. These partial sums are exactly the ones that are available at cell (i, j) by the computation of Ad. Therefore, although the presence of upph makes the formula for T more complex, its signal flow graph is actually simpler. With upps instead of upph, only $\zeta_{i,j}$ is needed. With upp, only $\zeta_{i,j+1}$ is needed. With a lower triangular part low, lows or lowh, partial sums $\hat{\zeta}_{i,j} = \sum_{k=j+1}^{n} a_{i,k} d_k$ and/or $\hat{\zeta}_{i-1,j}$ are needed. These are obtained by starting to accumulate the total sum at the other side of the array.

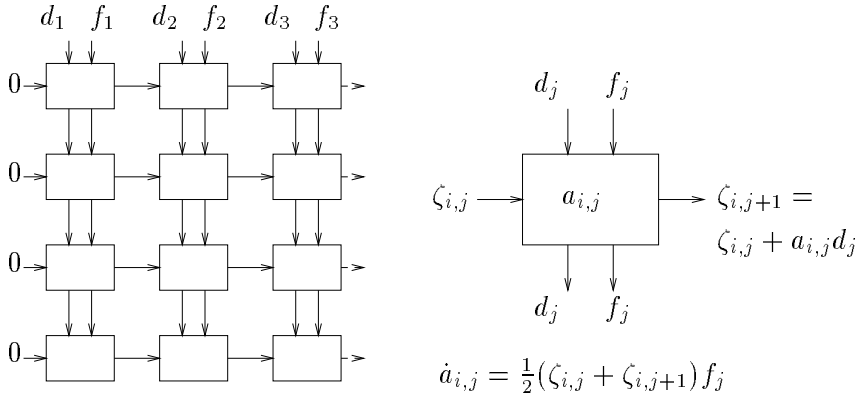


Figure 5.3: Signal flow graph for $\dot{A} = A\,\mathrm{upph}(df^T)$ of type II. Left: global picture and boundary conditions. Right: cell (i, j).

For terms of type III, consider the example $\dot{A} = \mathrm{upph}(cd^T A^{-1})A$, with $A \in \mathcal{S}_{upp}(n)$. Its signal flow graph is given in figure 5.5. For this term we need to calculate $b = A^{-T}d$. Since A is triangular, b can be obtained by a parallel realization of the back substitution scheme

$$b_j = (d_j - \sum_{k=1}^{j-1} b_k a_{k,j})/a_{j,j}, \tag{5.1}$$

where $d_j - \sum_{k=1}^{j-1} b_k a_{k,j}$ is computed through the partial sums

$$\eta_{i,j} = d_j - \sum_{k=1}^{i-1} b_k a_{k,j}. \tag{5.2}$$

The corresponding signal flow graph of figure 5.4 can be interpreted as the reverse of the signal flow graph for obtaining d from b by matrix vector multiplication (as in figure 5.1, but with triangular A).

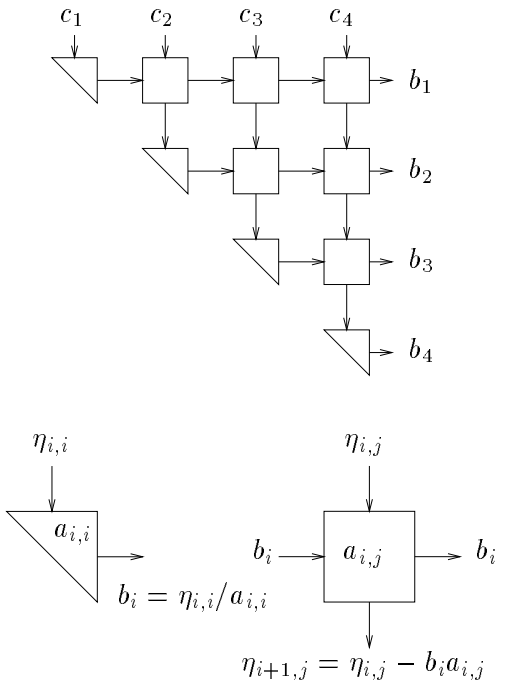


Figure 5.4: Signal flow graph for the calculation of $b = A^{-T}d$. Top: global picture and boundary conditions. Bottom: cells (i,i) and (i,j), $j > i$.

For the calculation of $T_{i,j} = \dot{a}_{i,j} = \frac{1}{2}c_i(\eta_{i,j} + \eta_{i+1,j})$, the signals $\eta_{i,j}$ and $\eta_{i+1,j}$ are exactly the ones that are available at cell (i,j) for the computation of $A^{-T}d$.

If upp or upps is used, only one of the signals $\eta_{i,j}$ or $\eta_{i+1,j}$ is needed. Other terms of type III are obtained in a similar way.

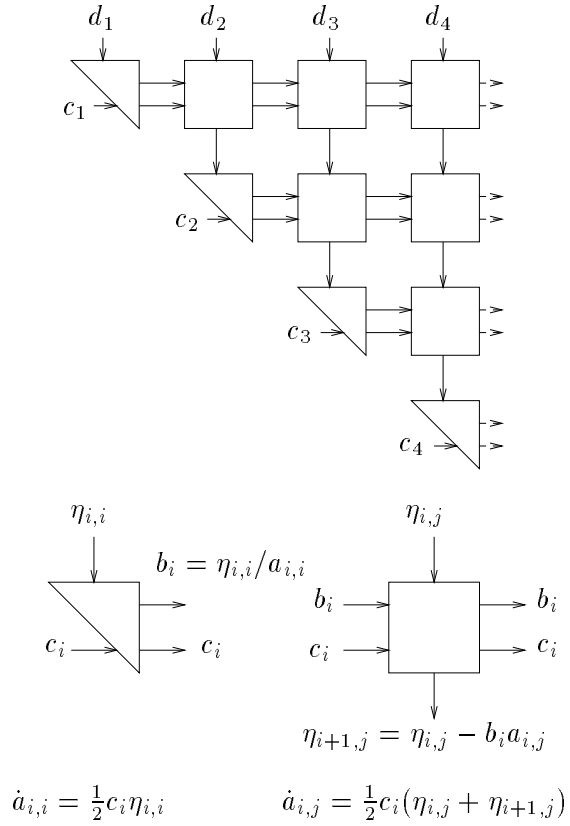


Figure 5.5: Signal flow graph for $\dot{A} = \mathrm{upph}(cd^T A^{-1})A$ of type III. Top: global picture. Bottom: cells (i,i) and (i,j), $j > i$.

Remark 5.2
The systems defined above (definition 5.1) are of course not the only ones that merit the label 'uniform parallel realization'. In some further examples we will allow small extensions. A term $T = \lambda A$ where $\lambda \in \mathbb{R}$, for example, has a trivial uniform parallel realization. And one could think of many others. However the class defined above was chosen because of its links, cited above, with the main themes of this work.

Chapter 6

Continuous-time recursive least-squares estimation

In this chapter, we derive three algorithms for continuous-time recursive least-squares estimation (RLS). Algorithm 6.10 and algorithm 6.23 belong to the class of algorithms with a uniform parallel realization, introduced in chapter 5, algorithm 6.19 and its variants assume a small extension of this class. The three algorithms are derived using lemma 3.4, for Cholesky factor tracking, LU tracking and inverse Cholesky factor tracking.

The parallel realization of the three algorithms can be interpreted as a neural network (chapter 9), but also as the continuous-time limit of systolic arrays. Algorithm 6.10 is a continuous-time version of the well known Gentleman-Kung (GK) systolic algorithm and will be referred to as the continuous-time GK algorithm. Algorithm 6.23 is a continuous-time version of the Moonen McWhirter systolic algorithm (MM), and will be referred to as the continuous-time MM algorithm. Algorithm 6.19 is a continuous-time version of a so called 'square root free' variant of the GK systolic algorithm, based on [46].

For researchers in neural networks, these continuous-time algorithms are illustrations of local adaptation laws, with a solid mathematical background, and whose global behavior solves a classical adaptive signal processing problem. Neural interpretations will be discussed in chapter 9. Researchers in systolic algorithms may be surprised by the simple neural form which a systolic algorithm takes in the continuous-time limit, and by the simple way in which they can be derived and manipulated. The link between continuous-time algorithms and systolic arrays for RLS, will be of importance for the derivation of a new systolic algorithm in chapter 8.

Finally, because of their simple analog signal flow graph algorithms 6.10 and 6.19 (or its simplified versions 6.20 and 6.21) are candidates for analog realization. We discuss 'high level' aspects of this topics, but a detailed investigation falls outside the scope of this work.

This chapter is organized as follows. Section 6.1 discusses the RLS estimation. Section 6.2 considers the solution by Cholesky factor tracking, and the resulting continuous-time GK algorithm. Section 6.3 discusses the link with the discrete-time GK systolic algorithm. Section 6.4 considers RLS estimation based on the LU decomposition, and some variations that are relevant for analog realization. In section 6.5 we consider the continuous-time MM algorithm, which will also play a role in chapter 8. Finally, in section 6.6 we consider aspects of analog realization. The main results of this chapter were published in [36, 26].

6.1 RLS estimation

In this section, we describe the continuous-time linear estimation problem and show how, like in the discrete-time case, the estimation problem can be replaced by a continuous-time RLS optimization problem.

Linear estimation is one of the most classical problems in adaptive signal processing. In continuous time it can be formulated as follows

Definition 6.1 (Linear estimation problem)
Let $y(t) \in \mathbb{R}^n$ be a stochastic signal generated by

$$y(t) = \bar{w}^T x(t) + \bar{e}(t). \tag{6.1}$$

The signals $x(t) \in \mathbb{R}^n$ are known *inputs*.
The signal $y(t) \in \mathbb{R}$ is a known *reference input*.
The signal $\bar{e}(t) \in \mathbb{R}$ is an unknown signal, which is uncorrelated with the inputs $x_1, \ldots, x_n$, and is called the *residual*.
The signal $\bar{s}(t) = \bar{w}^T x(t) = y(t) - \bar{e}(t)$ is called the *linear approximation*.
The vector $\bar{w} \in \mathbb{R}^n$ is the unknown *weight vector*.

For many applications $\bar{w}$ is also time varying, but 'on a slower time scale' than x (since to estimate $\bar{w}$ information of x and y over a long enough time interval is needed).

This problem and its discrete-time counterpart have a broad spectrum of well known applications, such as adaptive noise cancellation, channel equalization and beamforming. We refer to [50]. In some applications, one wants to estimate the linear approximation $\bar{s}(t)$, in others the residual $\bar{e}(t)$ and in yet others the weights $\bar{w}$.

The recursive least-squares (RLS) approach gives estimates w or e or s by tracking the solution of a time dependent least-squares optimization problem. This approach is mostly not stated in a stochastic framework, but rather involves the use of time averages. To this end, we introduce the following exponentially weighted inner product and the corresponding norm for functions (signals) on an interval $(-\infty, t]$.

Definition 6.2
Let $L_{2,\lambda}(-\infty, t]$ denote the set of functions f over $(-\infty, t]$ for which $\int_{-\infty}^{t} f(\tau)^2 e^{-\lambda(t-\tau)} d\tau$ is finite.
Let $f, g \in L_{2,\lambda}(-\infty, t]$, then

$$\begin{aligned} \langle f, g \rangle_{(t)} &= \int_{-\infty}^{t} f(\tau) g(\tau) e^{-\lambda(t-\tau)} d\tau, \\ \|f\|_{(t)} &= \sqrt{\langle f, f \rangle_{(t)}}. \end{aligned}$$

Let $x \in (L_{2,\lambda}(-\infty, t])^n$, $y \in (L_{2,\lambda}(-\infty, t])^m$ be column vectors of signals, then

$$\langle x, y^T \rangle_{(t)} = \int_0^t e^{-\lambda(t-\tau)} x(\tau) y(\tau)^T d\tau. \tag{6.2}$$

The scalar $\lambda \geqslant 0$ is called *forgetting rate*.

If the inner product is regarded as a generalization of an inner product in a finite-dimensional vector space, (6.2) generalizes a matrix product. One can think of the column vector $x \in (L_{2,\lambda}(-\infty, t])^n$ as a matrix with a continuous column index.

The continuous-time RLS problem can now be stated as the following *time dependent optimization problem*

Definition 6.3 (continuous-time RLS problem)
Let $x = [x_1, \ldots, x_n]^T \in (L_{2,\lambda}(-\infty, t])^n$ be a vector of given *input signals*.
Let $y \in L_{2,\lambda}(-\infty, t]$ be a *reference input signal*. Then the *optimal RLS weights* are defined by

$$\begin{aligned} w^{opt}(t) &= \arg\min_{w(t) \in \mathbb{R}^n} \|y - w(t)^T x\|_{(t)} \\ &= \arg\min_{w(t)} \int_{-\infty}^{t} (y(\tau) - w(t)^T x(\tau))^2 e^{-\lambda(t-\tau)} d\tau. \end{aligned}$$

The *RLS approximation* at time t is defined by

$$s(t) = w^{opt}(t)^T x(t) = \sum_{i=1}^{n} w_i^{opt}(t) x_i(t) \tag{6.3}$$

and the *RLS residual* at time t is defined by

$$e(t) = y(t) - s(t). \tag{6.4}$$

We will also denote

$$\begin{array}{rcl} s(t) & = & [y \parallel \{x_1, \ldots, x_n\}](t), \\ e(t) & = & [y \perp \{x_1, \ldots, x_n\}](t). \end{array} \tag{6.5}$$

Remark 6.4
The label 'recursive' is borrowed from discrete-time terminology, where it refers to the fact that one wants to keep track of the solution to the time dependent optimization problem, by recursively updating information (the R-factor of the data matrix) from which the solution can be read easily. In continuous time this will be translated into a differential equation keeping track of this information.

At any time instant t the signals $x_1, \ldots, x_n$ are considered to be functions on the interval $(-\infty, t]$. And $w(t)^T x$ is the best approximation of the reference signal y as a linear combination of the signals x_i, $i = 1, \ldots, n$, in terms of the norm of definition 6.2. The forgetting rate $\lambda \geqslant 0$ determines the relative importance of past information.

The notations $\perp$ and $\parallel$ reflect the fact that the best approximation to y as a linear combination of x_1,x_2,...,x_n is the projection of y onto the space spanned by x_1,x_2,...,x_n, whereas the minimal difference between y and its best approximation is the component of y, orthogonal to that space. Note however that whereas the best approximating function, $w^{opt}(t)^T x = \sum_{i=1}^{n} w_i(t) x_i$ is a whole new function over $(-\infty, t]$ for any t, the signals $s(t)$ and $e(t)$ take only the "last" values of these functions at all times t.

6.2 An algorithm based on Cholesky factor tracking

In this section, we show that the continuous-time RLS problem amounts to tracking the Cholesky factor of the input information matrix. The derivation is similar to the classical discrete-time derivation mostly given in terms of QR updating. Next, using lemma 3.4(i), we derive the continuous-time GK algorithm (algorithm 6.10). Finally we give a local interpretation to the cells of the corresponding signal flow graph.

6.2.1 Solution by Cholesky factor tracking

Below, we derive a solution for the continuous-time RLS problem based on continuous-time Cholesky factor tracking. The derivation is analogous to the solution of the discrete-time RLS problem through QR decomposition of the

data matrix, whose rows are time samples of x, weighted by a forgetting mechanism. To this end, we define the information matrix of x and a generalized QR decomposition.

Definition 6.5
The continuous-time information matrix of a signal x, with the forgetting rate $\lambda \geqslant 0$, is given by

$$N_c(t) = \langle x, x^T \rangle_{(t)}.$$

If x is a stochastic signal, $\lambda N = \int_{-\infty}^{t} x(\tau)x(\tau)^T e^{-\lambda(t-\tau)} d\tau / \int_{-\infty}^{t} e^{-\lambda(t-\tau)} d\tau$ is an estimate of the correlation matrix $E\{x(t)x(t)^T\}$.

The following lemma generalizes the QR decomposition for matrices 'with a continuous row index'.

Lemma 6.6 (a generalized QR decomposition)
Let $x \in (L_{2,\lambda}(-\infty, t])^n$, let $N_c(t) = \langle x, x^T \rangle_{(t)}$ be the corresponding information matrix.

Then, if the input signals $x_1, \ldots, x_n$ are linearly independent,
(i) they can be written in a unique way as

$$x^T = {q_{(t)}}^T R(t),$$

where $R(t) \in \mathcal{G}_{upp}(n)^+$ and $q_{(t)} \in (L_{2,\lambda}(-\infty, t])^n$ with $\langle q_{(t)}, {q_{(t)}}^T \rangle_{(t)} = I$,

(ii) $R(t) = \langle q_{(t)}, x^T \rangle_{(t)}$,

(iii) $R(t) = \mathrm{cf}(N_c(t)) = \mathrm{cf}(\langle x, x^T \rangle_{(t)})$.

Remark 6.7
Note that at any time t, $q_{(t)}$ is a different vector of functions over $(-\infty, t]$. For the sake of concise notation, below, we will not explicitly denote the time dependence of q and R, nor of the norm and inner product, except when required to avoid ambiguity.

Proof:
(i) As for the classical QR decomposition, q and R could be obtained by Gram-Schmidt orthonormalization of $x_1, \ldots, x_n$, in terms of the inner product $\langle q_i, q_j \rangle$.
(ii) $\langle q, x^T \rangle = \langle q, q^T R \rangle = \langle q, q^T \rangle R = R$.
(iii) $\langle x, x^T \rangle = \langle R^T q, q^T R \rangle = R^T \langle q, q^T \rangle R = R^T R$.

□

Remark 6.8
The condition of linear independence is not a very strong one in theory. If for instance one assumes a linear independent history for $x_1, \ldots, x_n$ on the interval $(-\infty, 0)$, later

values cannot make them dependent again. However, in practice the signals can still get arbitrarily close to being dependent, especially due to the exponential forgetting function in definition 6.2. We will come back to this issue in section 6.6.

The following theorem gives the solution to the RLS optimization problem.

Theorem 6.9
Let x, y, w^{opt}, s and e be defined as in definition 6.3.
Let $x = q^T R$ as in lemma 6.6(i). Then

$$\begin{array}{lrcl} \text{(i)} & w^{opt}(t) & = & R^{-1}(t)\langle q_{(t)}, y\rangle_{(t)}, \\ \text{(ii)} & s(t) & = & x(t)^T w(t)^{opt} = q_{(t)}(t)^T \langle q_{(t)}, y\rangle_{(t)}, \\ \text{(iii)} & e(t) & = & y(t) - q_{(t)}(t)^T \langle q_{(t)}, y\rangle_{(t)}. \end{array}$$

Proof:
The RLS minimization problem in definition 6.3 can be recast as

$$w^{opt}(t) = \arg\min_{w(t)}(\langle y, y\rangle - 2w(t)^T\langle x, y\rangle + w(t)^T N_c w(t)).$$

This positive definite quadratic function of $w(t)$ is minimal (at time t) if the gradient $2(N_c w(t) - \langle x, y\rangle)$ is equal to 0. With $N_c = R^T R$ and $x = R^T q$, we obtain $w^{opt}(t) = R^{-1}\langle q, y\rangle$. This proves (i). By substituting (i) in the definition of $s(t)$ and $e(t)$ and substituting $x(t)^T R^{-1}(t)$ by $q_{(t)}$ (lemma 6.6(i)), one also finds (ii) and (iii).
□

6.2.2 Derivation of the continuous-time GK algorithm

To keep track of the solution of the RLS problem (as a time dependent optimization problem), we need to track R and $\langle q, y\rangle$. If we can track R, we can also track $\langle q, y\rangle$ by considering $x_{n+1} = y$ as an additional component of the input, $\hat{x} = [x^T\ y]^T$, and tracking an $(n+1)\times(n+1)$-dimensional Cholesky factor $\hat{R}$ of $\hat{N}_c = \langle \hat{x}, \hat{x}^T\rangle$. Then $R = \hat{R}_{\leqslant n, \leqslant n}$ and from lemma 6.6(ii), we have

$$\hat{R}_{\leqslant n, n+1} = \langle q, y\rangle, \tag{6.6}$$

such that

$$w^{opt} = R^{-1}\hat{R}_{\leqslant n, n+1}. \tag{6.7}$$

The entry $\hat{r}_{n+1,n+1}$ is not needed.

The derivation of a continuous-time algorithm to track such a Cholesky factor, now reduces to the application of lemma 3.4(i). From $\dot{\hat{N}}_c = \hat{x}\hat{x}^T - \lambda\hat{N}_c$, one obtains

Algorithm 6.10 (Continuous-time Gentleman-Kung algorithm)

$$\dot{\hat{R}} = \mathsf{upph}(\hat{R}^{-T}\hat{x}\hat{x}^T\hat{R}^{-1})\hat{R} - \frac{\lambda}{2}\hat{R}.$$

The initial value can be taken equal to $R(0) = \sigma I$, which can be interpreted as assuming a white noise history with variance σ^2 on $(-\infty, 0)$.

The first term of this algorithm is of type III (definition 5.1). The second term has a trivial realization. Figure 6.1 shows the signal flow graph of the continuous-time GK algorithm.

The residual signal $e(t)$ can be extracted from this signal flow graph with no additional effort as $e(t) = \eta_{n+1,n+1}(t)$. For, $\langle q_{(t)}, y\rangle_{(t)}$ in theorem 6.9(iii) is found as $\hat{R}_{\leqslant n,n+1}$ (6.6) in the signal flow graph of figure 6.1 and the signals $q_{(t)i}(t)$ are found as $q_{(t)i}(t)$. Therefore $\eta_{n+1,n+1}(t) = x_{n+1}(t) - \sum_{k=1}^{n} q_{(t)k}(t)\hat{r}_{k,n+1}(t) = e(t)$.
If one is interested in $s(t)$, it can be found as $s(t) = y(t) - e(t)$. Parallel realization of $w^{opt}(t)$ as given by (6.7) requires an extra realization of the back-substitution scheme similar to figure 5.4, with input $R_{\leqslant n,n+1}$.

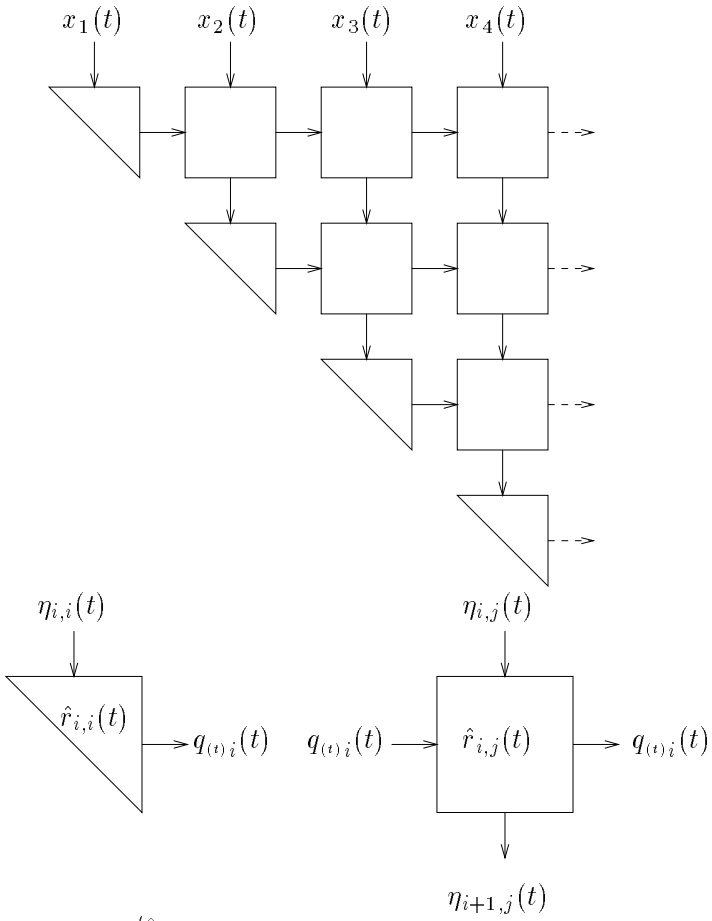
6.2.3 Local interpretation

The following lemma describes the meaning of local signals in the signal flow graph of the continuous-time GK algorithm. This is not only useful for understanding the algorithm, but also for the robustness analysis of an analog realization.

Theorem 6.11
In the signal flow graph of the continuous-time GK algorithm (figure 6.1)

(i) $\eta_{i+1,j} = x_j \perp \{x_1, \ldots, x_i\}$
(ii) $\eta_{i+1,j} = \eta_{i,i} \perp \eta_{i,j}$
(iii) $r_{i,i} = \|\eta_{i,i}\|$
(iv) $r_{i,j} = \langle \eta_{i,i}, \eta_{i,j}\rangle / \|\eta_{i,i}\|$
(v) $q_i(t) = \eta_{i,i}(t)/\|\eta_{i,i}\|$

Proof:
(i) This follows by regarding the cells of $R_{\leqslant i,\leqslant i}$ and $R_{\leqslant i,j}$ as a smaller signal flow graph for the problem with inputs $x_1, \ldots, x_i$ and reference input x_j.
(ii) This follows by regarding the cells (i,i) and (i,j) as a smaller signal flow graph for the problem with input $\eta_{i,i}$ and reference input $\eta_{i,j}$.



$$\text{(i,i)}: \quad \begin{cases} q_i &= \eta_{i,i}/\hat{r}_{i,i} \\ \dot{\hat{r}}_{i,i} &= \frac{1}{2} q_i \eta_{i,i} - \frac{\lambda}{2} \hat{r}_{i,i} \end{cases}$$

$$\text{(i,j)}: \quad \begin{cases} \eta_{i+1,j} &= \eta_{i,j} - q_i \hat{r}_{i,j} \\ \dot{\hat{r}}_{i,j} &= \frac{1}{2} q_i (\eta_{i,j} + \eta_{i+1,j}) - \frac{\lambda}{2} \hat{r}_{i,j} \end{cases}$$

Figure 6.1: Signal flow graph of the continuous-time GK algorithm (algorithm 6.10). Top: global picture and boundary conditions. Bottom: cells (i,i) and (i,j), $i > j$.

(iii,iv) Considering the same small problem as in (ii), we have from lemma 6.6(iii) that

$$\begin{bmatrix} r_{i,i} & r_{i,j} \\ 0 & r_{j,j} \end{bmatrix} = \mathrm{cf}(\begin{bmatrix} \langle \eta_{i,i}, \eta_{i,i} \rangle & \langle \eta_{i,i}, \eta_{i,j} \rangle \\ \langle \eta_{i,j}, \eta_{i,i} \rangle & \langle \eta_{i,j}, \eta_{i,j} \rangle \end{bmatrix}).$$

Since the Cholesky factor of a positive definite 2×2 matrix is given by

$$\mathrm{cf}(\begin{bmatrix} a & b \\ b & c \end{bmatrix}) = \begin{bmatrix} \sqrt{a} & b\sqrt{a} \\ 0 & \sqrt{c} \end{bmatrix},$$

(iii) and (iv) follow immediately. (One can also verify that the given formulas satisfy the differential equations of cells (i,i) and (i,j)).
(v) follows from $q_i = \eta_{i,i}/r_{i,i}$ in cell (i,i).
□

The connection between (i) and (ii) is as follows. The whole signal flow graph can be seen as as a composition of signal flow graphs for the problem with $n = 1$, where $\eta_{n+1,n+1}$ is calculated recursively according to the recurrence relation

$$x_j \perp \{x_1, \ldots, x_i\} = (x_j \perp \{x_1, \ldots, x_{i-1}\}) \perp (x_i \perp \{x_1, \ldots, x_{i-1}\}),$$

which can be easily verified.

6.3 The link with discrete-time systolic algorithms

In this section it is shown how algorithm 6.10 can be regarded as a limit of the discrete-time Gentleman-Kung systolic algorithm for an infinite sampling rate. Similarly, the Gentleman Kung algorithm can be seen as an appropriate discretization of the continuous-time algorithm, obtained by assuming piecewise constant input.

The Gentleman Kung systolic algorithm [47] is a discrete-time systolic algorithm which tracks the Cholesky factor of the discrete-time information matrix.

Definition 6.12
The discrete-time information matrix of a signal $x^{[k]}$, with the forgetting factor ρ, satisfying $0 < \rho \leqslant 1$, is given by

$$N_d^{[k]} = \sum_{l=-\infty}^{k} \rho^{k-l} x^{[l]} x^{[l]T}.$$

If x is a stochastic signal, $(1-\rho)N_d$ (like λN_c) estimates the correlation matrix $E\{xx^T\}$.

Before considering the correspondence between the parallel realizations of the discrete-time and continuous-time GK algorithms, the following lemma describes the obvious correspondence between the discrete-time and continuous-time information matrices, $N_c(t)$ and $N_d^{[k]}$ in two alternative ways.

Lemma 6.13
(i) Let $x^{[k]} = x(kT_s)\sqrt{T_s}$, then, in the limit for $T_s \to 0$, keeping $kT_s = t$ constant, one obtains

$$\lim_{\{T_s \to 0, kT_s = t\}} N_d^{[k]} = N_c(t) = \langle x, x^T \rangle_{(t)},$$

with $\rho = e^{-\lambda T_s}$.

(ii) Let $\sigma = (1-\rho)/\lambda$, and let $x(t)$ be a piecewise constant input signal, taking constant values $x(t) = x^{[k]}/\sqrt{\sigma}$ on every interval $[kT_s, (k+1)T_s)$, then

$$N_d^{[k]} = N_c(kT_s).$$

Remark 6.14
We will again use the notation $\hat{N}_d$ for the information matrix of the extended input $\hat{x}^{[k]}$.

Remark 6.15
For the sake of conciseness and clarity, we will leave out the forgetting mechanism. Below, $\lambda = 0$, $\rho = 1$, $\sigma = T_s$.

Algorithm 6.16 (Discrete-time Gentleman Kung algorithm ($\rho = 1$))
The discrete-time GK algorithm is given by

$p_{1,\cdot} = \hat{x}^T$
for $i = 1, \ldots, n+1$

$$\left[\begin{array}{ccc} \hat{r}'_{i,i} & \cdots & \hat{r}'_{i,n+1} \\ \hline p_{i+1,i} & \cdots & p_{i+1,n+1} \end{array}\right] = \begin{bmatrix} \cos(\phi_i) & \sin(\phi_i) \\ -\sin(\phi_i) & \cos(\phi_i) \end{bmatrix} \left[\begin{array}{ccc} \hat{r}_{i,i} & \cdots & \hat{r}_{i,n+1} \\ \hline p_{i,i} & \cdots & p_{i,n+1} \end{array}\right]$$

where $\phi_i = \mathrm{atan}(p_{i,i}/\hat{r}_{i,i})$
end

(Discrete-time indices $^{[k]}$ are not denoted, $\hat{r}^{[k+1]}$ is denoted as $\hat{r}'$.)

A less detailed but compact description is given by

$$\hat{R}' = \left[\begin{array}{c|c} I_{n+1} & 0_{n+1\times 1} \end{array}\right] G^{n+1|n+2}(\phi_{n+1}) \ldots G^{1|n+2}(\phi_1) \left[\begin{array}{c} R \\ \hline \hat{x}^T \end{array}\right],$$

where $G^{i|n+2}(\phi_i)$ denote Givens rotations as in definition 2.16.

Like for the continuous-time algorithm the initial value can be taken equal to a multiple of I.

Figure 6.2 shows a signal flow graph of this discrete-time algorithm. A pipelined version is obtained by simply adding one unit delay on horizontal and vertical lines between any two cells. (That is, by applying the cut theorem for vertical cuts between the columns).

Each new sample $\hat{x}$ is passed down as $p_{i,}$ from top to bottom, affecting all rows of $\hat{R}$ successively, and being changed at each row, so that the i-th component is annihilated in the diagonal cell (i,i) ($p_{i+1,i} = 0$). To this end, appropriate rotation angles ϕ_i (or rather, their cosine and sine, denoted by c_i and s_i) are calculated in the diagonal cells (i,i) and passed horizontally. (Two possible choices of ϕ_i make $p_{i+1,i} = 0$, choosing $\phi_i \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ keeps $r_{i,i}$ positive.)

Similarly, in the signal flow graph of the continuous-time GK algorithm (figure 6.1), $\hat{x}$ is passed down as $\eta_{i,j}$ from top to bottom, affecting all rows of $\hat{R}$, and losing one component in the diagonal cell (i,i) of each row. Signals q_i play the role of ϕ_i. This correspondence between both signal flow graphs is explained below. Note that the correspondence between the Cholesky factors, tracked by both algorithms was already clear from lemma 6.13.

Interpretation 6.17
The signal flow graph of the continuous-time GK algorithm (figure 6.1) with $\lambda = 0$, can be interpreted as a limit for $T_s \to 0$ of the signal flow graph of the discrete-time GK algorithm (figure 6.2) with $\rho = 1$, supplied with input signals $\hat{x}^{[k]} = \hat{x}(t)\sqrt{T_s}$, by identifying

$$\begin{array}{rcl} \eta_{i,j} & = & \lim_{T_s \to 0} p_{i,j}/\sqrt{T_s} \text{ and} \\ q_i & = & \lim_{T_s \to 0} p_{i,i}/(\hat{r}_{i,i}\sqrt{T_s}) = \lim_{T_s \to 0} \phi_i/\sqrt{T_s}. \end{array} \tag{6.8}$$

Proof:
Let $\tilde{\eta}_{i,j} = p_{i,j}/\sqrt{T_s}$. The i-th Givens rotation calculates $\hat{r}'_{i,j}$ and $p_{i+1,j} = \tilde{\eta}_{i+1,j}\sqrt{T_s}$, $i = 1, \ldots, n$, $j = i, \ldots, n+1$ by

$$\begin{array}{rcl} \hat{r}'_{i,j} & = & \hat{r}_{i,j}\cos(\phi_i) + \tilde{\eta}_{i,j}\sqrt{T_s}\sin(\phi_i), \\ \tilde{\eta}_{i+1,j}\sqrt{T_s} & = & -\hat{r}_{i,j}\sin(\phi_i) + \tilde{\eta}_{i,j}\sqrt{T_s}\cos(\phi_i). \end{array} \tag{6.9}$$

From $\phi_i = \text{atan}(p_{i,i}/\hat{r}_{i,i})$ we have

$$\begin{array}{rcl} \cos\phi_i & = & \hat{r}_{i,i}/\sqrt{\hat{r}_{i,i}^2 + \tilde{\eta}_{i,i}^2 T_s}, \\ \sin\phi_i & = & \tilde{\eta}_{i,i}\sqrt{T_s}/\sqrt{\hat{r}_{i,i}^2 + \tilde{\eta}_{i,i}^2 T_s}. \end{array} \tag{6.10}$$

Substituting (6.10) into (6.9), taking the limit for $T_s \to 0$, and making the identifications (6.8), one obtains

$$\begin{array}{rcl} \dot{\hat{r}}_{i,j} & = & \lim_{T_s \to 0}(\hat{r}'_{i,j} - \hat{r}_{i,j})/T_s = q_i\eta_{i,j} - \frac{1}{2}q_i^2\hat{r}_{i,j}, \\ \eta_{i+1,j} & = & \eta_{i,j} - q_i\hat{r}_{i,j}, \end{array}$$

which is equivalent to the operation of a cell of the continuous-time algorithm. □

6.4 Related algorithms by LDL tracking

Instead of tracking $\hat{R} = \mathrm{cf}(\lambda \hat{N}_c)$, one can also make use of the related LU decomposition or LDL decomposition (see below). This leads to a new algorithm with some advantages and disadvantages for the analog realization. Also, on the basis of a local interpretation, a generalization can be made, converging to the same stationary behavior in the case of stationary input. The obtained result is a triangular structure of small LMS algorithms (least mean squares - algorithm 6.22), whose adaptation rates can be chosen such that the overall system tracks the RLS solution.

Let

$$\begin{aligned} \hat{L} &= \mathrm{lf}(\lambda \hat{N}_c) = \mathrm{lf}(\hat{N}_c), \\ \hat{U} &= \mathrm{lf}(\lambda \hat{N}_c), \\ \hat{B} &= \hat{L}^T, \\ \hat{D} &= \mathrm{diag}(\hat{U}). \end{aligned}$$

One can easily verify that because $\hat{N}_c$ is symmetric.

$$\hat{U} = \hat{D}\hat{L}^T = \hat{D}\hat{B} = \hat{D}^{\frac{1}{2}}\hat{R}. \tag{6.11}$$

The decomposition $\hat{N}_c = \hat{L}\hat{D}\hat{L}^T$ is also called an LDL decomposition.

Denoting

$$g = \hat{B}^{-T}\hat{x}, \qquad h = \hat{U}^{-T}\hat{x},$$

the solution of the RLS problem in terms of the LU decomposition is given by

$$\begin{array}{rcccl} w^{opt} & = & \hat{B}^{-1}_{\leqslant n,\leqslant n}\hat{B}_{\leqslant n,n+1} & = & \hat{U}^{-1}_{\leqslant n,\leqslant n}\hat{U}_{\leqslant n,n+1}, \\ s & = & g^T\hat{B}_{\leqslant n,n+1} & = & h^T\hat{U}_{\leqslant n,n+1}, \\ e & = & y - g^T\hat{B}_{\leqslant n,n+1} & = & y - h^T\hat{U}_{\leqslant n,n+1}. \end{array}$$

We can either track $\hat{U}$, or $\hat{B}$ and $\hat{D}$ (preferring $\hat{B} = \hat{L}^T$ to $\hat{L}$ to obtain an upper triangular form again).

With (6.11) and lemma 3.4(ii) one finds

Algorithm 6.18

$$\begin{aligned} \dot{\hat{U}} &= \hat{D}\,\mathsf{upp}(\hat{U}^{-T}\hat{x}\hat{x}^T\hat{U}^{-1})\hat{U} - \lambda\hat{U}, \\ \dot{\hat{D}} &= \mathsf{diag}(\dot{\hat{U}}). \end{aligned}$$

By another application of lemma 3.4, and with $\dot{\hat{D}} = \mathrm{diag}(\dot{\hat{U}})$, one finds

Algorithm 6.19

$$\begin{aligned}\dot{\hat{B}} &= \hat{D}^{-1}\,\mathsf{upps}(\hat{B}^{-T}\hat{x}\hat{x}^T\hat{B}^{-1})\hat{B},\\ \dot{\hat{D}} &= \mathsf{diag}(\hat{B}^{-T}\hat{x}\hat{x}^T\hat{B}^{-1}) - \lambda\hat{D}.\end{aligned}$$

One can show in a similar way as for the GK algorithm that these algorithms correspond to square-root free versions of the discrete-time GK algorithm, based on [46]. We will concentrate on algorithm 6.19.

Algorithm 6.19 differs only slightly from systems of type III in that an extra multiplication with $\hat{D}^{-1}$ is required. A signal flow graph is shown in figure 6.3. The vector $g = \hat{B}^{-T}\hat{x}$ is again obtained by back substitution as in figure 5.4. The residual $e = y - g^T\hat{B}_{\leqslant n,n+1}$ is again simply found as $\eta_{n+1,n+1}$. By considering all $\eta_{i,j}$ as residuals of smaller problems (as in theorem 6.11), it follows that the signals $\eta_{i,j}$ are the same as for the Cholesky decomposition based GK algorithm (figure 6.1).

The diagonal matrix $\hat{D}$ is kept in the diagonal cells, and is updated by

$$\dot{\hat{d}}_{i,i} = \dot{\hat{D}}_{i,i} = \eta_{i,i}^2 - \lambda\hat{d}_{i,i}.$$

The diagonal entries of $\hat{B}$ are equal to 1 and need not be stored. As a consequence no division is needed in the diagonal cells to calculate $g = \hat{B}^{-T}\hat{x}$. However, a division by $\hat{d}_{i,i}$ is now needed to calculate $\dot{b}_{i,j} = g_i\eta_{i+1,j}/\hat{d}_{i,i}$. Rather than supplying $\hat{d}_{i,i}$ to all cells, $g_i/\hat{d}_{i,i}$ can be calculated in the diagonal cell and supplied to the whole row (next to g_i, which is still needed in cell (i,j) for the calculation of $\eta_{i+1,j}$).

However, if one simply omits the division by $\hat{d}_{i,i}$, the model gets a simple interpretation, showing that there is still convergence to the desired stationary behavior for stationary input. The nondiagonal block gets as input $\eta_{i,j}$ and $\eta_{i,i}$ and gives as output $\eta_{i,j} - \hat{b}_{i,j}\eta_{i,i}$, which in a stationary case should become uncorrelated with $\eta_{i,i}$. This is realized indeed as $E\{\dot{b}_{i,j}\}$ gets zero expectation if $E\{\eta_{i,i}\eta_{i+1,j}\} = 0$. One can also see that this stationary state is attractive. Actually the nondiagonal cell now realizes the well known LMS algorithm (algorithm 6.22 below), with input $\eta_{i,i}$, reference input $\eta_{i,j}$ and residual $\eta_{i+1,j}$. In that interpretation, $1/\hat{d}_{i,i}$ plays the role of the adaptation rate μ of the LMS algorithm. The resulting overall picture is then a triangular structure built of LMS cells, whose adaptation rate can be organized such as to obtain exact correspondence with the RLS algorithm.

Leaving the adaptation rates free per row, the algorithm can be described by

Algorithm 6.20

$$\dot{\hat{B}} = M \,\mathsf{upps}(\hat{B}^{-T}\hat{x}\hat{x}^T\hat{B}^{-1})\hat{B},$$

where $M \in \mathcal{S}_{diag}(n)$.

Or with free adaptation rates per cell, one obtains

Algorithm 6.21

$$\dot{\hat{B}} = M .* \,\mathsf{upps}(\hat{B}^{-T}\hat{x}\hat{x}^T\hat{B}^{-1})\hat{B},$$

where $M \in \mathcal{S}_{upps}(n)$ and $.*$ denotes elementwise multiplication.

The LMS algorithm is a well known simple algorithm for the linear estimation problem, given by

Algorithm 6.22 (Least-mean-squares algorithm)

$$\begin{aligned} e &= y - w^T x, \\ \dot{w} &= \mu \cdot e \cdot x. \end{aligned}$$

The LMS algorithm is known to have inferior convergence properties to a stationary solution but may have better tracking properties in nonstationary cases, especially in the case of sudden changes in $\bar{w}$. [50]. The new algorithms 6.20 and 6.21 can do as well as RLS, and the new freedom in the choice of M creates room for improvement. Especially in discrete-time versions (triangular structures built with discrete-time LMS cells) one could use this freedom to devise adaptation rate strategies (step size strategies) for nonstationary environments. For analog realizations, it might be better to use this freedom to obtain simpler versions, discussed in section 6.6.

6.5 An algorithm by inverse Cholesky factor tracking

One can also track the inverse Cholesky factor $\hat{S} = \hat{R}^{-1}$ instead of $\hat{R}$ (still assuming that $\hat{R}$ is not rank deficient, see remark 6.8). For the discrete-time case this idea has been developed by Pan and Plemmons [88] (originally formulated in terms of the QR decomposition), and turned into a systolic algorithm by Moonen and McWhirter [80].

In continuous-time, the resulting algorithm is found again by applying lemma 3.4 for the inverse Cholesky factor, or starting from the continuous-time GK algorithm, by applying lemma 3.3, stating that $\dot{\hat{S}} = -\hat{S}\dot{\hat{R}}\hat{S}$. One finds

Algorithm 6.23

$$\dot{\hat{S}} = -\hat{S}\,\mathsf{upph}(\hat{S}^T\hat{x}\hat{x}^T\hat{S}) + \frac{\lambda}{2}\hat{S}.$$

This algorithm also fits in the class of definition 5.1. Its first term is of type II. The second term has a trivial realization.

An advantage of tracking $\hat{S}$ is that the weights $w^{opt}(t)$ can be extracted from the last column of $\hat{S}$. For $w^{opt} = R^{-1}\hat{R}_{\leqslant n,n+1}$ (6.7) and from

$$\hat{R}\hat{S} = \begin{bmatrix} R & \hat{R}_{\leqslant n,n+1} \\ 0 & \hat{r}_{n+1,n+1} \end{bmatrix}\begin{bmatrix} S & \hat{S}_{\leqslant n,n+1} \\ 0 & \hat{s}_{n+1,n+1} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & 1 \end{bmatrix},$$

one finds, $R\hat{S}_{\leqslant n,n+1} + \hat{R}_{\leqslant n,n+1}\hat{s}_{n+1,n+1} = 0$. Therefore

$$w^{opt} = -\hat{S}_{\leqslant n,n+1}/\hat{s}_{n+1,n+1}.$$

It also follows that
$q_{n+1} = \sum_{i=1}^{n+1}\hat{x}_i\hat{s}_{i,n+1} = -(\sum_{i=1}^{n} x_i(w_i^{opt}\hat{s}_{n+1,n+1})) + y\hat{s}_{n+1,n+1}$. Such that, $e = y - w^{opt}x = q_{n+1}/\hat{s}_{n+1,n+1}$. Finally $s = y - e = y - q_{n+1}/\hat{s}_{n+1,n+1}$.

Figures 6.5 and 6.6 show a signal flow graph for the discrete-time MM algorithm, in a version without pipelining [88, 80]. We give a slightly adapted description of the algorithm, tailored for later use in chapter 8.

Algorithm 6.24

$$S^{[k+1]} = S^{[k]}\left[\; I_{n+1} \;\middle|\; 0_{n\times 1} \;\right] G^{n+2|1}(\phi_1^{[k]})\ldots G^{n+2|n+1}(\phi_{n+1}^{[k]})\left[\frac{I_{n+1}}{0_{1\times(n+1)}}\right],$$

where $G^{j|n+2}(\phi_j^{[k]})$, $j = 1,\ldots,n+1$ denote Givens rotations (definition 2.16), with angles $\phi_j^{[k]}$ determined such that

$$[\hat{y}|-1]G^{n+2|1}(\phi_1^{[k]})\ldots G^{n+2|j}(\phi_j^{[k]}) = [0_{1\times j}|\hat{y}_{j+1}\ldots\hat{y}_{n+1}|\epsilon_{j+1}],$$

where $\hat{y} = \hat{x}^T\hat{S}$.

Remark 6.25
The Givens rotations $G^{n+2|j}$ are the transpose of the Givens rotations $G^{j|n+2}$ in algorithm 6.16 [88].

To understand the signal flow graph of figures 6.5 and 6.6, the above algorithm should be read as follows. Add an extra column of zeros to $\hat{S}^{[k]}$. Apply $n+1$ successive Givens rotations to the extra column and the columns of $\hat{S}^{[k]}$, designed so as to 'roll' $[\hat{y}|-1]$ into its last entry, and drop the extra column. The extra column is realized by the signals $\zeta_{i,j}$. The givens rotations are calculated (as $c_j = \cos(\phi_j)$ and $s_j = \sin(\phi_j)$) in the cells $(0,j)$. The last entry of $[\hat{y}|-1]$ (multiplied by the first j Givens rotations), is realized as ϵ_{j+1}.

One can show in a similar way as for the GK algorithm that in the limit for $T_s \rightarrow 0$, this signal flow graph corresponds with the signal flow graph of the continuous-time MM algorithm (figure 6.4).

6.6 Analog realization

The signal flow graph of the continuous-time GK algorithm and the LDL based variants 6.19, 6.20 and 6.21 are the simplest ones considered in this work, and good candidates for analog realization. In this section we consider some aspects of such a realization. For a more detailed discussion with simulation examples and a VLSI realization of the nondiagonal cell, that has been simulated with SPICE, we refer to the recent master's thesis of Moons and Leman [81] (in Dutch).

The most appealing features of analog realizations are speed and cheap production. Problems are accuracy, and the need to know the ranges of different signals, which often makes the design more application specific. VLSI chips for analog signal processing have attracted much recent attention. As very recent examples we mention [67] which implements the LMS algorithm (algorithm 6.22), and [18], which implements the neural network based algorithm of [63, 19, 95] for independent component analysis.

The continuous-time GK algorithm tracks the Cholesky factor of $\hat{N}_c$ However, tracking a positive multiple $a\hat{N}_c$ is good as well, since $\hat{R}$ then scales with $\sqrt{a}$, and q with $1/\sqrt{a}$, leaving $\eta_{n+1,n+1}$, and w^{opt} if needed, unchanged. (The choice of $\hat{N}_c$ was made above to obtain correspondence with the discrete-time GK algorithm). This property can be exploited to make the expected range of $r_{i,j}$ and q independent of the choice of the forgetting rate λ. To this end one simply has to track the Cholesky factor of $\lambda\hat{N}_c$, estimating $E\{\hat{x}\hat{x}^T\}$. If $\hat{R}$ now denotes the Cholesky factor of $\lambda\hat{N}$ instead of $\hat{N}$ one obtains

Algorithm 6.26

$$\dot{\hat{R}} = \lambda\,\mathsf{upph}(\hat{R}^{-T}\hat{x}\hat{x}^T\hat{R}^{-1})\hat{R} - \frac{\lambda}{2}\hat{R}.$$

As mentioned above, knowing the range of the realized signals is important for analog design. As another result of this modification the integration can now be realized by a simple RC circuit with $\frac{1}{RC} = \frac{\lambda}{2}$. Figure 6.7 shows a more detailed signal flow graph.

The division in the diagonal cell of the signal flow graph, can be realized with a differential amplifier with big amplification factor and negative feedback as in figure 6.8. This scheme computes $K\eta_{i,i}/(Kr_{i,i} - 1)$. For big K, this is a good approximation of $\eta_{i,i}/r_{i,i}$. A problem with such realizations however is that noise at the input of the amplifier is also amplified with K. The resulting scheme for the diagonal cell is equivalent to a nondiagonal cell with an amplifier feeding back the output $\eta_{i+1,i}$ multiplied by a big amplification factor K to the input q_i (figure 6.8). This reflects the fact that in equation $\dot{R} = \text{upph}(qq^T)R - \frac{\lambda}{2}$ the diagonal entry plays no different role than the other entries. The only difference arises in the computation of $q = R^{-T}x$. But even for that computation, it was argued that the signal flow graph for the calculation of $q = R^{-T}x$ (as in figure 5.4) is "the reverse" of the signal flow graph for the calculation of $x = R^T q$ from q, which has identical diagonal and nondiagonal cells. Whereas reversing the nondiagonal cells is achieved by replacing an addition by a subtraction (and exchanging inputs and outputs), in the reverse diagonal cell, one has to produce q such as to make the output $\eta_{i+1,i}$ zero. This can be done by feedback through the amplifier.

Another problem is the range of q_i. From theorem 6.11(i,iii), one finds that q_i is equal to the ratio of $\eta_{i,i}$ and its exponentially weighted root mean square. Normally, q_i is not large except when $\eta_{i,i}$ has been small for some time and then suddenly raises. Simulations, keeping q_i small by not letting $r_{i,i}$ decrease below a low threshold, show good results. This also avoids the problem that $\dot{r}_{i,i}$ is undefined for $r_{i,i} = 0$, corresponding to the fact that R is assumed to be full rank.

In [81], also the robustness against inaccurate component realizations, offsets, and time delays are analyzed by simulation. It turns out that most inaccuracies, have less effect if they are shared by all cells. For high frequency applications, the system becomes sensitive to delays in the division of the diagonal cell.

Realization of the LDL based algorithm 6.19 instead of the GK algorithm offers some advantages and disadvantages. First glance advantages are the simple diagonal cell and the fact that $\eta_{i+1,j}$ can be used instead of $\frac{1}{2}(\eta_{i,j}+\eta_{i+1,j})$. Below we will see that the algorithm is also more robust against possible inaccuracies of the dividers in the diagonal cell. Disadvantages are that an exact integrator is to be realized in the nondiagonal cells (without the forgetting mechanism, which is more difficult) and that an extra signal has to be passed horizontally (in the non-simplified version 6.19).

Like the GK algorithm, the algorithm can be modified to track the L and D

factor of $\lambda\hat{N}_c$ instead of $\hat{N}_c$. One obtains (using the same notation, $\hat{D}$, for the new $\hat{D}$ - note that $\hat{B} = \mathrm{lf}(\lambda\hat{N}_c)^T = \mathrm{lf}(\hat{N}_c)^T$ is still the same)

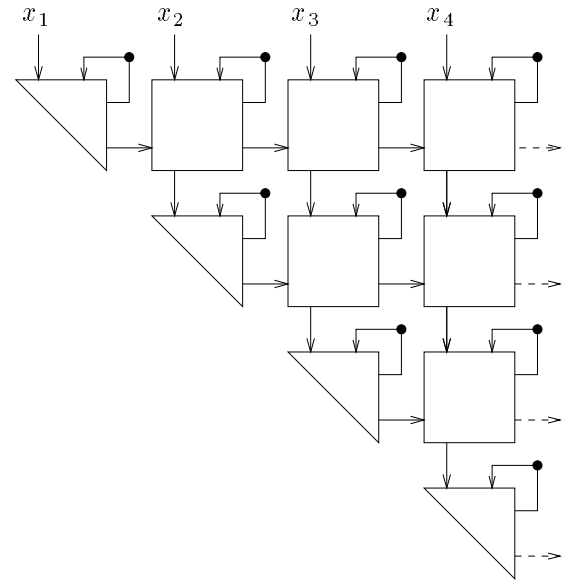
$$\begin{aligned}\dot{\hat{B}} &= \lambda\hat{D}^{-1}\,\mathrm{upps}(\hat{B}^{-T}\hat{x}\hat{x}^T\hat{B}^{-1})\,\hat{B},\\ \dot{\hat{D}} &= \lambda\,\mathrm{diag}(\hat{B}^{-T}\hat{x}\hat{x}^T\hat{B}^{-1}) - \lambda\hat{D}.\end{aligned}$$

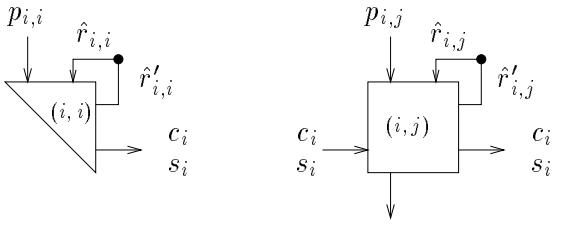
The adaptation rates of the LMS cells are now $\lambda/\hat{d}_{i,i}$. For this algorithm, one can afford to have inaccurate dividers, as deviations of the quotient $\eta_{i,j}$ simply correspond to other adaptation rates of the LMS cells. This is an advantage over the GK algorithm. For the same reason also the sensitivity to delays in the division disappears.

Compared to the LMS algorithm, a disadvantage is that the RLS type algorithms of this chapter require a number of cells of the order $\mathrm{O}(n^2)$ while LMS only requires $\mathrm{O}(n)$. However all feedback for the RLS algorithms is local, such that delay times in feedback loops (which are important for high frequency applications) do not increase with n for the RLS algorithm, while they are roughly proportional to n for the LMS algorithm. Like for the discrete-time case we believe that both algorithms will turn out to be useful, depending on the application.

6.7 Conclusion

We have derived three algorithms (algorithms 6.10, 6.23 and 6.19) for continuous-time recursive least-squares estimation. They are continuous-time versions of systolic arrays. Their behavior corresponds to that of the corresponding systolic algorithms when supplied with piecewise constant input. Once the discrete-time approach by QR updating was translated to the continuous-time framework (as Cholesky factor tracking, inverse Cholesky factor tracking or LU decomposition tracking), the derivation of the three algorithms followed from a simple application of lemma 3.4. As the algorithms fit in the class of definition 5.1, parallel signal flow graphs were obtained easily. Because of their simple form, two algorithms and some variants are candidates for analog realization. We have discussed high level aspects of this topic. In chapter 9 we will discuss the neural network interpretation of these algorithms. In chapter 8, the link between algorithm 6.23 and its discrete-time version will be exploited to derive new systolic algorithms for subspace tracking applications.

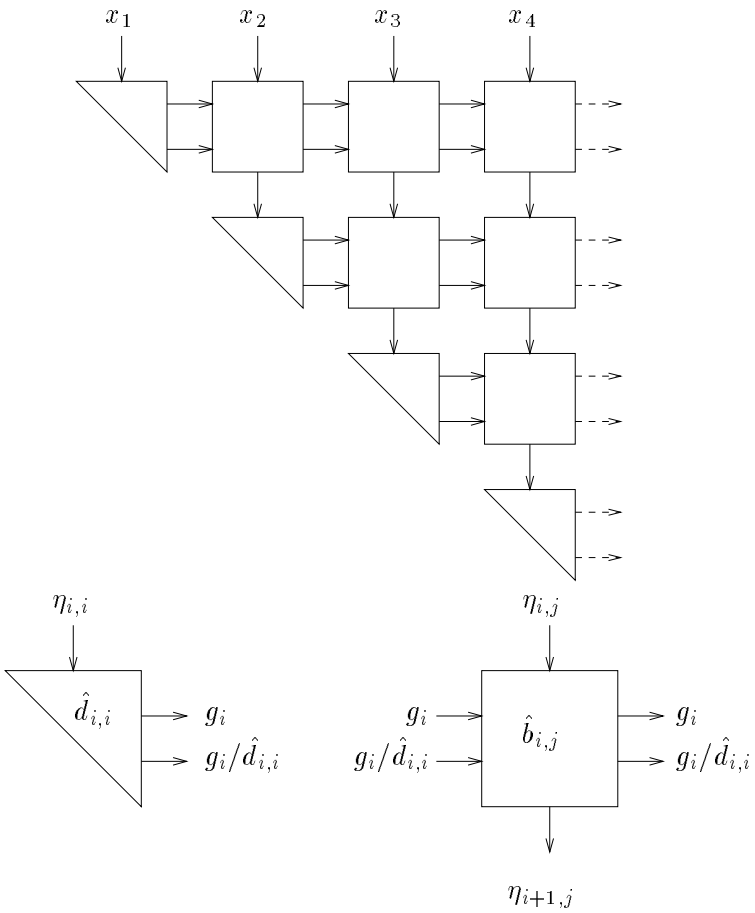




(i,i) : c_i, s_i and $\hat{r}'_{i+1,j}$ from $c_i^2 + s_i^2 = 1$ and $\begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix} \begin{bmatrix} \hat{r}_{i,i} \\ p_{i,i} \end{bmatrix} = \begin{bmatrix} \hat{r}'_{i,i} \\ 0 \end{bmatrix}$

(i,j): $\begin{bmatrix} \hat{r}'_{i,j} \\ p_{i+1,j} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix} \begin{bmatrix} \hat{r}_{i,j} \\ p_{i,j} \end{bmatrix}$

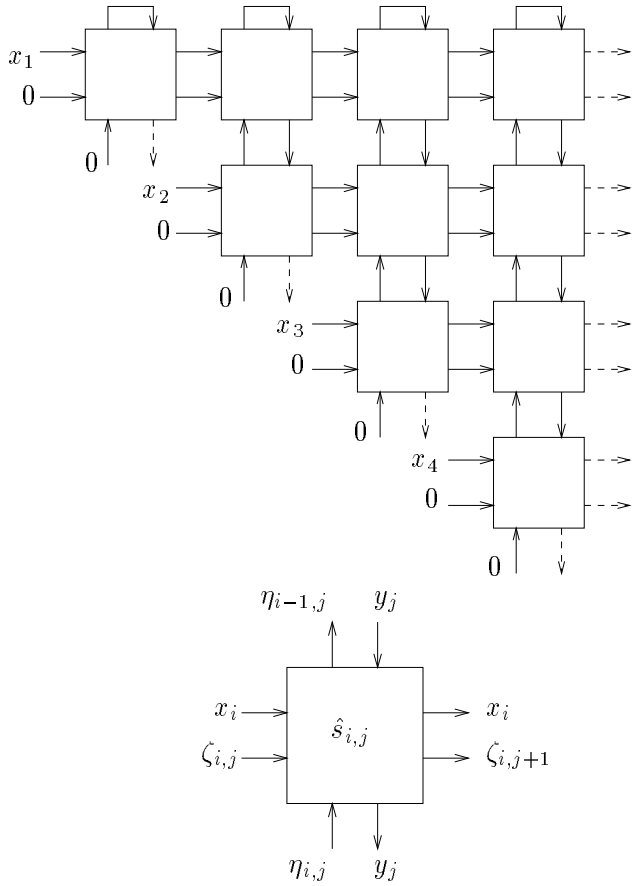
Figure 6.2: Signal flow graph of the (unpipelined) discrete-time Gentleman Kung algorithm (algorithm 6.16). Top: Global picture and input. Solid balls on arcs represent discrete-time unit delays. Bottom: cells (i,i) and (i,j), $i < j$.



$$\text{(i,i)}: \quad \begin{cases} g_i &= \eta_{i,i} \\ \dot{\hat{d}}_{i,i} &= \eta_{i,i}^2 - \lambda \hat{d}_{i,i} \end{cases}$$

$$\text{(i,j)}: \quad \begin{cases} \eta_{i+1,j} &= \eta_{i,j} - g_i \hat{b}_{i,j} \\ \dot{\hat{b}}_{i,j} &= \eta_{i+1,j}(g_i/\hat{d}_{i,i}) \end{cases}$$

Figure 6.3: Signal flow graph of the continuous-time algorithm 6.19. Top: global picture and boundary conditions. Bottom: cells (i,i) and (i,j), $i > j$.



$$\begin{aligned}
\eta_{i-1,j} &= \eta_{i,j} + x_i \hat{s}_{i,j} \\
\zeta_{i,j+1} &= \zeta_{i,j} + \hat{s}_{i,j} y_j \\
\hat{s}_{i,j} &= -\tfrac{1}{2}(\zeta_{i,j} + \zeta_{i,j+1}) y_j
\end{aligned}$$

Figure 6.4: Signal flow graph of the continuous-time MM algorithm (algorithm 6.23). Top: global picture, input and boundary conditions. Bottom: cell (i, j), $j \geqslant i$

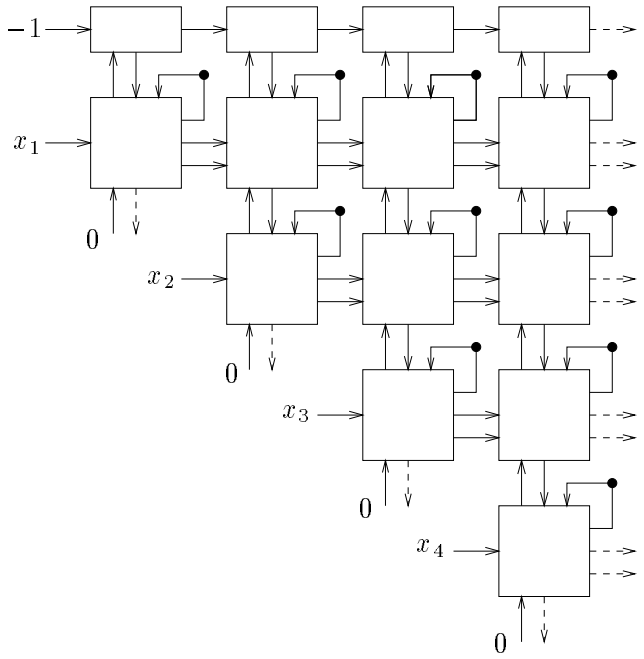
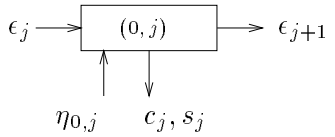
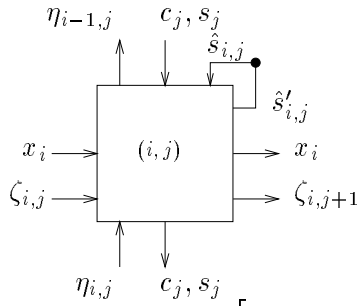


Figure 6.5: Signal flow graph of the (unpipelined) discrete-time Moonen-McWhirter algorithm (algorithm 6.24). Global picture and boundary conditions.





$$(0,j): \quad \left\{ c_j,\ s_j,\ \epsilon_{j+1} \text{ from } c_j^2 + s_j^2 = 1 \text{ and } [y_j\ \epsilon_j] \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} = [0\ \epsilon_{j+1}] \right.$$

$$(i,j): \quad \begin{cases} \eta_{i-1,j} = \eta_{i,j} + x_i \hat{s}_{i,j} \\ [\hat{s}'_{i,j}\ \zeta_{i,j+1}] = [\hat{s}_{i,j}\ \zeta_{i,j}] \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} \end{cases}$$

Figure 6.6: Signal flow graph of the (unpipelined) discrete-time MM algorithm (algorithm 6.24). Cell (i,j), $j \geqslant i$ and top cell $(0,j)$.

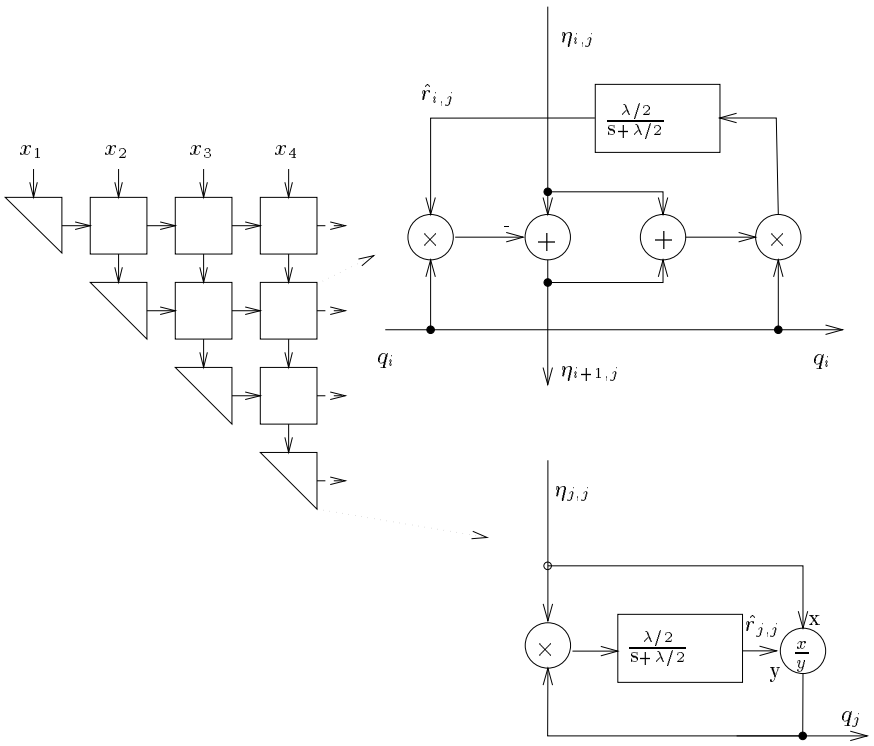


Figure 6.7: Signal flow graph for the realization of algorithm 6.26.

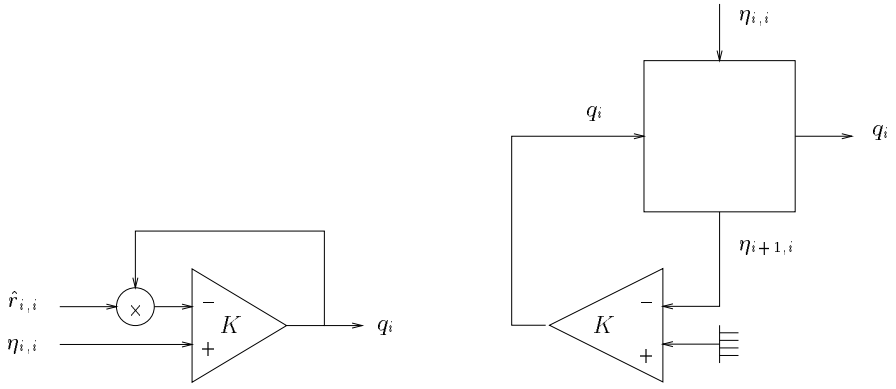


Figure 6.8: Left: Realization of the division $q_i = \eta_{i,i}/r_{i,i}$ with a multiplier and an amplifier. Right: Realization of a diagonal cell with a nondiagonal cell and an amplifier.

Chapter 7

Design and analysis of continuous-time subspace tracking algorithms.

In this chapter we consider a class of continuous-time algorithms, referred to as cross-product subspace trackers, that can be used for subspace tracking. We will pay special attention to one algorithm of the class, referred to below as algorithm C. We show that the algorithms are related to a variety of known discrete-time algorithms: stochastic gradient algorithms, [86, 107, 85], spherical subspace tracking algorithms [21], Oja's subspace algorithm [84] (independently obtained in different contexts and with different names by [105] and [3]), projection approximation subspace tracking [108], the generalized Hebbian algorithm [91], and others. We also refer to [68], which also gives a comparison with more classical algorithms, and to [61], putting these algorithms in a broader neural context. Neural interpretations date back to [83], and are discussed in chapter 9.

Remark 7.1 (names)
We will pay most attention to the continuous-time algorithm 7.10, and its rectangular version 7.9, which are formally most related to Oja's (discrete-time) stochastic gradient algorithm (algorithm 7.16). However this name is often used for a whole set of discrete-time subspace tracking algorithms (and also for other applications). Rather than inventing yet another name, with a trail of adjectives to distinguish it from all related algorithms, we refer to algorithm 7.10 and its rectangular version 7.9, as 'algorithm C' and allow the reader to substitute 'C' with his or her favorite name. In the next chapter, we will derive a new discrete-time algorithm, based on the continuous-time algorithm C. This discrete-time algorithm will be referred to as algorithm D.

Several authors [86, 91, 85, 40, 106] have already studied some of these discrete-time algorithms, using autonomous differential equations parameterized by the input correlation matrix $E\{xx^T\}$, to analyze the asymptotic behavior, referring to the theory of Ljung [73] or Kushner and Clark [70]. Our approach is different in that we use the continuous-time approach for the *design* of continuous time and discrete-time algorithms, and that we directly derive *non*autonomous continuous-time algorithms, without needing the equation with $E\{xx^T\}$. Under interpretation 7.18 below, we give a derivation of algorithm C, based on lemma 3.9, analogous to the derivation of the continuous-time GK algorithm 6.10, and using the idea of spherical subspace tracking. The derivation is much shorter than the lengthy derivation of the spherical subspace tracker [21] and of its relation with the Riccati differential equation [40]. In the next chapter we will make the connection with discrete-time algorithms, in the opposite direction than usual, deriving a new discrete-time systolic array, by integration of the continuous-time algorithm for piecewise constant input.

Next to the design and different interpretations, we also give a new analysis of algorithm C in section 7.4. Using the lemmas of section 3.1, we derive the evolution of the canonical angles between the subspace estimate and the subspace to be estimated. The analysis results will be inherited by the discrete-time algorithm derived in chapter 8.

This chapter is organized as follows. Section 7.1 defines the problem of subspace tracking, section 7.2 introduces a class of related continuous-time algorithms for subspace tracking and section 7.3 places them in different contexts. In section 7.4, we further analyze the algorithms and in section 7.5 we take some steps toward a further performance analysis based on these results.

The main results of this and the next chapter were published in [25].

7.1 Subspace estimation, subspace tracking

Subspace estimation, is a well known problem in signal processing, with a broad spectrum of applications, such as beamforming, adaptive noise cancellation, direction of arrival estimation and frequency estimation. We refer to [39, 98, 77, 50].

Definition 7.2 (Subspace estimation, subspace tracking)
Let $x(t) \in \mathbb{R}^n$ be a vector of given *input signals*, generated by

$$x(t) = Ms(t) + n(t), \tag{7.1}$$

where $M \in \mathbb{R}^{n \times \kappa}$ is of full rank, $\kappa < n$,
$s(t) \in \mathbb{R}^\kappa$ is the *source signal*, with nonsingular correlation matrix $E\{ss^T\}$

and $n(t)$ is the *noise signal* with $E\{nn^T\} = \sigma_n^2 I$.
The column space $\mathcal{S} = \text{col}(M)$ is called the *signal subspace*.
The orthogonal complement $\mathcal{N} = \mathcal{S}^\perp$ is called the *noise subspace*.

The *subspace estimation problem* consists in estimating the signal subspace $\mathcal{S}$, given x.
If M is slowly time-varying (compared to x), this problem is referred to as *subspace tracking*.

Remark 7.3
As the continuous-time algorithms below will be used for the analysis and design of discrete-time algorithms, we will avoid the complications of continuous-time stochastic processes. We will think of x as one physical realization of the process.

The following results are well known,

Proposition 7.4
Let x, in definition 7.2, be a stationary signal with $C = E\{xx^T\}$, then
(i) the signal subspace $\mathcal{S}$ is spanned by the κ principal eigenvectors of $C = E\{xx^T\}$. The noise subspace is spanned by the other eigenvectors.

(ii) an orthogonal matrix $A \in \mathbb{R}^{n\times m}$, spanning the signal subspace $\mathcal{S}$, solves the following optimization problem

$$A \in \arg \max_{A^TA=I} \text{Tr}(A^TCA).$$

Remark 7.5
By analogy with the generalized QR decomposition (for a 'continuous row index') of lemma 6.6, we could now define a generalized SVD. This would establish a more direct link with the classical SVD approach. In that approach proposition 7.4(i) is formulated in terms of the data matrix X, whose rows are time samples of the input x, weighted by a forgetting mechanism. The right singular vectors of X are the eigenvectors of N_d (definition 6.12). The singular values of X are the squares of the eigenvalues of N_d. As the left singular vectors are not considered in the sequel, it is more convenient to state everything directly in terms of the EVD of N_d or N_c.

By analogy with the derivation of algorithm 6.10, tracking the Cholesky factor of N_c (estimating a multiple of $C = E\{xx^T\}$), a logical step is now to design an algorithm to track the EVD of N_c. This derivation is given below under interpretation 7.18. The result is algorithm 7.19 whose parallel realization is more complex than the algorithms of definition 5.1, and requires that signals can get infinitely large. Only after an additional approximation, an algorithm fitting in the class of parallel algorithms of definition 5.1 is obtained. As this algorithm is related to a number of known discrete-time algorithms, we directly introduce it below and then discuss the link with different other algorithms.

7.2 Cross-product subspace tracking algorithms

We consider the class of cross-product subspace trackers (algorithms 7.6), two special cases (algorithms 7.8 and 7.9) and a square version (algorithm 7.10). Lemma 7.11 shows how they are related.

Algorithm 7.6 (Cross-product subspace trackers)

$$\dot{A} = \gamma \cdot (xx^T A + AK(A,t)),$$

where $A \in \mathbb{R}^{n\times m}$, $x(t) \in \mathbb{R}^n$, $K(A,t) \in \mathbb{R}^{m\times m}$ and $\gamma(t) \geqslant 0$. K and x are continuously differentiable and K is assumed to be sufficiently nice, for $\dot{L} = LK$ to have a globally existent solution in $\mathsf{GL}(m)$ (see remark 7.7).

The column space $\mathrm{col}(A)$ is an estimate of the signal subspace. The name 'cross product' refers to the product of x and $y^T = x^T A$.

Remark 7.7
The restriction on K is not a strong condition. If $K = 0$ it is satisfied trivially. In that case, also the global existence of $A(t)$, is obvious, from linear system theory. All other cases of interest below are of the form $K + A^T xx^T A \in \mathcal{S}_{skew}(m)$. For these cases global existence of full rank solutions will be proved in corollary 7.40.

The following algorithms are special cases.

Algorithm 7.8

$$\dot{A} = \gamma \cdot (xx^T A - AA^T xx^T A),$$

where $A \in \mathbb{R}^{n\times m}$, $x(t) \in \mathbb{R}^n$ and $\gamma(t) \geq 0$

Algorithm 7.9 (Continuous-time algorithm C, rectangular version)

$$\dot{A} = \gamma \cdot (xx^T A - 2A\,\mathsf{upph}(A^T xx^T A)),$$

where $A \in \mathbb{R}^{n\times m}$, $x(t) \in \mathbb{R}^n$, and $\gamma(t) \geqslant 0$.

Algorithm 7.10 (Continuous-time algorithm C, square version)

$$\dot{W} = \gamma \cdot (xx^T W - 2W\,\mathsf{upph}(W^T xx^T W)),$$

where $W(t) \in \mathbb{R}^{n\times n}$, $x(t) \in \mathbb{R}^n$, and $\gamma(t) \geqslant 0$.

The space $\mathrm{col}(W_{\cdot,\leqslant\kappa})$ is an estimate of the signal subspace. The space $\mathrm{col}(W_{\cdot,>\kappa})$ is an estimate of the noise subspace.

Lemma 7.11
(i) For any m the first m columns of the square version 7.10 of algorithm C follow the rectangular version 7.9:
if $A(0) = W(0)_{\cdot,\leqslant m}$, also $A(t) = W(t)_{\cdot,\leqslant m}$, $t \geqslant 0$.

(ii) All cross-product subspace trackers (algorithm 7.6) give the same subspace estimates:
Let A and $\tilde{A}$ satisfy

$$\begin{aligned} \dot{A} &= \gamma \cdot xx^T A, \\ \dot{\tilde{A}} &= \gamma \cdot xx^T \tilde{A} + \tilde{A}K, \end{aligned}$$

and let $\mathrm{col}(A(0)) = \mathrm{col}(\tilde{A}(0))$, then

$$\mathrm{col}(A(t)) = \mathrm{col}(\tilde{A}(t)).$$

Proof:
(i) Let $E_m = I_{\cdot,\leqslant m} \in \mathbb{R}^{n\times m}$. Let W obey the continuous-time algorithm C and let $A = WE_m$. Using $\mathrm{upph}(X)E_m = E_m \,\mathrm{upph}(E_m^T X E_m)$ we find

$$\begin{aligned} \dot{A} = \tfrac{d}{dt}(WE_m) &= \gamma \cdot (xx^T WE_m - 2WE_m \,\mathrm{upph}(E_m^T W^T xx^T WE_m)), \\ &= \gamma \cdot (xx^T A - 2A \,\mathrm{upph}(A^T xx^T A)). \end{aligned}$$

(ii) Since $\mathrm{col}(A(0)) = \mathrm{col}(\tilde{A}(0))$, there exists $L(0) \in \mathbb{R}^{m\times m}$ such that $\tilde{A}(0) = A(0)L(0)$. Define $L(t)$ and $\hat{A}$ by $\dot{L} = \gamma \cdot LK$, and $\hat{A}(t) = A(t)L(t)$. Then

$$\dot{\hat{A}} = \dot{A}L + A\dot{L} = \gamma(xx^T AL + ALK) = \gamma(xx^T \hat{A} + \hat{A}K).$$

Therefore (assuming existence and uniqueness of solutions) $\hat{A}(t) = \tilde{A}(t) = AL(t)$, and $A(t)$ and $\tilde{A}(t)$ span the same column space.
□

As a direct consequence, one can unambiguously define *cross product subspaces* (independent of the representation by a basis).

Definition 7.12 (Cross-product subspaces)
The cross-product subspaces of a signal x with weight γ are the spaces $\mathcal{C}_m(t) = \mathrm{col}(A(t))$, $m = 1, \ldots, n$, where $A \in \mathbb{R}^{n\times m}$ is given by any cross-product subspace tracker.

Remark 7.13
The initial condition in definition 7.12 is rather arbitrary. Any $A(0)$ of full rank could be used. We will mostly assume $A(0) = I$. Its influence on the subspaces generally decreases with increasing t, as is clear for instance from the convergence results of theorem 7.29 and [85].

Statement 7.14
The interesting thing about algorithm C (among other cross-product subspace trackers) is that *each* submatrix $W_{\cdot,\leqslant m}$, $m = 1, \ldots, n$, of W in algorithm C spans one of the cross-product subspaces $\mathcal{C}_m$, while the columns of $W_{\cdot,\leqslant m}$ converge to (or track) an *orthogonal* basis of $\mathcal{C}_m$.

This statement will be given a more precise content by the analysis of section 7.4 (theorems 7.29 and 7.37 and corollary 7.33). A consequence is also that the columns of W are estimates of the eigenvectors of $E\{xx^T\}$ (see interpretation 7.35).

Finally, observe that algorithm C has a uniform parallel realization of type II (definition 5.1), and algorithm 7.8 has a uniform parallel realization of type I. The signal flow graph of algorithm C is given in figure 7.1. Considering columns as neurons, it corresponds to the signal flow graph given by Oja [85] for algorithm 7.16.

7.3 Interpretations in different contexts

7.3.1 Relation with stochastic gradient algorithms

Interpretation 7.15
The continuous-time algorithm C is a continuous-time version of the discrete-time neural stochastic gradient ascent algorithm [85], closely related to the generalized Hebbian algorithm [91]. In the same sense algorithm 7.8 is a continuous-time version of Oja's subspace algorithm [84]. The label 'stochastic gradient algorithm' is appropriate in some sense, for all cross-product subspace trackers.

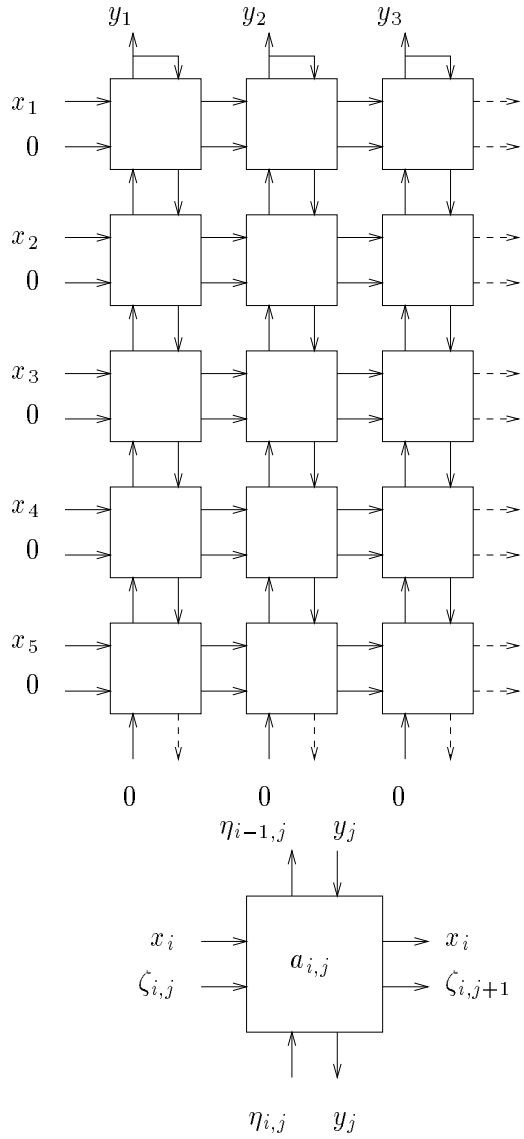
With our notation upph, the discrete-time stochastic gradient ascent algorithm can be put into the form

Algorithm 7.16

$$\begin{aligned} W^{[k+1]} &= W^{[k]} + \gamma^{[k]}\{x^{[k]}x^{[k]T}W^{[k]} \\ &\quad -2W^{[k]}\,\mathrm{upph}(W^{[k]T}x^{[k]}x^{[k]T}W^{[k]})\}, \end{aligned}$$

where $\gamma^{[k]} > 0$ is a scalar.

The resemblance with algorithm C is obvious. In the same sense algorithm 7.8 is a continuous-time version of Oja's subspace algorithm [84]. A slightly different discrete version with upp instead of 2 upph, was described in [91]. This algorithm



$$\begin{aligned} \eta_{i-1,j} &= \eta_{i,j} + x_i a_{i,j} \\ \zeta_{i,j+1} &= \zeta_{i,j} + a_{i,j} y_j \\ \dot{a}_{i,j} &= \gamma(x_i - \zeta_{i,j} - \zeta_{i,j+1}) y_j \end{aligned}$$

Figure 7.1: Signal flow graph for the continuous-time algorithm C. Top: Global picture, input, output and boundary conditions. Bottom: cell (i, j).

is called the generalized Hebbian algorithm, referring to the neural network interpretation.

The name 'stochastic gradient algorithms' comes from the formulation of subspace estimation as an optimization problem as in proposition 7.4(ii) and part (i) of the following lemma

Lemma 7.17
Let $\Phi(A) = \frac{1}{2}\,\mathrm{Tr}(A^T C A)$, with $A \in \mathbb{R}^{n\times m}$ and with $C \in \mathcal{S}_{symm}(n)$ constant. Then,

(i) the gradient of $\Phi(A)$ as a function over $\mathbb{R}^{n\times m}$ (with the standard inner product of definition 2.7) is

$$\mathrm{grad}_{A\in\mathbb{R}^{n\times m}}\Phi = CA.$$

(ii) the gradient of $\Phi(A)$ as a function over $\mathrm{St}(m,n) = \{X \in \mathbb{R}^{n\times m} | X^T X = I\}$ (with the standard inner product - see lemma 3.20(ii)) is

$$\mathrm{grad}_{A\in\,\mathrm{St}(m,n)}\Phi = CA - AA^T CA.$$

Proof:
(i)

$$\frac{d}{dt}\frac{1}{2}\,\mathrm{Tr}(A^T C A) = \frac{1}{2}\,\mathrm{Tr}(\dot{A}^T C A + A^T C \dot{A}) = \langle \dot{A}, CA\rangle. \tag{7.2}$$

By definition 3.21 of the gradient we obtain (i).

(ii) By proposition 2.22(i) a velocity $\dot{A}$ has the form $\dot{A} = A\Omega + B\Psi$ with $B \in \mathrm{St}(n-m,n)$, $B^T A = 0$, $\Omega \in \mathcal{S}_{skew}(m)$ and $\Psi \in \mathbb{R}^{(m-n)\times m}$. Working further on (7.2) to write it as an inner product of $\dot{A}$ and another tangent vector, yielding the gradient, we find

$$\begin{aligned}\dot{\Phi} &= \langle A\Omega + B\Psi, CA\rangle \\ &= \langle \Omega, A^T CA\rangle + \langle \Psi, B^T CA\rangle \\ &= \langle \Omega, 0\rangle + \langle \Psi, B^T CA\rangle.\end{aligned}$$

With lemma 3.20(ii) for the inner product of tangent vectors to $\mathrm{St}(m,n)$, one finds $\dot{\Phi} = \langle \dot{A}, A\cdot 0 + BB^T CA\rangle$ and finally with definition 3.21 of the gradient

$$\mathrm{grad}_{A\in\,\mathrm{St}(m,n)}\Phi = BB^T CA = CA - AA^T CA.$$

(Another proof is given in [106]).
□

Discrete-time stochastic gradient algorithms for subspace tracking, derive their name from the fact that they put a step in the direction $xx^T A$, estimating the gradient CA ($C = E\{xx^T\}$) with the current input sample, and then reorthogonalize. If one defines the gradient on the Stiefel manifold, the label gradient algorithm is most appropriate for algorithm 7.8. But wen we think in terms of a manifold of subspaces, rather than orthogonal bases for subspaces, all cross product subspace trackers again deserve the name 'stochastic gradient algorithm'.

7.3.2 Relation with spherical subspace trackers

Interpretation 7.18
With orthogonal initial conditions $W(0)$ (or $A(0)$ in algorithm 7.9), algorithm C and other cross-product subspace trackers can be obtained by a continuous-time derivation, using the idea of spherical subspace tracking [21].
The resulting subspace estimation algorithm corresponds to a continuous-time limit of the original discrete-time spherical subspace tracking algorithm, but with different orthogonal representations of the subspaces.

Before giving the continuous-time derivation, we give a brief description of the ideas of spherical subspace tracking, originally introduced for the discrete-time case.

Spherical subspace tracking was introduced in [21] as a reduced complexity approximation of the well known SVD approach (see remark 7.5, proposition 7.4). Exact updating of the matrix of singular vectors V, with each new time sample, would require $\mathrm{O}(n^3)$ per sample [48, p. 239], which is too much for many applications. Spherical subspace tracking is an $\mathrm{O}(n^2)$ approximation of this algorithm, obtained by the following approximation. At each iteration the signal singular values (i.e. the largest κ singular values) are replaced by their root mean square $\sigma_{(s)} = (\frac{1}{\kappa}\sum_{i=1}^{\kappa}\sigma_i^2)^{\frac{1}{2}}$, and the noise singular values are replaced by their root mean square $\sigma_{(n)} = (\frac{1}{n-\kappa}\sum_{i=\kappa+1}^{n}\sigma_i^2)^{\frac{1}{2}}$. By averaging the singular values in this way, the fact that the signal is more present in some directions of the signal subspace than in others is ignored, and the component of the signal in the estimated signal subspace is redistributed uniformly over all directions of that space. The same is done for the estimated noise subspace. This idea traces back to [64].

For a continuous-time derivation, using these ideas, it is more convenient to use a formulation in terms of the EVD of N_c (definition 6.5), estimating $\lambda E\{xx^T\}$.

From the definition 6.5 of N_c, it follows that

$$\dot{N} = xx^T - \lambda N.$$

If we assume that N has distinct eigenvalues, exact tracking of N is obtained by application of lemma 3.9, yielding

Algorithm 7.19

$$\begin{aligned}\dot{\Lambda} &= \mathsf{diag}(V^T xx^T V) - \lambda\Lambda,\\ \dot{V} &= V\mu(V^T xx^T V).\end{aligned}$$

Where μ is an elementwise scaling, defined by $\mu_{i,i}(X) = 0$, and $\mu_{i,j}(X) = X_{i,j}/(\lambda_j - \lambda_i)$, $i = 1,\ldots,n$, $j = 1,\ldots,n$, $j \neq i$. Even if we leave out of consid-

eration the problems involved with multiple eigenvalues, a signal flow graph for the parallel realization of this algorithm is necessary complex. If the entries of V are stored in the cells of a rectangular array, the number of connections between different cells would need to be proportional to the problem size. Alternatively one could think of a three-dimensional array.

However, as for the discrete algorithm, a reduction of the complexity is obtained by applying the idea of spherical subspace tracking. That is, the signal eigenvalues $\lambda_1, \ldots, \lambda_\kappa$ are replaced by $\lambda_{(s)} = \frac{1}{\kappa}\sum_{i=1}^{\kappa} \lambda_i$, and the noise eigenvalues are replaced by $\lambda_{(n)} = \frac{1}{n-\kappa}\sum_{i=\kappa+1}^{n} \lambda_i$. The eigenvalues are continuously reaveraged, by also averaging their time derivatives.

$$\begin{aligned} \dot{\lambda}_{(s)} &= \tfrac{1}{\kappa}\textstyle\sum_{i=1}^{\kappa}(W^T x)_i^2 - \lambda\lambda_{(s)}, \\ \dot{\lambda}_{(n)} &= \tfrac{1}{n-\kappa}\textstyle\sum_{i=\kappa+1}^{n}(W^T x)_i^2 - \lambda\lambda_{(n)}. \end{aligned}$$

where W denotes the resulting approximation of V. Applying lemma 3.9 to derive the evolution of W, we are now in the case with repeated eigenvalues. Therefore, the quantities $\omega_{i,j} = -\omega_{j,i}$ with $i, j \leqslant \kappa$ or $i, j > \kappa$ can be chosen arbitrarily. Different choices result in identical estimations for the signal and noise subspaces, but represented by different basis vectors, given by the columns of W. If the arbitrary components of Ω are chosen equal to 0, Ω can be written as

$$\Omega = \gamma \cdot (M E_\kappa E_\kappa^T - E_\kappa E_\kappa^T M), \tag{7.3}$$

with $M = W^T x x^T$, $\gamma = 1/(\lambda_{(s)} - \lambda_{(n)})$, and $E_\kappa = I_{\cdot,\leqslant\kappa}$ (lemma 3.9). This results in

$$\dot{W} = \gamma W (W^T x x^T W E_\kappa E_\kappa^T - E_\kappa E_\kappa^T W^T x x^T W). \tag{7.4}$$

For the first κ columns of W, $A = W_{\cdot,\leqslant\kappa} = W E_\kappa$, spanning the signal subspace, we find back algorithm 7.8

$$\dot{A} = \dot{W} E_\kappa = \gamma \cdot (x x^T A - A A^T x x^T A).$$

If the arbitrary components of Ω are not set to 0 but to $\gamma = 1/(\lambda_{(n)} - \lambda_{(s)})$ if $i > j$ and to $-\gamma$ if $i < j$, we find

$$\Omega = \gamma \rho_{\text{skew}}(M) = \gamma \cdot (M - 2\,\text{upph}(M)),$$

and

$$\begin{aligned} \dot{W} &= \gamma W (W^T x x^T W - 2\,\text{upph}(W^T x x^T W)) \\ &= \gamma (x x^T W - 2 W\,\text{upph}(W^T x x^T W)). \end{aligned} \tag{7.5}$$

This corresponds to algorithm C, with an adaptive scaling factor $1/(\lambda_{(s)} - \lambda_{(n)})$.

Remark 7.20
The way in which lemma 3.9 is applied to obtain (7.4) and (7.5), makes use of the fact that in continuous time the adaptation of the eigenvectors and the averaging of the eigenvalues are done simultaneously, to justify on the one hand the use of the

lemma for the case of *equal* eigenvalues and on the other hand present the adaptation of the averaged eigenvalues as an average of *differing* adaptations. A stricter justification can be found in the fact that the evolution of the subspace does not depend on its representation by a specific orthogonal basis (lemma 7.11(ii)). One could choose a representation with one signal (noise) eigenvector, parallel to the projection of x on the estimated signal (noise) subspace (as is done explicitly for the original discrete-time spherical subspace tracker). This is a set of eigenvectors as described in lemma 3.15(vi,v). In this representation only the eigenvalues corresponding to the parallel signal and noise eigenvectors will change and the change of the averaged eigenvalues will be the same as found above. In the end one finds again that really it was not necessary to change the representation, as the evolution of the subspace does not depend on it.

7.3.3 Relation with autonomous continuous-time algorithms

Interpretation 7.21
With orthogonal initial conditions $W(0)$, algorithm C can be interpreted as a stochastic version of the eigenvector algorithm associated with the QR-flow (algorithm 4.9). The matrix W, started from an orthogonal initial condition $W(0)$, tracks $\mathrm{qf}(B)$ where $\dot{B} = xx^T B$, $B(0) = W(0)$ (algorithm 7.6 with $K = 0$).

In section 4.1.2, the continuous-time QR algorithm (algorithm 4.9) was derived as an algorithm tracking the Q-factor of a matrix satisfying $\dot{X} = AX$, to find the eigenvectors of A. To find the eigenvectors of $C = E\{xx^T\}$, this yields

$$\dot{Q} = Q\rho_{\mathrm{skew}}(Q^T C Q).$$

A stochastic version, estimating C with the current input as xx^T, yields

$$\dot{Q} = Q\rho_{\mathrm{skew}}(Q^T xx^T Q).$$

As xx^T is symmetric, $\rho_{\mathrm{skew}}(Q^T xx^T Q) = Q^T xx^T Q - 2\,\mathrm{upph}(Q^T xx^T Q)$ (lemma 2.11(vi)), and with $QQ^T = I$, we obtain

$$\dot{Q} = xx^T Q - 2Q\,\mathrm{upph}(Q^T xx^T Q),$$

which is equivalent to algorithm C with $\gamma = 1$.

With lemma 3.4(iii), it is also clear that this algorithm, when started from an orthogonal initial matrix, tracks the Q-factor of B, obeying $\dot{B} = xx^T B$. This also holds for the rectangular algorithm 7.9.

Interpretation 7.22
Cross-product subspace trackers with $K + A^T xx^T A \in \mathcal{S}_{skew}(n)$, including algorithm C and algorithm 7.8 can be regarded as stochastic square root versions of a

Riccati equation, and with an orthogonal initial condition as stochastic 'square root' versions of the double bracket flow.

Consider the evolution of $Z = AA^T$, (we say that A is a 'square root' of Z), where A satisfies algorithm C or algorithm 7.8 or any cross-product subspace tracker (algorithm 7.6) with $K + A^T xx^T A \in \mathcal{S}_{skew}(n)$. One finds

$$\dot{Z} = \gamma \cdot (Z\dot{Z}^T + \dot{Z}Z^T) = \gamma \cdot (Zxx^T - 2Zxx^T Z + xx^T Z).$$

Substituting xx^T by $C = E\{xx^T\}$, and putting $\gamma = 1$ (or absorbing it into the time scale by $t' = \int_0^t \gamma(\tau)d\tau$) one obtains a Riccati equation. The link between algorithm 7.8 and Riccati equations has been studied in [106]. The connection between the double bracket flow and Riccati equations was studied in [53]. The connection between discrete-time spherical subspace tracking and Riccati equations was studied in [40].

If $A^T A = I$ and $Z = Z^2$ the equation can be recast as

$$\dot{Z} = \gamma \cdot (Z^2 xx^T - 2Z^T xx^T Z + xx^T Z^2) = \gamma \cdot [Z, [Z, xx^T]].$$

We will consider this equation in more detail in chapter 10. Substituting xx^T with $C = E\{xx^T\}$ one obtains the double bracket flow (algorithm 4.11). Finally we also mention another continuous-time algorithm for subspace tracking, described by Brockett [10], which is also based on the double bracket flow (in the form (4.3) with a diagonal matrix N), but which is more complex than algorithm C.

7.3.4 Geometric interpretations

Interpretation 7.23
For any cross-product subspace tracker, the evolution of col(A) can be interpreted instantaneously as a planar rotation towards x with angular velocity $\|x\|\|x_{(a)}\| \sin(\angle(x, x_{(a)}))$, where $x_{(a)}$ is the projection of x on col(A) (figure 7.2).

By lemma 7.11, we can reason with one particular subspace tracker and assume A orthogonal. With $A^T A = I$, algorithm 7.8 can be recast as

$$\dot{A} = (xx^T AA^T - AA^T xx^T)A = (xx_{(a)}{}^T - x_{(a)}x^T)A.$$

The interpretation as planar rotations follows from lemma 2.15 (or rather, from a straightforward generalization for rectangular matrices).

Interpretation 7.24
Interpreting algorithm C (or $\dot{B} = xx^T B$) in the local basis of lemma 3.12, the component affecting the columns q_i and q_j has angular velocity

$$\omega_{i,j} = (q_i^T x)(q_j^T x).$$

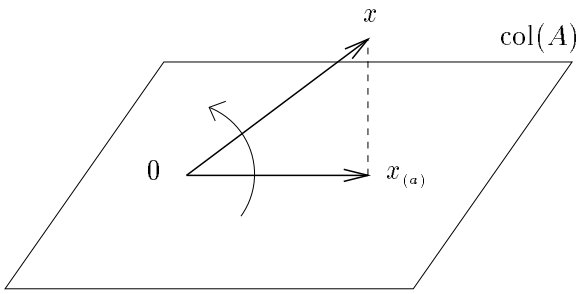


Figure 7.2: Geometric interpretation of the evolution of col(A) (interpretation 7.23).

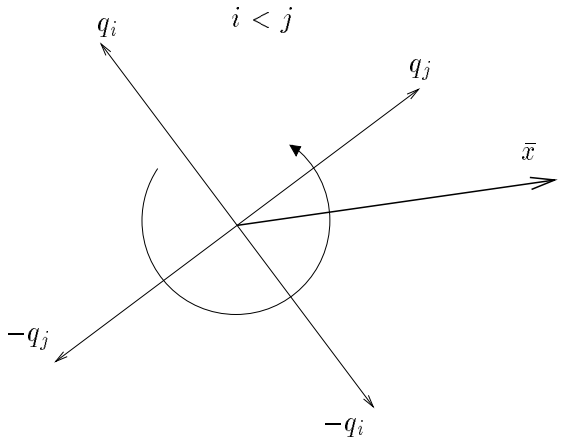


Figure 7.3: Rotation of q_i and q_j, $i < j$. The direction of q_i rotates towards $\bar{x}$, the projection of x on span$\{q_i, q_j\}$.

Inspecting the sign of $\omega_{i,j}$ one finds that the column with the lowest index, say q_i (or $-q_i$), rotates towards the projection of x on span$\{q_i, q_j\}$.

7.4 Analysis of the algorithm

In this section we analyze algorithm C and related algorithms.

First, we analyze the evolution of the estimated signal subspace, by considering the evolution of the canonical angles between the estimation and the generating subspace of the data model. This analysis is new. The interpretation for the

noise free stationary case, given in corollary 7.33 corresponds with the classical convergence result as explained in remark 7.36.

We also prove that the matrix A is attracted to the set of orthogonal matrices, from any initial matrix $A(0)$ of full rank, under very mild conditions for the input signals (theorem 7.37).

The properties analyzed in this section will be inherited by the discrete-time algorithm derived in chapter 8.

7.4.1 Evolution of the signal subspace estimate col(A)

We analyze the evolution of the signal subspace estimate col(A), by tracking the canonical angles between the estimated subspace col(A) and the generating subspace of the data model col(M).

To this end, we first recast the data model (7.1) for the subspace tracking problem (definition 7.25), and prove a lemma about the canonical angles (lemma 7.26). The main result is stated as theorem 7.29 and interpreted by interpretation 7.32. Finally we discuss the noise free stationary case in corollary 7.33 and the interpretation of the columns of A as eigenvectors 7.35.

Definition 7.25 (Modified data model)

$$x = \hat{M}\hat{m} + \hat{N}\hat{n},$$

with $[\hat{M}\hat{N}] \in \mathsf{O}(n)$, $(\hat{M}^T\hat{M} = I,\ \hat{N}^T\hat{N} = I,\ \hat{M}^T\hat{N} = 0)$.

This can be obtained from (7.1) by setting the columns of $\hat{M}$ to an orthonormal basis of col(M), say $M = \hat{M}R$ and setting

$$\begin{aligned} \hat{m} &= Rs + \hat{M}^T n, \\ \hat{n} &= \hat{N}^T n. \end{aligned}$$

That is, the subspace col(M) is represented by an orthogonal basis $\hat{M}$, s is adapted accordingly, and the component of n in col(M) is included in $\hat{m}$ as a signal. Expressing $\hat{m}$ and $\hat{n}$ in terms of x one simply obtains

$$\begin{aligned} \hat{m} &= \hat{M}^T x, \\ \hat{n} &= \hat{N}^T x. \end{aligned}$$

Below we will drop the hats on M,N,m and n. (To avoid confusion of the signals m and n with the matrix dimensions m and n, we will explicitly denote the time dependence of the former if necessary).

The canonical angles between M and A are related to the canonical angles between N and A by the following lemma.

Lemma 7.26
(i) Let $A, M \in \mathbb{R}^{n\times m}$, $N \in \mathbb{R}^{n\times(n-m)}$, $m \leqslant n-m$ and

$$A^TA = I \qquad M^TM = I \qquad N^TN = I \qquad M^TN = 0.$$

Then the SVD of M^TA and N^TA are related by

$$\begin{aligned} M^TA &= U\Sigma V^T, \\ N^TA &= U_{(n)}\Sigma_{(n)}V^T \qquad \text{(same } V\text{)}. \end{aligned}$$

$$\begin{aligned} &\sigma_i^2 + {\sigma_{(n)}}_i^2 = 1, \quad i = 1,\ldots,m, \\ &\sigma_{(n)i} = 1, \qquad\quad i = m+1,\ldots,n-m. \end{aligned}$$

(ii) the canonical angles (or principal angles) θ_i, between $\mathrm{col}(M)$ and $\mathrm{col}(A)$ and $\theta_{(n)i}$, between $\mathrm{col}(N)$ and $\mathrm{col}(A)$ are given by

$$\begin{array}{lcll} \cos(\theta_i) & = & \sin(\theta_{(n)i}) & = \sigma_i = \Sigma_{i,i}, \\ \cos(\theta_{(n)i}) & = & \sin(\theta_i) & = \sigma_{(n)i} = \Sigma_{(n)i,i}, \\ \theta_i & = & \angle(Av_i, Mu_i), & \\ \theta_{(n)i} & = & \angle(Av_i, Nu_{(n)i}), & \end{array}$$

$u_i = U_{.,i}$, $u_{(n)i} = U_{(n)_{.,i}}$, $v_i = V_{.,i}$, $i = 1,\ldots,m$.

Proof:
(i) This follows by considering the EVD of $A^TMM^TA = V\Sigma^T\Sigma V^T$ and $A^TNN^TA = {V_{(n)}\Sigma_{(n)}}^T\Sigma_{(n)}{V_{(n)}}^T$. Since $A^TMM^TA = A^T(I - NN^T)A = I - A^TNN^TA$, one has $V = V_{(n)}$ and $\sigma^2 = 1 - {\sigma_{(n)}}^2$.
(ii) this follows from the definition of canonical (principal) angles and vectors [48, section 12.4.3]).
□

Remark 7.27
The lemma can be adapted for the case $m > n-m$ by exchanging the role of N and M

Interpretation 7.28
Because $\mathrm{col}(N) = \mathrm{col}(M)^\perp$, θ_i and $\theta_{(n)i}$ are complementary and share the principal vector Av_i in $\mathrm{col}(A)$ (figure 7.4).

Theorem 7.29
Let $\bar{A} \in \mathbb{R}^{n\times m}$ obey any cross-product subspace tracker (definition 7.6), where x is a continuously differentiable signal, generated by the data model of definition 7.25 (we drop the hats on M,N,m,n).

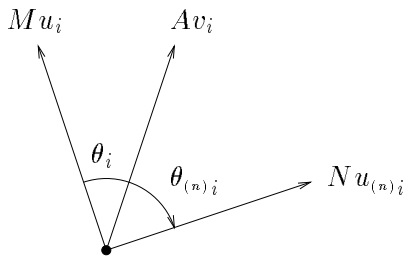


Figure 7.4: The canonical angles θ_i, between $\mathrm{col}(M)$ and $\mathrm{col}(A)$ and $\theta_{(n)i}$, between $\mathrm{col}(N)$ and $\mathrm{col}(A)$. And the corresponding canonical vectors $Av_i \in \mathrm{col}(A)$, $Mu_i \in \mathrm{col}(M)$ and $Nu_{(n)i} \in \mathrm{col}(N)$.

Let $M(t)$ be continuously differentiable and described by $\dot{M} = M\Omega + N\Psi \in T_M \mathrm{St}(m,n)$ (proposition 2.22(i)).
Let A, satisfy $A^TA = I$ and $\mathrm{col}(A) = \mathrm{col}(\bar{A})$, and let $U, U_{(n)}, \Sigma, \Sigma_{(n)}$ and V, be defined as in lemma 7.26 at any t.
then

$$\begin{aligned} \dot{\sigma}_i &= \gamma \cdot [(\mu_i^2 - \nu_i^2)\sigma_i\sigma_{(n)i}{}^2 - \mu_i\nu_i(1 - 2\sigma_i^2)\sigma_{(n)i}] + \sigma_{(n)i}\psi_i, \\ \dot{\sigma}_{(n)_i} &= \gamma \cdot [(\nu_i^2 - \mu_i^2)\sigma_{(n)i}\sigma_i{}^2 - \mu_i\nu_i(1 - 2\sigma_{(n)i}{}^2)\sigma_i] - \sigma_i\psi_i, \end{aligned} \tag{7.6}$$

$$\begin{aligned} \dot{\theta}_i &= -\tfrac{1}{2}\gamma(\mu_i^2 + \nu_i^2)\sin(2(\theta_i - \alpha_i)) + \psi_i, \\ \dot{\theta}_{(n)_i} &= -\tfrac{1}{2}\gamma(\mu_i^2 + \nu_i^2)\sin(2(\theta_{(n)i} - (\tfrac{\pi}{2} - \alpha_i))) - \psi_i, \end{aligned} \tag{7.7}$$

where

$$\begin{aligned} \mu_i &= u_i^T m = (Mu_i)^T x, \\ \nu_i &= u_{(n)_i}{}^T n = (Nu_{(n)i})^T x, \\ \alpha_i &= \mathrm{atan}(\nu_i/\mu_i), \\ \psi_i &= u_i^T \Psi^T u_{(n)i}, \end{aligned} \quad i = 1, \ldots, m. \tag{7.8}$$

Proof: (also see interpretation 7.32 below)
Since all cross-product subspace trackers track the same subspace, (starting from initial conditions spanning the same subspace, lemma 7.11(ii)), we can concentrate on one algorithm to track the canonical angles between $\mathrm{col}(A)$ and $\mathrm{col}(M)$ or $\mathrm{col}(N)$. As canonical angles are most easily defined, using orthogonal representations of the subspace, we choose algorithm 7.8 and assume A is orthogonal (by choosing $A(0)$ orthogonal). Because x is continuously differentiable, A is also continuously differentiable. Because M is continuously differentiable, so is M^TA. By lemma 3.7(iv), also Σ is continuously differentiable.

From lemma 3.9, when the singular values are distinct, Σ satisfies

$$\dot{\Sigma} = \mathrm{diag}(U^T \frac{d}{dt}(M^TA)V) = \mathrm{diag}(U^T(M^T\dot{A} + \dot{M}^TA)V).$$

With $\dot{A}$ as given by algorithm 7.8, $xx^T = Mmm^TM + Mmn^TN + Nnm^TM + Nnn^TN$ (from definition 7.25), and $\dot{M} = M\Omega + N\Psi$, one obtains

$$\begin{array}{rcll}\dot{\Sigma} & = & \mathrm{diag}(\gamma \cdot [& U^TM^TMmm^TM^TAV \quad -U^TM^TAA^TMmm^TM^TAV \\ & & + & U^TM^TMmn^TN^TAV \quad -U^TM^TAA^TMmn^TN^TAV \\ & & + & U^TM^TNnm^TM^TAV \quad -U^TM^TAA^TNnm^TM^TAV \\ & & + & U^TM^TNnn^TN^TAV \quad -U^TM^TAA^TNnn^TN^TAV] \\ & & + & U^T\Omega^TM^TAV \quad +U^T\Psi^TN^TAV).\end{array}$$

With $M^TA = U\Sigma V^T$ and $N^TA = U_{(n)}\Sigma_{(n)}V^T$, $M^TN = 0$, $M^TM = I$, $N^TN = I$ and $\mathrm{diag}(DX) = D\,\mathrm{diag}(X)$ for $D \in \mathcal{S}_{diag}(n)$, $X \in \mathbb{R}^{n\times n}$ one gets

$$\begin{array}{rcll}\dot{\Sigma} & = & \gamma \cdot [\mathrm{diag}(U^Tmm^TU)\Sigma & -\Sigma^2\,\mathrm{diag}(U^Tmm^TU)\Sigma \\ & & \mathrm{diag}(U^Tmn^TU_{(n)})\Sigma_{(n)} & -\Sigma^2\,\mathrm{diag}(U^Tmn^TU_{(n)})\Sigma_{(n)} \\ & & & -\Sigma\Sigma_{(n)}\,\mathrm{diag}({U_{(n)}}^Tnm^TU)\Sigma \\ & & & -\Sigma\Sigma_{(n)}\,\mathrm{diag}({U_{(n)}}^Tnn^TU_{(n)})\Sigma_{(n)}] \\ & & +\,\mathrm{diag}(U^T\Omega^TU)\Sigma & +\,\mathrm{diag}(U^T\Psi^TU_{(n)})\Sigma_{(n)}.\end{array}$$

For $\dot{\sigma}_i = \dot{\Sigma}_{i,i}$ one finds the given equation (7.6). The derivation of $\dot{\sigma}_{(n)_i}$ is completely analogous.

Finally, $\dot{\theta}_i$ (and $\dot{\theta}_{(n)_i}$) is obtained by the substitution $\cos(\theta_i) = \sigma_i$, yielding

$$\dot{\theta}_i = \gamma(\mu_i^2 - \nu_i^2)\cos(\theta_i)\sin(\theta_i) - \gamma\mu_i\nu_i(1 - 2\cos^2(\theta_i)) + \psi_i. \tag{7.9}$$

With $\cos(\theta_i)\sin(\theta_i) = \frac{1}{2}\sin(2\theta_i)$, $1 - 2\cos^2(\theta_i) = -\cos(2\theta_i)$, $\mu_i^2 - \nu_i^2 = -(\mu_i^2 + \nu_i^2)\cos(2\alpha_i)$ and $\mu_i\nu_i = \frac{1}{2}(\mu_i^2 + \nu_i^2)\sin(2\alpha_i)$, (7.9) becomes

$$\begin{array}{rcl}\dot{\theta}_i & = & \frac{\gamma}{2}(\mu_i^2 + \nu_i^2)(-\cos(2\alpha_i)\sin(2\theta_i) + \sin(2\alpha_i)\cos(2\theta_i)) + \psi_i \\ & = & -\frac{\gamma}{2}(\mu_i^2 + \nu_i^2)\sin(2(\theta_i - \alpha_i)) + \psi_i.\end{array} \tag{7.10}$$

Two more details are to be settled. In the above substitution we have assumed $\sin(\theta_i) \neq 0$. Nevertheless, the obtained formula (7.10) is valid if $\sin(\theta_i) = 0$ also, as follows for instance from an analogous derivation of (7.10) by substituting $\sin(\theta_i) = \sigma_{(n)_i}$ in the equation for $\dot{\sigma}_{(n)_i}$.

Finally, in the above application of lemma 3.9 we have assumed that the singular values are always distinct. In the case of a multiple singular value in an isolated point, one specific choice for u_i, $u_{(n)_i}$ may be needed in (7.8). From a result similar to lemma 3.15(v,vi) for the EVD, it follows that these should be chosen to diagonalize (in the sense of SVD) the submatrices ${U_{\cdot,\alpha}}^T\frac{d}{dt}(M^TA)V_{\cdot,\alpha}$ of $U^T\frac{d}{dt}(M^TA)V$, where α contains the indices corresponding to the repeated singular value. One finds that for one of the (identical) σ_i this vector u_i (and similarly for $u_{(n)i}$) is a scalar multiple of the projection of x onto the space of left singular vectors corresponding to σ_i, while for the others $\dot{\sigma}_i = 0$. In the unlikely cases that u_i is not continuously differentiable, $\dot{\sigma} = 0$, as follows from

the discussion of statement 3.18, allowing σ to be still continuously differentiable (in accordance with lemma 3.7(iv)). In the case of a multiple singular value over a time interval of nonzero length $\dot{\sigma}_i$ is zero and any choice of the corresponding singular vectors is appropriate.
□

Remark 7.30
Whereas canonical angles are often defined to lie in $[0, \frac{\pi}{2}]$, the above formulas are only valid for the cases $\theta = 0$ and $\theta = \frac{\pi}{2}$ if one allows theta to evolve outside this interval. Also α should be allowed to take any value. (Alternatively one can think of angles as equivalence classes $\theta + 2\pi\mathbb{Z}$).

Remark 7.31
Note that when the equation (7.6) for $\dot{\sigma}_i$ (a similar argument holds for $\dot{\sigma}_{(n)_i}$) is considered as a separate differental equation (without the equation for $\dot{\sigma}_{(n)_i}$)), solutions are not unique around $\sigma = \pm 1$. The right hand side is not Lipschitz continuous in σ at $\sigma = \pm 1$, as it is proportional to $\sigma_{(n)_i} = \pm\sqrt{1-\sigma_i^2}$. A solution $\sigma_i(t)$ arriving at 1 or -1 arrives there with $\dot{\sigma}_i = 0$, may stay there for some time and only then carry on, back down or back up into $(-1, 1)$, and still satisfy the differential equation (7.6). However it is clear that these solutions are not the right solutions, as follows for instance from the unique solution of $\sigma_{(n)_i}$ through 0 when σ_i comes through ± 1.

Interpretation 7.32
Theorem 7.29 (and its proof) considers the evolution of A in a time-varying coordinate frame, given by an orthonormal basis of canonical vectors Mu_i in col(M), $Nu_{(n)_i}$ in col(N) (and vectors orthogonal to both N and M if $m < n - m$).

From lemma 7.26 and interpretation 7.28, each angle θ_i is found in the plane spanned by Mu_i and $Nu_{(n)_i}$, and containing Av_i, referred to below as 'the plane of θ_i' (see figure 7.4). Now, θ_i could change due to a change of the subspaces col(M) and col(A) or due to evolution of Mu_i and Av_i in col(M) and col(A). However because θ_i is a canonical angle, the latter have no effect on $\dot{\theta}_i$ (canonical angles are by definition insensitive up to first order to a perturbation of the corresponding canonical vectors).

The equation (7.7) for $\dot{\theta}_i$ is also pictured in figure 7.5. The term $\frac{\gamma}{2}(\mu_i^2+\nu_i^2)\sin(2(\theta_i - \alpha_i))$ in equation (7.7) for $\dot{\theta}_i$ reflects the evolution of Av_i in the plane of θ_i. This term attracts θ_i towards α_i. The quantities $\mu_i = (Mu_i)^T x$ and $\nu_i = (Nu_{(n)i})^T x$ are coordinates of x in the coordinate frame of canonical vectors. The quantity $\mu_i^2 + \nu_i^2$ is the squared norm of the projection of x on the plane of θ_i, and α_i is the angle between the projection of x_i on the plane of θ_i, and the signal subspace col(M).

The term $\psi_i = u_i^T \Psi^T u_{(n)_i}$ reflects the component of the rotation of M and N in the plane of θ_i.

The transformations $e_{U^T,V}$ and $e_{U_{(n)}{}^T,V}$ are appropriate coordinate transformations of velocities, implied by the moving coordinate frame, and can be compared to the

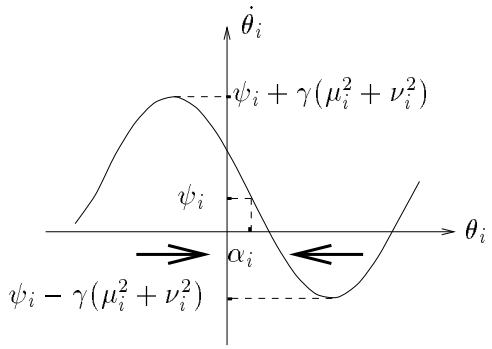


Figure 7.5: Evolution of the canonical angles θ_i: $\dot{\theta}_i$ as a function of θ_i.

transformation s_P of lemma 3.15 and remark 3.16.

Corollary 7.33
Assume the notation and conditions of theorem 7.29.
If $n(t) = 0$, $\dot{M} = 0$ and $\int_0^\infty \gamma\mu_i^2 d\tau = \infty$, $i = 1, \ldots, m$ (see interpretation 7.34), then each canonical angle θ_i, for which $\theta_i(0) \neq \frac{\pi}{2}$ converges to 0.

Proof:
With $n = 0$ and $\dot{M} = 0$, also $\nu_i = 0$ and $\Psi = 0$. Equation (7.6) becomes

$$\dot{\sigma}_i = \gamma\mu_i^2(\sigma_i - \sigma_i^3).$$

Substituting the time scale by $t' = \int_0^t \gamma(\tau)\mu_i^2(\tau)d\tau$, one obtains

$$\frac{d}{dt'}\sigma_i = \sigma_i - \sigma_i^3.$$

Figure 7.6 shows $\frac{d}{dt'}\sigma$ as a function of σ. Positive σ are attracted to 1. Considering $(\sigma_i - 1)^2(\sigma_i + 1)^2$ as a Lyapunov function, (or simply $\sum_{i=1}^m (\sigma_i - 1)^2$ for positive σ), one finds that all σ_i, for which $\sigma_i(0) \neq 0$, converge to 1, provided that the new time scale t' goes to ∞ when t goes to ∞. This is ensured by the condition $\int_0^\infty \gamma\mu_i^2 d\tau = \infty$. As a consequence θ_i converges to 0, if $\theta_i(0) \neq \frac{\pi}{2}$. (Alternatively one can directly think in terms of θ_i and consider figure 7.5 for $\alpha_i = 0$, $\psi_i = 0$ and $\nu_i = 0$.)
□

Interpretation 7.34
The condition $\int_0^\infty \gamma\mu_i^2 d\tau = \infty$ can be interpreted in a wide sense as 'x does not converge (or at least not too fast) to a subspace of col(M)'.

For the sake of clarity, we assume $\gamma = 1$. Other cases can be reduced to this case by taking $\int_0^t \gamma(\tau)d\tau$ as a new time scale.

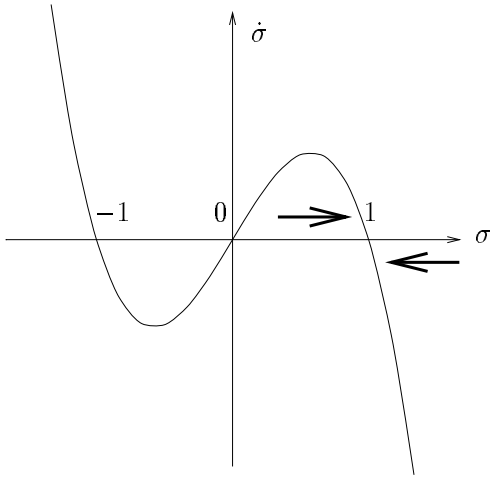


Figure 7.6: Plot of $\frac{d}{dt'}\sigma$ as a function of σ. Positive σ are attracted to 1.

For $\int_0^\infty \mu_i^2 d\tau$ to be finite, μ_i needs to go to 0 fast enough. Since $\mu_i = (Mu_i)^T x$ is the component of x in the direction of $Mu_i \in \mathrm{col}(M)$, this means that x converges fast enough to the subspace orthogonal to Mu_i. To justify interpretation 7.34, it is sufficient to prove that u_i converges if μ_i goes to 0 that fast. If more than one μ_i goes to 0 fast enough, say $\int_0^\infty \mu_i(\tau)^2 d\tau$ is finite for $i \in \eta \subset \{1, \ldots, n\}$, it is even sufficient that the subspace $\mathrm{col}(U_\eta)$ spanned by the corresponding u_i converges, as x then converges to $\mathrm{col}(U_\eta)^\perp$. That is, rotations of some u_i inside $\mathrm{col}(U_\eta)$ need not be considered.

Below, we show that $\dot u_i$ is proportional to μ_i such that in most cases u_i converges indeed. However, it is possible to contrive examples where u_i (or $\mathrm{col}(U_\eta)$) does not converge.

With lemma 3.9(ii), one finds

$$\dot u_i = \sum_{k=1}^{n} u_k \psi_{k,i},$$

where (in the case of distinct singular values, which can be assumed for $t \to \infty$ if $i \in \eta$ and $k \notin \eta$) $\psi_{k,i} = (\sigma_i n_{k,i} + \sigma_k n_{i,k})/(\sigma_i^2 - \sigma_k^2)$ with $n_{k,i} = u_k^T (M^T \dot A) v_i$. Substituting $M^T \dot A = M^T x x^T M M^T A - M^T A A^T M M^T x x^T M M^T A$ one finds

$$\psi_{k,i} = \mu_k \mu_i (\sigma_i^2 - 2\sigma_i^2 \sigma_k^2 + \sigma_k^2)/(\sigma_i^2 - \sigma_k^2).$$

If $i \in \eta$ and $k \notin \eta$, $\mu_k(\sigma_i^2 - 2\sigma_i^2\sigma_k^2 + \sigma_k^2)/(\sigma_i^2 - \sigma_k^2)$ converges to a finite value (actually to $-\mu_k(\infty)$ as $\sigma_k \to 1$). If $\int_0^\infty \mu_i$ is finite, $\mathrm{col}(U_\eta)$ converges. However, in the following (contrived) example $\int_0^\infty \gamma \mu_i^2 d\tau$ is finite while $\int_0^\infty \gamma \mu_i d\tau$ is infinite.

Consider $A \in \mathbb{R}^{n\times \mathbf{2}}$ and

$$\begin{aligned} \dot{A} &= xx^TA^T - AA^Txx^TA, \\ x &= \tfrac{1}{t+a}Mu_1 + Mu_2. \end{aligned}$$

The choice of x implies $\mu_1 = \frac{1}{t+a}$ and $\mu_2 = 1$. The constant $a > 0$ avoids infinite x at $t = 0$. With lemma 3.9(ii) one finds

$$\begin{aligned} \dot{\sigma}_1 &= \tfrac{1}{(t+a)^2}(\sigma_1 - \sigma_1^3), \\ \dot{\sigma}_2 &= \tfrac{1}{(t+a)^2}(\sigma_2 - \sigma_2^3), \\ \dot{u}_1 &= \psi_{1,2}u_2, \\ \dot{u}_2 &= -\psi_{1,2}u_1, \end{aligned}$$

where $\psi_{1,2} = \frac{1}{t+a}(\sigma_1^2 - 2\sigma_1^2\sigma_2^2 + \sigma_2^2)/(\sigma_2^2 - \sigma_1^2)$. If $\sigma_1(0) \neq 1$, σ_1 does not converge to 1 as it 'slows down too fast'. Meanwhile u_1 and therefore also x keeps evolving n a two-dimensional space. However, even here, because u_1 is evolving very slowly (its velocity converges to 0, as fast as μ, but without convergence of u_1 itself), one could say that actually x lies in a very slowly evolving *one*-dimensional subspace, which again explains the convergence of only one canonical angle.

We consider this example as a by-product of our deterministic approach, without further practical relevance. Further, we will simply require that 'x does not converge too fast to a subspace of $\mathrm{col}(M)$', also understanding 'not even a moving one with velocity going to 0'.
□

The above discussion defines the performance of the subspace trackers directly in terms of angles with the generating subspace $\mathrm{col}(M)$. Other authors take the EVD of $E\{xx^T\}$ (see proposition 7.4) as a reference. Or, in the case of nonstationary input signals the EVD of N_d is used (introducing the forgetting factor ρ into the performance measure). Below, we also interpret our result in terms of the EVD.

Interpretation 7.35
If W obeys algorithm C, with stationary input x, the columns of W estimate the eigenvectors of $E\{xx^T\}$. Also the eigenvalues can be estimated.

This was already implicitly contained in interpretations 7.15, 7.21, and 7.22 above, and is confirmed by corollary 7.33. Let $E\{xx^T\} = P\Lambda P^T$ be the EVD of $E\{xx^T\}$, then, by corollary 7.33, if W obeys algorithm C, all subspaces $\mathrm{col}(W_{\cdot,\leqslant m})$, $m = 1, \ldots, n$ can be viewed as estimates of the subspace $\mathrm{col}(P_{\cdot,\leqslant m})$, where $P_{\cdot,>m}P_{\cdot,>m}^Tx$ is a noise term disturbing the estimation. As this holds for $m = 1, \ldots, n$, if W is orthogonal (we will see below that W is attracted to the set of orthogonal matrices - theorem 7.37), the columns of W estimate the eigenvectors.

Also the eigenvalues of N_c can be estimated, by the equation

$$\dot{\lambda}_i = (A_{\cdot,i}^T x)^2 - \lambda\lambda_i,$$

(λ without subscript is the forgetting rate).

If A gave the exact eigenvectors of N_c, this formula would give the evolution of the eigenvalues, as follows from lemma 3.9. One can also prove that if A converges (for stationary x), λ_i converges to $A_{\cdot,i}^T E\{xx^T\}A_{\cdot,i}$ which is an eigenvalue of $\frac{1}{\lambda}E\{xx^T\}$. See [86] for corresponding discrete-time formulas. Finally note that the average of the obtained estimates for the signal or noise eigenvalues correspond to $\lambda_{(s)}$ and $\lambda_{(n)}$ for the spherical subspace tracker.

Remark 7.36
Corollary 7.33 is related to the result obtained by other authors (see [85] and references therein), using the theory of [73] or [70], to analyze the asymptotic behavior of algorithm 7.16, by studying the behavior of the autonomous differential equation $\dot{A} = CA - AA^TCA$ (also see [106]). To justify their analysis some constraints on $\gamma^{[k]}$ in algorithm 7.16 have to be imposed, typically [86]

$$\gamma^{[k]} \geqslant 0, \sum_{k=0}^{\infty} {\gamma^{[k]}}^2 < \infty, \sum_{k=0}^{\infty} \gamma^{[k]} = \infty. \tag{7.11}$$

Similar continuous-time conditions,

$$\gamma(t) \geqslant 0, \int_0^{\infty} \gamma(\tau)d\tau = \infty, \int_0^{\infty} \gamma(\tau)^2 d\tau < \infty, \tag{7.12}$$

can be interpreted as follows. The velocity $\dot{A}$ of the algorithm goes to 0, but slowly enough for the time scale $t' = \int_0^t \gamma(\tau)d\tau$ to go to ∞ as t goes to ∞. (Note that substitution of t by t' makes algorithms of the form $\dot{A} = \gamma F(A, x)$ independent of γ). In corollary 7.33, we do not assume that x goes to 0, but only $n(t) = 0$ (generalizations with n *going* to 0 could be made). Also, we need not assume the stationarity of $m(t)$ but only that $\dot{M} = 0$ and that $x(t)$ does not describe a smaller subspace (in the sense of interpretation 7.34). On the other hand, with the statistical approach, the interpretation of W as eigenvector estimates (interpretation 7.35) is established more directly as a convergence result: under the conditions 7.11, W converges (in a statistical sense) to the matrix of eigenvectors.

7.4.2 Attraction to orthogonality

Below, we consider the attraction of A to the set of orthogonal matrices, for a subclass of the cross-product subspace trackers, including algorithm C and algorithm 7.8.

Theorem 7.37
If $A \in \mathbb{R}^{n\times m}$ obeys algorithm C, algorithm 7.8 or any cross-product subspace tracker with $\bar{\Omega} = K + A^T x x^T A \in \mathcal{S}_{skew}(m)$,

$$\dot{A} = \gamma \cdot (x x^T A - A A^T x x^T A + A\bar{\Omega}), \tag{7.13}$$

then
(i) the singular values of A obey

$$\dot{\sigma_i} = \gamma(\sigma_i - \sigma_i^3)(u_i^T x)^2, \qquad i = 1, \ldots, n, \tag{7.14}$$

where σ_i is the i-th singular value and u_i is a corresponding left singular vector.

(ii) If $\int_0^\infty \gamma(u_i^T x)^2 dt = \infty$, A converges to the set of orthogonal matrices from any initial condition $A(0)$ with full rank.

Remark 7.38
A (long) proof of a result, similar to property (ii), using the approach of [73] (see remark 7.36), was given by [91] for the related generalized Hebbian algorithm (see interpretation 7.15). A similar result for algorithm 7.8, also through the substitution of xx^T by $E\{xx^T\}$, was given in [106] using the connection with Riccati equations. Our proof was given independently in [35] (in terms of eigenvalues instead of singular values) and in [24].

Proof:
(i) Let $A = U\Sigma V^T$ be the singular value decomposition of A. The evolution of the singular values of A is found by applying lemma 3.9 to the evolution of A (7.13). This yields

$$\begin{aligned} \dot{\Sigma} &= \gamma \operatorname{diag}(U^T x x^T U\Sigma - \Sigma\Sigma^T U^T x x^T U\Sigma + \Sigma V^T \bar{\Omega} V) \\ &= \gamma \operatorname{diag}(U^T x x^T U)\Sigma - \Sigma^2 \operatorname{diag}(U^T x x^T U)\Sigma. \end{aligned}$$

where we made use of $\operatorname{diag}(DX) = D \operatorname{diag}(X)$ for D diagonal, and $V^T\bar{\Omega}V \in \mathcal{S}_{skew}(m)$ implying $\operatorname{diag}(V^T\bar{\Omega}V) = 0$. Part (i) follows from $\dot{\sigma_i} = \dot{\Sigma}_{i,i}$. The case of multiple singular values can be treated as in the proof of theorem 7.29.

(ii) We prove that the singular values of A converge to 1. The proof is the same as for corollary 7.33 (also see figure 7.6), with x now replacing $\hat{m} = M^T x$ (and $u_i^T x$ replacing μ_i). Actually, in the noise free case with stationary subspace, $M^T A$ in corollary 7.33 follows algorithm C with input $M^T x$.
□

Remark 7.39
The condition $\int_0^\infty \gamma(u_i^T x)^2 dt = \infty$ can be interpreted in a similar way as the condition $\int_0^\infty \gamma\mu_i^2 d\tau = \infty$ in interpretation 7.34. We will again simply require that 'x does not converge too fast to a subspace of R^n' also understanding 'not even a moving one with velocity going to 0'.

Corollary 7.40
If $K(A,t)$ and $x(t)$ are continuously differentiable, any cross product subspace tracker with $A^T x x^T A + K \in \mathcal{S}_{skew}(m)$, has unique solutions $A(t)$, existing for all $t \in [0,\infty]$.

Proof:
Local existence and uniqueness follow from the continuous differentiability of the right hand side of the algorithm. Global existence follows from the fact that $\sum_{i=1}^{m}(\sigma_i-1)^2(\sigma_i+1)^2$ (or simply $\sum_{i=1}^{m}(\sigma_i-1)^2$ if one considers only nonnegative σ) is non-increasing and has compact sublevel sets. It also follows that if $A(0)$ is of full rank, $A(t)$ stays of full rank, as no singular value can get 0 once it is different from 0.
□

7.5 Notes on a statistical performance analysis

Theorem 7.29 gives intuitive insight into the tracking behavior of cross-product subspace trackers. It would be useful to turn this insight into quantitative mathematical formulas in a stochastic framework, such as formulas for the expected value θ_i in the case of moving subspaces ($\Psi \neq 0$), or the variance of θ_i in the case of stationary subspaces (in which case the expected value is 0). Some steps in this direction were taken in [23] and [20]. We briefly summarize some ideas, and possible assumptions that may simplify the analysis. This will also make clear that an elaborate investigation of this issue would lead us far from continuous-time algorithms and outside the scope of this work.

A first useful observation is that through the substitution $\vartheta_i = \tan(\theta_i)$, equation (7.7) for $\dot{\theta}_i$ becomes

$$\dot{\vartheta}_i = -2\gamma(\mu_i^2 + \nu_i^2)\vartheta_i + (\psi_i - \dot{\alpha}_i)(1+\vartheta_i^2), \tag{7.15}$$

which is linear in the noise free stationary case ($\alpha_i = 0$, ψ_i=0).

Further, since practical implementations of this algorithm will probably be discrete-time algorithms, we do not consider it useful to go through the complications of continuous-time stochastic processes, but propose to assume piecewise constant input signals, taking values $x^{[k]}$ in intervals $I^{[k]} = [kT_s, (k+1)T_s)$, as in the derivation of the discrete-time algorithm D in chapter 8.

If T_s is sufficiently small one can assume constant u_i, α_i, μ_i and ν_i on every interval. Under these conditions, and considering the case of a stationary subspace ($\psi_i = 0$) equation (7.7) for $\dot{\theta}_i$ can be integrated exactly over $I^{[k]}$, (using (7.15))

$$\theta_i^{[k+1]} = \alpha_i^{[k]} + \operatorname{atan}(e^{-\gamma^{[k]}(\mu_i^{[k]2}+\nu_i^{[k]2})T_s}\tan(\theta_i^{[k]} - \alpha_i^{[k]})), \tag{7.16}$$

where $\theta_i^{[k]} = \theta_i(kT_s)$.

The greatest challenge will be to take into account correlations between the input $x^{[k]}$ and the angles $\theta^{[k]}$, resulting from all previous input. This problem can of course be eliminated by assuming that the input is not correlated in time. However, for many applications, such an assumption does not correspond to reality. For the case of a moving subspace $\psi_i \neq 0$, we consider it reasonable to assume ψ_i independent of x and θ_i. Alternatively one could consider a deterministic approach, assuming a worst case constant value, reflecting the maximal angular velocity of a planar rotation component $\|\Psi\|_2$ of the rotation of M.

7.6 Conclusion

We have studied a class of continuous-time algorithms for subspace tracking, referred to as 'cross-product subspace tracking algorithms', and focused on algorithm C, (algorithm 7.10). This algorithm fits in the class of algorithms with a *uniform parallel signal flow graph* defined by definition 5.1.

The algorithms are related to a number of existing discrete-time algorithms. The continuous-time viewpoint has proved to offer a good framework for *unifying* different discrete-time algorithms. Moreover it provides geometric insight.

Besides, we have given a new *analysis*, using singular value tracking to show convergence to orthogonality and to analyze the tracking behavior. Continuous-time dynamical systems have been used before by other authors to analyze the statistic asymptotic behavior of these discrete-time algorithms, through the approach of [73] or [70]. Our approach is different in that we directly *derive and analyze non*autonomous *continuous*-time algorithms. Although we do not claim that this can always replace [73] or [70], one advantage is that we avoid some of the long derivations made by other authors. Moreover, whereas the more classical approach analyzes the asymptotic behavior for stationary input and under certain assumptions for the step size $\gamma^{[k]}$, we also obtain descriptions of the 'transient' behavior. Of course, the stochastic viewpoint is also important for applications, and we have formulated the problem of turning our new insight into statistical performance results. Although some initial steps were taken, this is left as an open problem.

In the next chapter we will derive a new discrete-time systolic version of algorithm C, referred to as algorithm D, inheriting the main properties of the continuous-time algorithm.

Chapter 8

Derivation of discrete-time systolic subspace tracking algorithms

In the previous chapter, we have described a continuous-time algorithm for subspace tracking, algorithm C. This algorithm fits in the class of systems with uniform parallelization, described in definition 5.1. The resulting signal flow graph is given in figure 7.1.

In this chapter we *convert the algorithm into a discrete-time algorithm, called algorithm D, by exact integration of the continuous-time algorithm for a piecewise constant input signal.* Each new value of the input corresponds to a new discrete iteration step.

The integration of the continuous-time algorithm and the parallel realization of the resulting discrete-time algorithm are inspired by an analogy with the MM algorithm 6.23 (figure 6.4) for recursive least squares estimation. Note that the second term in the continuous-time algorithm C is similar to the continuous-time MM algorithm 6.23.

The obtained discrete-time algorithm can be pipelined, horizontally and at the expense of a more complicated structure also vertically, again exploiting the analogy with the MM algorithm [80, 90]. This leads to implementations with problem size independent throughput. We also briefly consider a different realization storing $\mathrm{lf}(A)$ and $\mathrm{iuf}(A)$. With hindsight, these algorithms could be obtained also without the continuous-time approach. However, through this approach, they also inherit the properties obtained through the analysis of section 7.4. We come back to this point in chapter 12.

Finally we compare the algorithm with the original spherical subspace tracker and discrete-time stochastic gradient algorithms and give some numerical experiments.

This chapter is organized as follows. In section 8.1 we derive the discrete-time algorithm D (algorithm 8.3) and its signal flow graph (figures 8.1 and 8.2). In section 8.2 we turn it into a pipelined systolic signal flow graph (figures 8.5 and 8.6). Section 8.3 and appendix 8.A discuss the LU based variant, and section 8.4 compares the algorithm with related algorithms.

8.1 Exact integration for a piecewise constant input

The similarity between algorithm (6.23) for RLS and the second term of algorithm C, suggests that a discretization of algorithm C (in the square version 7.10 or rectangular version 7.9) exists, with a signal flow graph similar to figures 6.5 and 6.6. Below we derive this discrete-time algorithm (8.3) and its (unpipelined) signal flow graph, shown in figures 8.1 and 8.2.

Theorem 8.1 gives the integration of algorithm C for a constant input $x^{[k]}$ over the interval $[t^{[k]}, t^{[k+1]}]$ where $t^{[k+1]} = t^{[k]} + T_s$ and with the initial condition $A(t^{[k]}) = A^{[k]}$. The discrete-time algorithm is then defined by $A^{[k+1]} = A(t^{[k+1]})$.

Theorem 8.1
Let $x(t) = x^{[k]} \in \mathbb{R}^n$ be constant for $t \in [t^{[k]}, t^{[k+1]})$ and let

$$\begin{array}{rclcrcl} \dot{A}(t) &=& \gamma(x^{[k]}x^{[k]T}A(t) - 2A(t)\,\mathsf{upph}\,(A(t)^T x^{[k]}x^{[k]T}A(t))), & & A(t^{[k]}) &=& A^{[k]}, \\ \dot{B}(t) &=& \gamma x^{[k]T}x^{[k]}B(t), & & B(t^{[k]}) &=& A^{[k]}, \\ \dot{S}(t) &=& -2\gamma S(t)\,\mathsf{upph}\,(A(t)^T x^{[k]}x^{[k]T}A(t)), & & S(t^{[k]}) &=& I. \end{array}$$

Then,

$$\begin{array}{rcl} B(t) &=& A^{[k]} + \alpha(t)x^{[k]}x^{[k]T}A^{[k]}, \\ S(t) &=& \mathsf{icf}(I + \beta(t)A^{[k]T}x^{[k]}x^{[k]T}A^{[k]}), \\ A(t) &=& B(t)S(t), \end{array}$$

where

$$\begin{array}{rcl} \alpha(t) &=& \frac{\nu(t)-1}{x^{[k]T}x^{[k]}}, \\ \beta(t) &=& \frac{\nu(t)^2-1}{x^{[k]T}x^{[k]}}, \\ \nu(t) &=& \exp(x^{[k]T}x^{[k]}\int_{t^{[k]}}^{t}\gamma(\tau)d\tau). \end{array}$$

Proof: These formulas are easily verified, by derivation, using lemma 3.4(i), and observing that $x^{[k]T}B(t) = \nu(t)x^T A^{[k]}$ and $x^{[k]T}A^{[k]} = \nu(t)x^T A^{[k]}S(t)$.

A more constructive way to obtain the above formulas (though not strictly proving equivalence for nonorthogonal initial conditions) is by analogy with the derivation under interpretation 7.21. There, the continuous-time algorithm C was derived as an algorithm tracking $\mathrm{qf}(B)$, when started from an orthogonal initial condition, where $\dot{B} = xx^TB$, or $B(t) = e^{xx^Tt}A^{[k]} = (I + \alpha xx^T)A^{[k]}$. One finds $A(t) = A^{[k]}\,\mathrm{qf}(I + \alpha A^{[k]T}xx^TA^{[k]})$. The above formula for $A(t)$ is obtained by analogy with the substitution of $\rho_{\mathrm{skew}}(A^Txx^TA)$ by $A^Txx^TA - 2\,\mathrm{upph}(A^Txx^TA)$, but now substituting

$$\begin{aligned}\mathrm{qf}(I + \alpha A^{[k]T}xx^TA^{[k]}) &= (I + \alpha A^{[k]T}xx^TA^{[k]})\,\mathrm{irf}(I + \alpha A^{[k]T}xx^TA^{[k]})\\ &= (I + \alpha A^{[k]T}xx^TA^{[k]})\,\mathrm{icf}((I + \alpha A^{[k]T}xx^TA^{[k]})^2)\\ &= (I + \alpha A^{[k]T}xx^TA^{[k]})\,\mathrm{icf}(I + \beta A^{[k]T}xx^TA^{[k]}).\end{aligned}$$

□

Remark 8.2
A label (k) is to be added to B, S, α, β and ν to distinguish for example between $B^{(k)}(t^{[k+1]})$ and $B^{(k+1)}(t^{[k+1]})$. However, since below we never need to refer to $B^{(k+1)}$, we avoid this heavy notation. We will also denote $A^{[k]} = A(t^{[k]})$, $B^{[k]} = B(t^{[k+1]})$, $S^{[k]} = S(t^{[k+1]})$, $\alpha^{[k]} = \alpha(t^{[k+1]})$, $\beta^{[k]} = \beta(t^{[k+1]})$ and $\nu^{[k]} = \nu(t^{[k+1]})$. When no confusion is possible we will drop the discrete time index $^{[k]}$ and denote $^{[k+1]}$ with a prime.

The discrete-time algorithm can then be written as

$$\begin{aligned}y &= A^Tx,\\ \nu &= exp(x^Tx\textstyle\int_t^{t'}\gamma(\tau)d\tau),\\ \alpha &= \frac{\nu-1}{x^Tx},\\ \beta &= \frac{\nu^2-1}{x^Tx},\\ A' &= (A + \alpha xy^T)\,\mathrm{icf}(I + \beta yy^T).\end{aligned}$$

The parameter $\alpha^{[k]}$ (or $\nu^{[k]}$ or $\beta^{[k]}$) can be viewed as a step size, taking over the role of $\gamma(t)$ in the continuous-time algorithm. The integral formula for $\nu^{[k]}$ connects the continuous-time with the discrete-time algorithm. However, as we have not specified $\gamma(t)$, we can also consider $\alpha^{[k]}$ as a free parameter of the discrete-time algorithm. We get

Algorithm 8.3 (Algorithm D)

$$\begin{aligned}y &= A^Tx,\\ \beta &= 2\alpha + \alpha^2x^Tx,\\ A' &= (A + \alpha xy^T)\,\mathrm{icf}(I + \beta yy^T).\end{aligned}$$

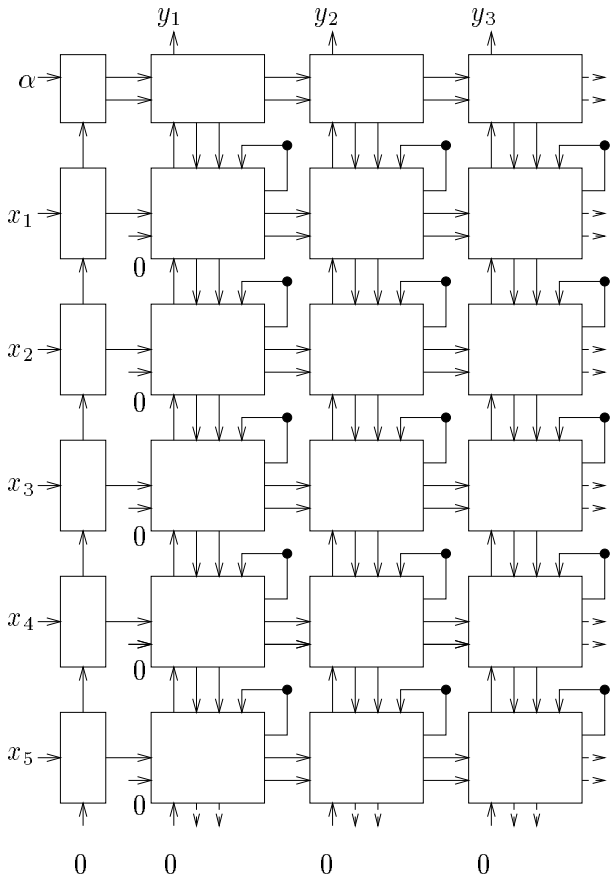
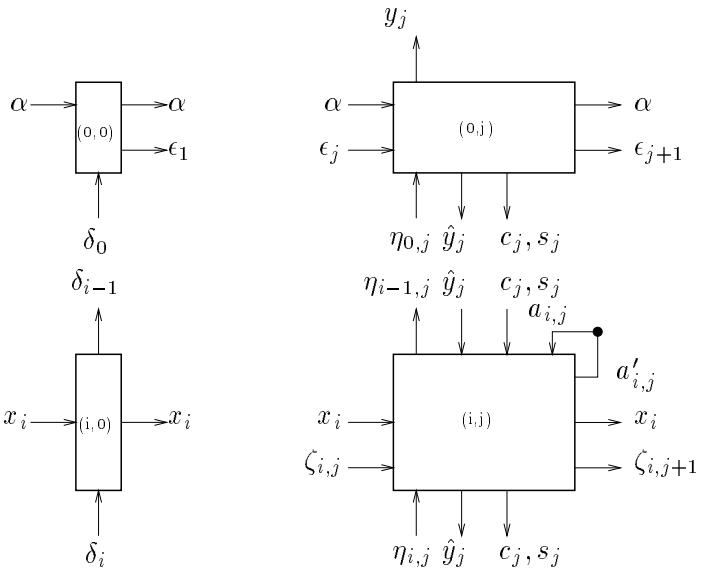


Figure 8.1: Signal flow graph for the discrete-time algorithm D. Global picture, input, output and boundary conditions.

Figures 8.1 and 8.2 show a signal flow graph for this discrete-time algorithm. It consists of a grid of $n \times m$ identical cells (i,j), bordered by a top row and left column of special cells $(i,0)$ and $(0,j)$ and a corner cell $(0,0)$ $(i = 1,\ldots,n,$ $j = 1,\ldots,m)$. Obtaining A' from A is done in two steps. First, B is obtained as $A + \alpha x x^T A$, then A' is obtained by multiplying with S.

To obtain B, the vector $y^T = x^T A$ is accumulated from bottom to top by the signals η in figures 8.1 and 8.2, each cell (i,j) calculating $\eta_{i-1,j} = \eta_{i,j} + x_i a_{i,j}$. The quantity $x^T x$ is accumulated by the signals δ in the cells $(i,0)$. The external input α in cell $(0,0)$ is passed to all cells $(0,j)$ of the top row. There $\hat{y} = \alpha y$ is calculated and sent downwards to the cells (i,j). With this information, each cell (i,j) can already add $x_i \hat{y}_j$ to $a_{i,j}$ to obtain $b_{i,j}$.



$$
\begin{aligned}
&(0,0): \quad \epsilon_1 = -1/\sqrt{2\alpha + \alpha^2\delta_0} \\
&(0,\mathrm{j}): \quad \begin{cases} y_j = \eta_{0,j} \\ \hat{y}_j = \alpha y_j \\ c_j,\ s_j,\ \epsilon_{j+1} \text{ from } c_j^2 + s_j^2 = 1,\ [y_j\ \epsilon_j] \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} = [0\ \ \epsilon_{j+1}] \end{cases} \\
&(\mathrm{i},0): \quad \delta_{i-1} = \delta_i + x_i^2 \\
&(\mathrm{i},\mathrm{j}): \quad \begin{cases} \eta_{i-1,j} = \eta_{i,j} + x_i a_{i,j} \\ [a'_{i,j}\ \zeta_{i,j+1}] = [a_{i,j} + x_i\hat{y}_j \ \ \zeta_{i,j}] \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} \end{cases}
\end{aligned}
$$

Figure 8.2: Signal flow graph for the discrete-time algorithm D. Cells $(0,0)$ $(0,j)$, $(i,0)$ and (i,j)

The multiplication by S to obtain $A' = BS$ is done by analogy with the MM algorithm 6.24. Setting $S^{[k]} = I$ and $S^{[k+1]} = S$ and $x = y/\sqrt{\beta}$ in algorithm 6.24 (where left hand sides use the notation of algorithm 6.24, right hand sides use the notation of this section), we obtain

$$
\begin{aligned}
S &= \mathrm{icf}(I + \beta yy^T) \\
&= \left[\ I_n \ \middle|\ 0_{n\times 1}\ \right] G^{n+1|1} \ldots G^{n+1|n} \left[\frac{I_n}{0_{1\times n}}\right],
\end{aligned} \tag{8.1}
$$

where $G^{n+1|j}(\phi_j)$, $j = 1, \ldots, n$ are Givens rotations (definition 2.16), with angles

ϕ_j determined such that

$$[y_1 \ldots y_n| - \frac{1}{\sqrt{\beta}}]G^{n+1|1}(\phi_1) \ldots G^{n+1|j}(\phi_j) = [0_{1\times j}|y_{j+1} \ldots y_n|\epsilon_{j+1}].$$

This can be read as follows. Post-multiplying a matrix with S means adding an extra column of zeros, applying n successive Givens rotations that roll $[y^T \ -\frac{1}{\sqrt{\beta}}]$ into its last entry, and dropping the extra column.

The extra column is realized by the signals ζ in figures 8.1 and 8.2. To calculate the Givens rotations, cell $(0,0)$ calculates $\epsilon_1 = -\frac{1}{\sqrt{\beta}}$ and every cell $(0,j)$ calculates $c_j = \cos(\phi_j)$ and $s_j = \sin(\phi_j)$ and ϵ_{j+1} from y_j and ϵ_j such that $c_j^2 + s_j^2 = 1$ and

$$[y_j \ \epsilon_j] \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix} = [0 \ \epsilon_{j+1}].$$

Finally to calculate $A' = BS$, the quantities c_j and s_j are passed downwards such that each cell (i,j) can calculate the $a'_{i,j}$ and $\zeta_{i,j+1}$ from $b_{i,j} = a_{i,j} + x_{i,j}\hat{y}_j$ and $\zeta_{i,j}$, by

$$[a'_{i,j} \ \zeta_{i,j+1}] = [b_{i,j} \ \zeta_{i,j}] \begin{bmatrix} c_j & s_j \\ -s_j & c_j \end{bmatrix}.$$

8.2 Pipelining

The signal flow graph of figures 8.1 and 8.2, does not yet describe a systolic algorithm, as information calculated in one cell is assumed to be available to other cells in the same discrete-time step. However it provides a good basis for deriving pipelined (systolic) versions of the algorithm. Below we describe how horizontal pipelining (which is trivial) and vertical pipelining can be obtained. With horizontal pipelining only, a throughput of $O(n^{-1})$ is achieved. With additional vertical pipelining a problem size independent throughput ($O(1)$) is achieved, at the expense of a more complicated array. The obtained result can then further be tailored to specific processor architectures.

Since there is information flow from left to right and not from right to left, horizontal pipelining can be obtained by simple retiming, introducing a delay on all horizontal arrows of the signal flow graph of figures 8.1 and 8.2. By mapping each column onto a processor, a linear systolic array is obtained. Since each processor has to perform the operations of the n cells of a column, this results in a throughput of $O(n^{-1})$.

One can obtain a problem size independent throughput at the expense of a complication of the array structure. To obtain this structure, we again follow the ideas of [80, 90], which are directly applicable to the multiplication by S in algorithm D. But the fact that first B is to be calculated causes some extra complications. The result is shown in figures 8.5 and 8.6.

Introducing delays on the vertical lines is more complicated than on the horizontal lines, because of the bidirectional information flow. The strategy of [80, 90] consists of two steps, described below. In a *first* step, shown in figures 8.3 and 8.4, delays are introduced on the lines going downwards, storing older versions of A in lower rows, and correcting for the error made by extra operations on the signals going upwards. The corrections are calculated in the cells $< i, j >$ $(i > 2)$ in figures 8.3 and 8.4. For these calculations, some products of the input vector x with older versions of x are needed, and calculated in the cells $[i, j]$. In the *second* step the fully pipelined array of figures 8.5 and 8.6 is obtained by two-slowing and cut-set retiming.

Different rows of the array now store rows of the matrix A from different steps of the discrete-time algorithm. At time k, when $x^{[k]}$ comes in, cell (i, j) calculates $a_{i,j}^{[k-i+2]}$ from $a_{i,j}^{[k-i+1]}$. The signals $\eta_{i,j}$, accumulating $y = A^T x$ in figures 8.1 and 8.2 are replaced by new signals $\tilde{\eta}$, and $\bar{\eta}$, similar to $\eta_{i,j}$ but calculated with an older version of A (not of x), depending on the row:

$$\begin{aligned} \tilde{\eta}_{i,j}^{[k]} &= x^{[k]T}_{>i} A^{[k-i+1]}_{>i,j}, \\ \bar{\eta}_{i,j}^{[k]} &= x^{[k]T}_{\geqslant i} A^{[k-i+1]}_{\geqslant i,j}. \end{aligned}$$

To calculate $\tilde{\eta}_{i-1,j}^{[k]}$, first $x_i^{[k]} a_{i,j}^{[k-i+1]}$ is added to $\tilde{\eta}_{i,j}^{[k]}$, yielding $\bar{\eta}_{i,j}^{[k]}$, then a correction is made, compensating for the use of an older version of A than needed in cell $(i-1, j)$.

This correction is derived as follows:

$$\begin{aligned} \tilde{\eta}_{i-1,.}^{[k]} &= x^{[k]T}_{>i-1} (A^{[k-i+2]}_{>i-1,.}) \\ &= x^{[k]T}_{>i-1} (A^{[k-i+1]}_{>i-1,.} + x^{[k-i+1]}_{>i-1} \hat{y}^{[k-i+1]^T}) S^{[k-i+1]} \\ &= (\bar{\eta}_{i,.}^{[k]} + \chi_i^{[k]} \hat{y}^{[k-i+1]^T}) S^{[k-i+1]}, \end{aligned}$$

where $\chi_i^{[k]} = x^{[k]T}_{>i-1} x^{[k-i+1]}_{>i-1}$. Note that the correction has the same form as the adaptation of A in every step, that is, adding something $(\chi_i^{[k]} \hat{y}^{[k-i+1]^T})$ and performing Givens rotations $(S^{[k-i+1]})$.

To calculate the coefficients $\chi_i^{[k]}$ an extra triangular piece (cells $[i, j]$) is added to the array. The calculation of $\tilde{\eta}^{[k]}$ is done in the cells $< i, j >$ $(i = 2, \ldots, n,$ $j = 1, \ldots, m)$. The signals $\epsilon_{i,j}$ implement the extra column for the realization of the Givens rotations.

Finally the array of figures 8.5 and 8.6 is obtained by applying two-slowing and cut-set retiming. First, by two-slowing, all delays are replaced by two delays. Even superscripts $[2z]$ in figures 8.5 and 8.6 correspond to superscripts $[z]$ in figures 8.3 and 8.4. Odd superscripts $[2z + 1]$ correspond to void signals. Also, the time between two consecutive input samples must now correspond to two delays instead of one. Slowing down by a factor two is largely compensated by the fact that the pipelining reduces the number of computations needed during

one delay from $O(n)$ (in the horizontally pipelined version) to $O(1)$ (in the completely pipelined version). Also, only half the amount of processors is still necessary, since at any moment half of the cells receive only void signals. Then, the signal flow graph is cut along the dashed lines in figure 8.3, adding delays on lines crossing the cut in the upward direction and removing delays on lines crossing the cut in the downward direction. Finally, the fully pipelined array (horizontally and vertically) is obtained by also cutting along the dotted lines in figure 8.3 and adding delays on lines crossing the cuts from left to right or from top to bottom. The obtained systolic algorithm has a throughput, independent of the problem size, at the expense of a more complicated array. The correctness has been verified experimentally.

Finally, note that in most real situations one can not afford to map single cells but only rectangular subarrays onto different processors. It is clear that the above approach can be applied to introduce only those algorithmic corrections needed for the pipelining at the processor level. Also note that due to the extra triangular part, the number of operations per discrete-time step is of order $O(n^2)$ instead of $O(nm)$. However, the additional operations are so cheap that in the triangular part more cells (or even all cells of a couple of rows) could be mapped onto one processor than in the rectangular part, reducing again this extra cost.

8.3 An LU based algorithm

In this section we briefly consider a different fully pipelined array for algorithm D. If A is square, the algorithm stores $L = \mathrm{lf}(A)$ and $V = \mathrm{iuf}(A)$ instead of A itself. For a rectangular matrix $A \in \mathbb{R}^{n\times m}$, one can think of A as part of a square matrix $\bar{A} \in \mathbb{R}^{n\times n}$. If $\bar{L} = \mathrm{lf}(\bar{A})$ and $\bar{V} = \mathrm{iuf}(\bar{A})$, the new algorithm stores $L = \bar{L}_{\cdot,\leqslant m}$ and $V = \bar{V}_{\leqslant m,\leqslant m}$. Note that again this implies $A = LU$.

A problem with the LU based algorithm is that the LU decomposition of A is assumed to exist, which is not always the case (see definition 3.1). We do not know whether applications exist where the existence of an LU decomposition of A can be guaranteed. A motivation to include this example, is that it is *another illustration of the complete way from continuous-time algorithm to systolic array.* In this section we give a description of the algorithms. In appendix 8.A, we briefly discuss the derivation of the signal flow graphs.

With lemma 3.4 one can derive different continuous-time algorithms storing $L = \mathrm{lf}(A)$ or $V = \mathrm{ilf}(A)$, and $\mathrm{uf}(A)$ or $\mathrm{iuf}(A)$. With the theory of chapter 5, one can derive parallel signal flow graphs for the continuous-time algorithm. Again signal flow graphs for the discrete-time algorithm show roughly the same information flow. The version storing $L = \mathrm{lf}(A)$ and $V = \mathrm{iuf}(A)$, has bidirectional information flow only in the vertical direction, and can be fully pipelined with a variant of the strategy of [80, 90] as above. The same extra triangle is

necessary as for algorithm D. (Appendix 8.A)

Application of lemma 3.4 to algorithm C, yields

Algorithm 8.4

$$\begin{aligned} \dot{L} &= L\,\mathsf{lows}(L^{-1}xx^TL) \\ \dot{V} &= -V\,\mathsf{upp}(L^{-1}xx^TL) + 2\,\mathsf{upph}(V^{-T}L^Txx^TLV^{-1})V \end{aligned}$$

The equation for $\dot{L}$ is an algorithm of type III (definition 5.1). If we consider $w = L^{-1}x$ and $y = L^Tx$ as inputs coming from the part tracking L, the equation for $\dot{V}$ has a first term of type II and a second term of type III. The resulting signal flow graph is shown in figures 8.7 and 8.10.

The corresponding discrete-time algorithm is given by

Algorithm 8.5

$$\begin{aligned} y &= L^Tx, \\ w &= L^{-1}x, \\ z &= V^{-T}y, \\ \beta &= 2\alpha + \alpha^2x^Tx, \\ L' &= L\,\mathsf{lf}(I + \alpha wy^T), \\ V' &= \mathsf{cf}(I + \beta zz^T)V\,\mathsf{iuf}(I + \alpha wy^T), \end{aligned}$$

where α is a free parameter as in algorithm D.

Note that z in this algorithm corresponds to y in algorithm D. The signal flow graph of the unpipelined algorithm is shown in figures 8.9 and 8.10. The correspondence with the continuous-time algorithm can be shown in a similar way as for algorithms C and D (theorem 8.1), or one can check that $A = LV^{-1}$ satisfies algorithm D. The unpipelined signal flow graph is shown in figures 8.9 and 8.10. The pipelined signal flow graph is shown in figures 8.11, 8.12 and 8.13.

8.4 Comparison with related algorithms

In this section we compare algorithm D with other related discrete-time algorithms. As far as performance is concerned, algorithm D is not a new algorithm. Therefore, it is not our aim to present an elaborate discussion of the performance and for more numerical experiments we refer to [21], [107], and the many references in [85].

We first summarize the main properties, distinguishing algorithm D from other algorithms. Then we give a comparison with different algorithms. Meanwhile, we also show how by an appropriate choice of $\alpha^{[k]}$, algorithm D tracks exactly the same subspaces as the original spherical subspace tracker.

Before comparing algorithm D with other algorithms, the following theorem summarizes its main properties, inherited from the continuous-time algorithm C.

Theorem 8.6
(i) Algorithm D has a cellular signal flow graph (figures 8.1 and 8.2), that can be the starting point for different parallel realizations. Realizations with problem size independent throughput can be obtained.

(ii) The number of operations per discrete-time step is of the order $O(nm)$ for the unpipelined or horizontally pipelined version. The vertically pipelined version has an additional number of operations of order $O(n^2)$ (but cheap).

(iii) When initialized with an orthogonal matrix, A stays orthogonal. When initialized with a nonorthogonal full rank matrix, A converges to orthogonal, if x does not converge to a subspace of $\mathbb{R}^n$ (in the sense of remark 7.39).

(iv) Each submatrix $A_{:,\leqslant m}$, $m = 1, \ldots, n$ tracks the discrete-time cross product subspace $\mathcal{C}_m^{[k]} = \mathcal{C}_m(kT_s) = \mathrm{col}(B^{[k]})$ where $B^{[k]} \in \mathbb{R}^{n \times m}$ is given by

$$B^{[k+1]} = B^{[k]} + \alpha^{[k]} x^{[k]} x^{[k]T} B^{[k]},$$

with $B^{[0]} = A^{[0]}_{:,\leqslant m}$.

(v) The canonical angles between $\mathrm{col}(A)$ and $\mathrm{col}(M)$ follow (7.7), with piecewise constant x (interpreting the derivatives as left or right derivatives at points of discontinuous x).

(vi) For the noise free case with a stationary subspace, $\mathrm{col}(A)$ converges to $\mathrm{col}(M)$, if x does not converge to a subspace of $\mathrm{col}(A)$ (in the sense of interpretation 7.34).

Proof:
Properties (i) and (ii) follow from the derivation of the signal flow graph above. Properties (iii), (iv), (v) and (vi) are inherited from theorems 7.37 and 7.29 and corollary 7.33 for the continuous-time algorithm C through theorem 8.1.
For properties (iii),(v) and (vi), note that theorems 7.37 and 7.29 and corollary 7.33 for the continuous-time case actually assume continuously differentiable $x(t)$. However this can be extended to the piecewise constant case by concatenating trajectories, for the different intervals. Since $A(t)$ is still continuous, also the singular values (of A or $M^T A$) are continuous, and the derivatives of singular values and canonical angles should simply be considered as left and right derivatives at the concatenation points.
□

As in interpretation 7.35, property (iv) again implies that the columns of A estimate the eigenvectors of $E\{xx^T\}/(1-\rho)$. Also the eigenvalues of $E\{xx^T\}$ can be estimated by

$$\lambda_i^{[k+1]} = \rho\lambda_i^{[k]} + (A_{\cdot i}^{[k]T} x^{[k]})^2. \tag{8.2}$$

8.4.1 Comparison with stochastic gradient algorithms

Discrete-time stochastic gradient algorithms are often cast as ([86, 107])

$$A^{[k+1]} = \text{orth}(A^{[k]} + \alpha^{[k]} x^{[k]} x^{[k]T} A^{[k]}), \tag{8.3}$$

where $A^{[k]} \in \mathbb{R}^{n\times m}$ and orth represents some orthogonalization algorithm.

This algorithm tracks exactly the same subspace $\text{col}(A^{[k]})$ as algorithm D. If orth = qf one obtains even the same estimations $A^{[k]}$ as algorithm D, as follows from interpretation 7.21. In [107] a systolic implementation of such an algorithm is given. However the orthogonalization needs $O(nm^2)$ operations [48], and the throughput is $O((2n+m+1)^{-1})$ which is worse than both the linear and the fully pipelined systolic arrays given above.

Oja's neural stochastic gradient algorithm (algorithm 7.16) was derived by replacing orth in (8.3) by a first order approximation, with computational cost $O(nm)$ instead of $O(nm^2)$. Algorithm D obtains *exact* reorthogonalization at the same cost $O(nm)$. For the neural stochastic gradient algorithm, attraction to an orthogonal matrix, starting from any full rank initial condition, is proved statistically assuming the conditions (7.11). For algorithm D, the convergence results, inherited from theorems 7.29 and 7.37 are valid for arbitrary large step sizes. Finally, for Oja's algorithm, no systolic implementations with problem size independent throughput are known.

8.4.2 Comparison with the spherical subspace tracker

Theorem 8.7
The original spherical subspace tracker with forgetting factor ρ yields the same subspace estimates as algorithm D (algorithm 8.3), if

$$\alpha^{[k]} = \frac{1}{\rho\lambda_{(s)}{}^{[k]} - \lambda_{(n)}{}^{[k+1]}}, \tag{8.4}$$

where the values of $\lambda_{(s)}{}^{[k+1]}$ and $\lambda_{(n)}{}^{[k+1]}$ (as in the original spherical subspace tracker) are the eigenvalues of

$$\begin{bmatrix} \rho\lambda_{(s)}{}^{[k]} & 0 \\ 0 & \rho\lambda_{(n)}{}^{[k]} \end{bmatrix} + \begin{bmatrix} c_{(s)}{}^{[k]} \\ c_{(n)}{}^{[k]} \end{bmatrix} \begin{bmatrix} c_{(s)}{}^{[k]} & c_{(n)}{}^{[k]} \end{bmatrix},$$

with ${c_{(s)}}^{[k]} = \sqrt{\sum_{i=1}^{m} {y_i}^{[k]2}}$ and ${c_{(n)}}^{[k]} = \sqrt{\sum_{i=m+1}^{n} {y_i}^{[k]2}}$.
These eigenvalues can be obtained by

$$\begin{array}{ll} {\sigma_{(s)}}^{[k]} = \rho {\lambda_{(s)}}^{[k]} + {c_{(s)}}^{[k]2} & {\sigma_{(n)}}^{[k]} = \rho {\lambda_{(n)}}^{[k]} + {c_{(n)}}^{[k]2}, \\ D^{[k]} = ({\sigma_{(s)}}^{[k]} - {\sigma_{(n)}}^{[k]})^2 + 4{c_{(s)}}^{[k]2}{c_{(n)}}^{[k]2}, & \\ {\lambda_{(s)}}^{[k+1]} = \frac{1}{2}({\sigma_{(s)}}^{[k]} + {\sigma_{(n)}}^{[k]} + \sqrt{D^{[k]}}) & {\lambda_{(n)}}^{[k+1]} = \frac{1}{2}({\sigma_{(s)}}^{[k]} + {\sigma_{(n)}}^{[k]} - \sqrt{D^{[k]}}). \end{array}$$

Proof:
From lemma 7.11(ii) we may represent the subspace, estimated by algorithm D, by the same basis vectors as for the original spherical subspace tracker. In this representation both algorithms reduce to applying one Givens rotation to two columns of A. For the spherical subspace tracker this rotation is found from the diagonalization (EVD) of

$$\begin{bmatrix} {\sigma_{(s)}}^{[k]} & {c_{(s)}}^{[k]}{c_{(n)}}^{[k]} \\ {c_{(s)}}^{[k]}{c_{(n)}}^{[k]} & {\sigma_{(n)}}^{[k]} \end{bmatrix},$$

while for algorithm D it is found from the triangularization (QR factorization) of

$$\begin{bmatrix} 1 + \alpha^{[k]}{c_{(s)}}^{[k]2} & \alpha^{[k]}{c_{(s)}}^{[k]}{c_{(n)}}^{[k]} \\ \alpha^{[k]}{c_{(s)}}^{[k]}{c_{(n)}}^{[k]} & 1 + \alpha^{[k]}{c_{(n)}}^{[k]2} \end{bmatrix}.$$

For the spherical subspace tracker one finds a rotation over an angle θ_S, satisfying

$$\tan(2\theta_S) = \frac{2{c_{(s)}}^{[k]}{c_{(n)}}^{[k]}}{{\sigma_{(s)}}^{[k]} - {\sigma_{(n)}}^{[k]}}. \tag{8.5}$$

For algorithm D one finds an angle θ_A satisfying

$$\tan(\theta_A) = \frac{\alpha^{[k]}{c_{(s)}}^{[k]}{c_{(n)}}^{[k]}}{1 + \alpha^{[k]}{c_{(s)}}^{[k]2}}.$$

Identifying both angles, one finds after some calculations that

$$\alpha^{[k]} = \frac{1}{\rho {\lambda_{(s)}}^{[k]} - {\lambda_{(n)}}^{[k+1]}}.$$

□
This result was checked experimentally.

Although the subspace estimates of both algorithms are the same (with $\alpha^{[k]}$ as in theorem 8.7), the representation by the columns of A is different, as in the original spherical subspace tracker one of the vectors spanning the signal (noise) subspace is always parallel to the projection of the current input vector on the subspace. As a consequence, the original spherical subspace tracker keeps changing its representation, and even small (noisy) input can imply big changes. Also, the columns of A cannot be interpreted as estimates of the eigenvectors.

As this implies that also the eigenvalues are not estimated, the dimension κ of the space to be tracked is to be known in advance. This is a disadvantage of the implementation as a spherical subspace tracker. In [22, 41] DeGroat and Dowling have shown that this problem can be overcome by tracking more than two subspaces. However the algorithm gets more complex, and no realizations with problem size independent throughput exist.

For the original spherical subspace tracker, the updates are performed by explicitly applying orthogonal transformations to the columns of A. Numerical deviations from orthonormality are not restored as for algorithm D. Therefore an extra reorthogonalization scheme is needed.

Recently, a new systolic array for spherical subspace tracking was proposed by Vanpoucke [99]. This array also achieves a problem size independent throughput. The orthogonal matrix is stored by a minimal parametrization, such that no deviations from orthogonality are possible. The only disadvantages left are that the eigenvectors and eigenvalues are not estimated, and that no other step size strategies can be implemented.

8.4.3 Step sizes

Since the derivation of the spherical subspace tracker contains a heuristic approximation, and as the presence of the forgetting factor leaves a heuristic element in equation (8.4) for the step size, it is not worth to implement this step size exactly.

Since all values $y_1^{[k]}, \ldots, y_m^{[k]}$ are needed to calculate $\alpha^{[k]}$ as in (8.4) (note that ${c_{(n)}}^2$ can be calculated as ${c_{(n)}}^2 = x^T x - {c_{(s)}}^2 = \delta_{0,1} - {c_{(s)}}^2$, avoiding the need for $y_{m+1}, \ldots, y_n$), it is impossible to calculate $\alpha^{[k]}$ in time with the horizontally pipelined realization given above. However, the step size (8.4) and other step size strategies making use of eigenvalue information (8.2) can be implemented with throughput $O(m^{-1})$, and approximately with throughput $O(1)$. Throughput $O(m^{-1})$ is obtained by keeping the vertical pipelining, and only pipelining the triangular part (cells $[i,j]$) both horizontally and vertically. This is achieved by simply omitting the final vertical cuts in the derivation of the signal flow graph of figures 8.5 and 8.6. Each row of cells (i,j) and $< i,j >$ can then be mapped onto one processor. The obtained throughput $O(m^{-1})$ is for most applications significantly better than $O(n^{-1})$.

Throughput $O(1)$ can only be obtained by using older versions of $y_2, \ldots, y_m$ to estimate the eigenvalues. Figure 8.14 shows that this only introduces a small error, although of course the effect increases with growing m (or with improved wanted throughput).

However, since the derivation of the spherical subspace tracker contains a heuris-

tic approximation and the presence of the forgetting factor leaves a heuristic element in equation 8.4, implementing (8.4) exactly is not worth the effort. Moreover step size (8.4) is not proved to be optimal in any way, as the spherical subspace tracker was derived as an approximate EVD update, and its step size is designed to update the EVD after averaging, assuming the averaged result is good enough, and not attempting to correct the approximation. Experiments show that the resulting algorithm gives acceptable results [21], but worse than more complex algorithms like Jacobi SVD updating [76, 79] as was observed by [99]. The identification between spherical subspace tracking and algorithm D, discussed above (theorem 8.7), allows one to compare the spherical subspace tracker with algorithms with other step size strategies. The experiments shown in figure 8.15, show that there is room for improvement. These experiments were done with

$$\alpha = K/z, \text{ where } z^{[k+1]} = \rho z^{[k]} + x^{[k]T} x^{[k]}, \tag{8.6}$$

(yielding $z^{[k]} = \sigma_{(n)}^{[k]} + \sigma_{(s)}^{[k]} = \sum_{i=1}^{n} \lambda_i^{[k]}$, with $\lambda^{[k]}$ as in (8.2)), and with different values of $K = 1$,$K = 10$ and $K = 20$. For experiments with less noise even more improvements can be made. In the noiseless case, one may take $\alpha^{[k]}$ arbitrarily large, as follows for instance from corollary 7.33. For $\alpha \to \infty$, the subspace estimate is rotated (with a plane rotation) until it contains the current input x. Note that these step sizes (8.6) can be realized with throughput O(1) without any approximation as $x^T x = \delta_{0,1}$.

The search for a good step size strategy is not a closed issue, but a further invesigation falls outside the scope of this work. Analyzing strategies is not straightforward and can be highly application dependent, as many different conditions can occur for signal-to-noise ratios, time correlation of the signals, spread of the signal eigenvalues, and the time variation of all these conditions. A main feature to borrow from the spherical subspace step size (8.4) is that it is inverse proportional to the input signal x. This implies that scaling x (over all time samples) does not affect the subspace estimates. Only this feature was kept in the step sizes (8.6). We hope however that the statiscal analysis, initiated at the end of the previous chapter, may form the basis for new and well motivated step size strategies.

8.5 Conclusion

We have derived a discrete-time version of the continuous-time algorithm C, by integration for a piecewise constant input. The new algorithm, algorithm D, has a signal flow graph, similar to the continuous-time algorithm, serving as the starting point for the derivation of different pipelined versions. The number of operations per iteration step is of order $O(mn)$. Full pipelining, with problem size independent throughput can be obtained at the expense of a complication of the array, and an additional number of order $O(n^2)$ (cheap) operations.

The algorithm inherits the main properties of the continuous-time algorithm, analyzed in chapter 7, including the attraction to orthogonality and the tracking analysis. The main advantages over other stochastic gradient algorithms are the cheap orthogonalization (exact reorthogonalization, when starting from an orthogonal matrix), and the possibility of high throughput realizations. An advantage over Oja's neural stochastic gradient algorithm is also that arbitrarily large step sizes are possible. With an appropriately chosen step size, the new algorithm yields the same subspace estimates as the original spherical subspace tracking algorithm. Experiments show that there is room for improvement by other step size strategies.

Although there is no automatic method for deriving systolic algorithms, the example of another LU based version of algorithm D illustrates that more systolic algorithms can be obtained from continuous-time algorithms of the class of definition 5.1. We briefly come back to the issue of generalization in chapter 12.

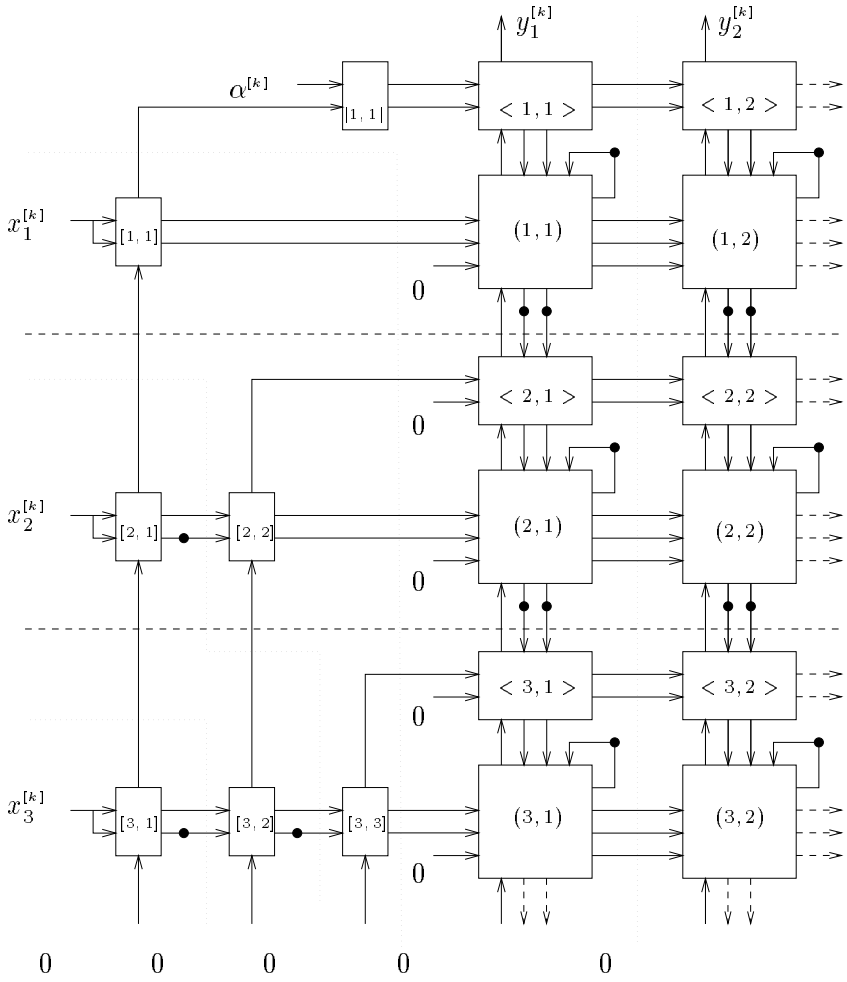
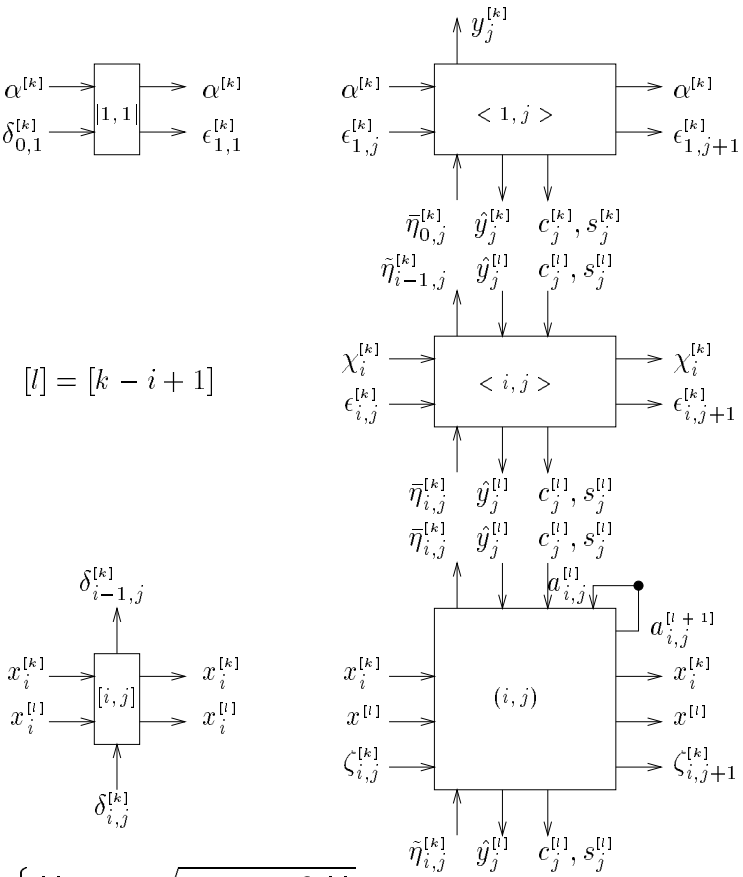


Figure 8.3: Signal flow graph for the discrete-time algorithm D, with delays on lines going downwards, corrections on lines going upwards and extra triangular array. Global picture, input, output and boundary conditions.



$$|1,1|: \quad \left\{ \epsilon_{1,1}^{[k]} = -1/\sqrt{2\alpha^{[k]} + \alpha^{[k]2}\delta_{0,1}^{[k]}} \right.$$

$$< 1,j >: \quad \left\{ \begin{array}{l} y_j^{[k]} = \bar{\eta}_{0,j}^{[k]} \\ \hat{y}_j^{[k]} = \alpha^{[k]} y_j^{[k]} \\ c_j^{[k]}, s_j^{[k]},\ \epsilon_{1,j+1}^{[k]} \text{ from } {c_j^{[k]}}^2 + {s_j^{[k]}}^2 = 1, \text{ and} \\ [y_j^{[k]} \ \ \epsilon_{1,j}^{[k]}] \begin{bmatrix} c_j^{[k]} & s_j^{[k]} \\ -s_j^{[k]} & c_j^{[k]} \end{bmatrix} = [0 \ \ \epsilon_{1,j+1}^{[k]}] \end{array} \right.$$

$$< i,j >: \quad \left\{ [\tilde{\eta}_{i-1,j}^{[k]} \ \ \epsilon_{i,j+1}^{[k]}] = [\bar{\eta}_{i,j}^{[k]} + \chi_i^{[k]} \hat{y}_j^{[l]} \ \ \epsilon_{i,j}^{[k]}] \begin{bmatrix} c_j^{[l]} & s_j^{[l]} \\ -s_j^{[l]} & c_j^{[l]} \end{bmatrix} \right.$$

$$[i,j]: \quad \left\{ \delta_{i-1,j}^{[k]} = \delta_{i,j}^{[k]} + x_i^{[k]} x_i^{[l]} \right.$$

$$(i,j): \quad \left\{ \begin{array}{l} \bar{\eta}_{i,j}^{[k]} = \tilde{\eta}_{i,j}^{[k]} + x_i^{[k]} a_{i,j}^{[l]} \\ [a_{i,j}^{[l+1]} \ \ \zeta_{i,j+1}^{[k]}] = [a_{i,j}^{[l]} + x^{[l]} \hat{y}_j^{[l]} \ \ \zeta_{i,j}^{[k]}] \begin{bmatrix} c_j^{[l]} & s_j^{[l]} \\ -s_j^{[l]} & c_j^{[l]} \end{bmatrix} \end{array} \right.$$

Figure 8.4: Signal flow graph for the discrete-time algorithm D, with delays on lines going downwards, corrections on lines going upwards and extra triangular array. Cells $|1,1|$, $< 1,j >$, $< i,j >$, $[i,j]$ and (i,j).

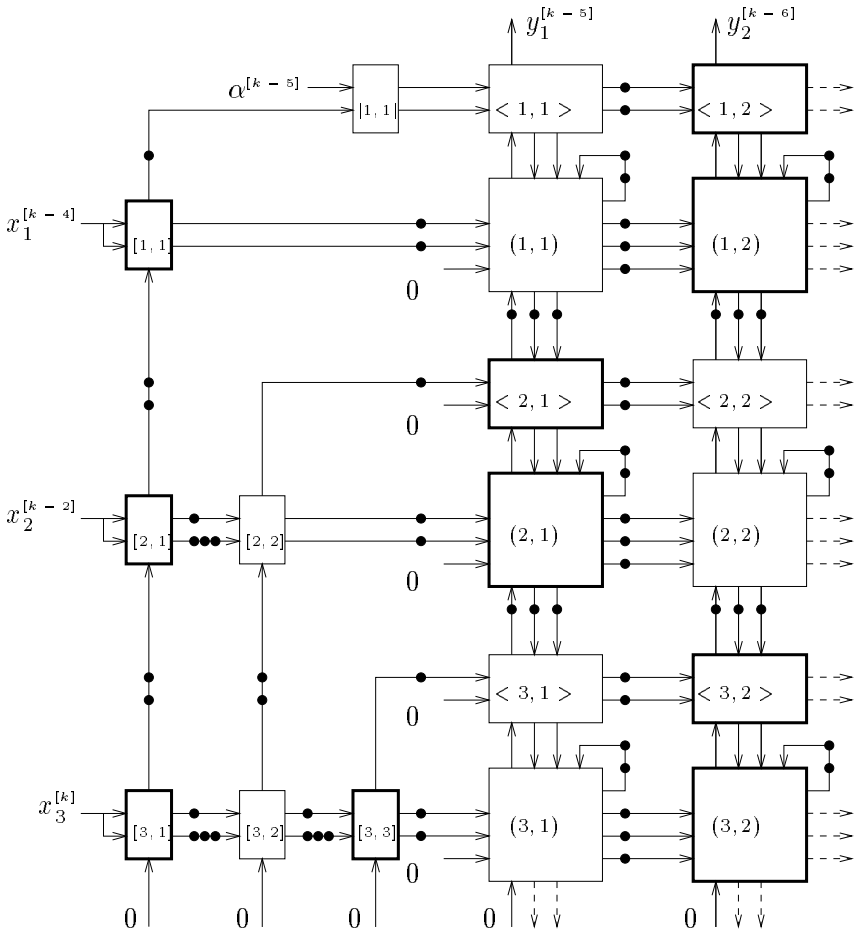
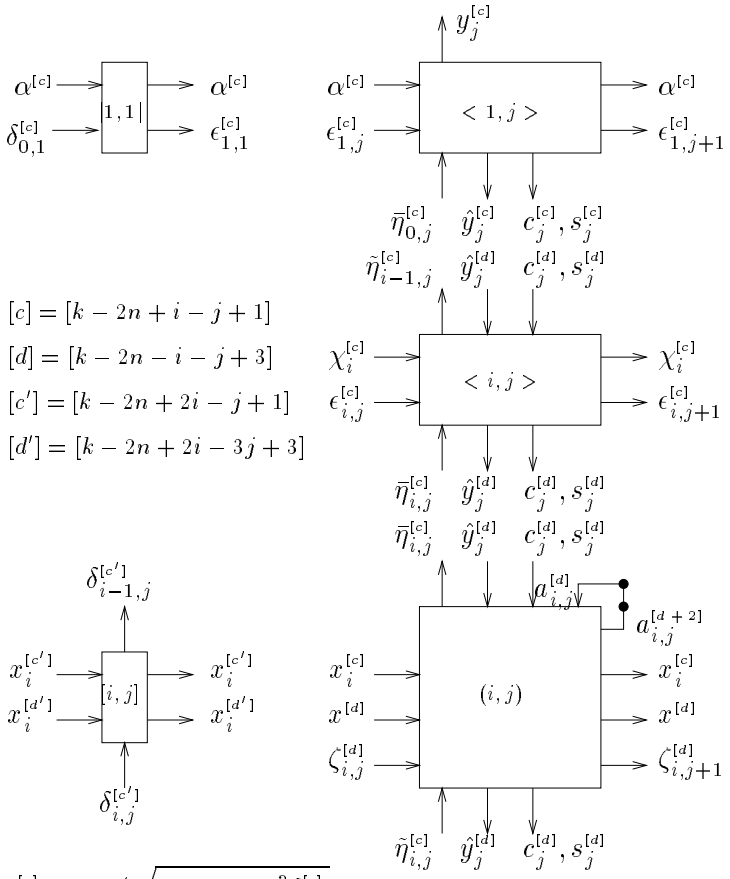


Figure 8.5: Signal flow graph for the fully pipelined discrete-time algorithm D. Global picture, input, output and boundary conditions. Thick cells are active at even times. Thin cells are active at odd times.



$|1,1|$: $\epsilon_{1,1}^{[c]} = -1/\sqrt{2\alpha^{[c]} + \alpha^{[c]2}\delta_{0,1}^{[c]}}$

$< 1,j >$:
$$\begin{cases} y_j^{[c]} = \bar{\eta}_{0,j}^{[c]} \\ \hat{y}_j^{[c]} = \alpha^{[c]} y_j^{[c]} \\ c_j^{[c]}, s_j^{[c]}, \epsilon_{1,j+1}^{[c]} \text{ from } {c_j^{[c]}}^2 + {s_j^{[c]}}^2 = 1, \text{ and} \\ [y_j^{[c]} \;\; \epsilon_{1,j}^{[c]}] \begin{bmatrix} c_j^{[c]} & s_j^{[c]} \\ -s_j^{[c]} & c_j^{[c]} \end{bmatrix} = [0 \;\; \epsilon_{1,j+1}^{[c]}] \end{cases}$$

$< i,j >$: $[\tilde{\eta}_{i-1,j}^{[c]} \;\; \epsilon_{i,j+1}^{[c]}] = [\bar{\eta}_{i,j}^{[c]} + \chi_i^{[c]} \hat{y}_j^{[d]} \;\; \epsilon_{i,j}^{[c]}] \begin{bmatrix} c_j^{[d]} & s_j^{[d]} \\ -s_j^{[d]} & c_j^{[d]} \end{bmatrix}$

$[i,j]$: $\delta_{i-1,j}^{[c']} = \delta_{i,j}^{[c']} + x_i^{[c']} x_i^{[d']}$

(i,j) :
$$\begin{cases} \bar{\eta}_{i,j}^{[c]} = \tilde{\eta}_{i,j}^{[c]} + x_i^{[c]} a_{i,j}^{[d]} \\ [a_{i,j}^{[d+2]} \;\; \zeta_{i,j+1}^{[d]}] = [a_{i,j}^{[d]} + x^{[d]} \hat{y}_j^{[d]} \;\; \zeta_{i,j}^{[d]}] \begin{bmatrix} c_j^{[d]} & s_j^{[d]} \\ -s_j^{[d]} & c_j^{[d]} \end{bmatrix} \end{cases}$$

Figure 8.6: Signal flow graph for the fully pipelined discrete-time algorithm D. Cells $|1,1|$, $< 1,j >$, $< i,j >$, $[i,j]$ and (i,j).

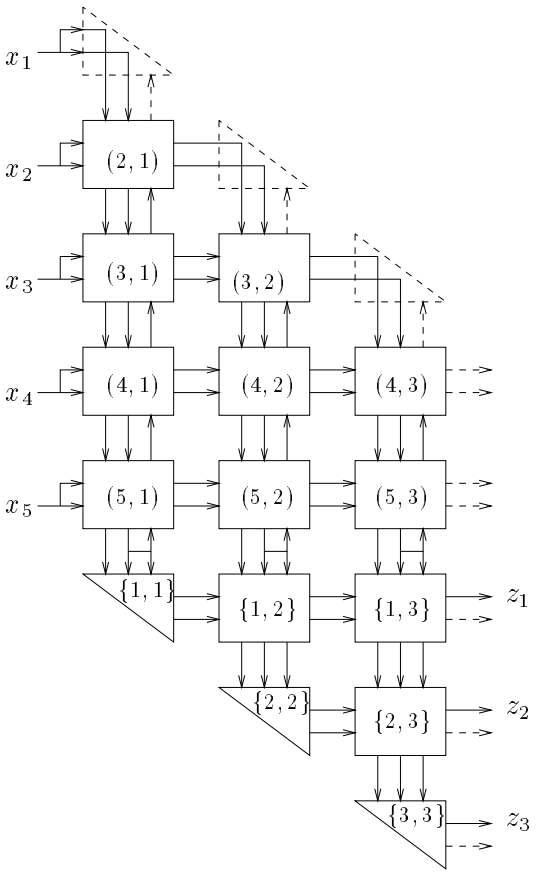
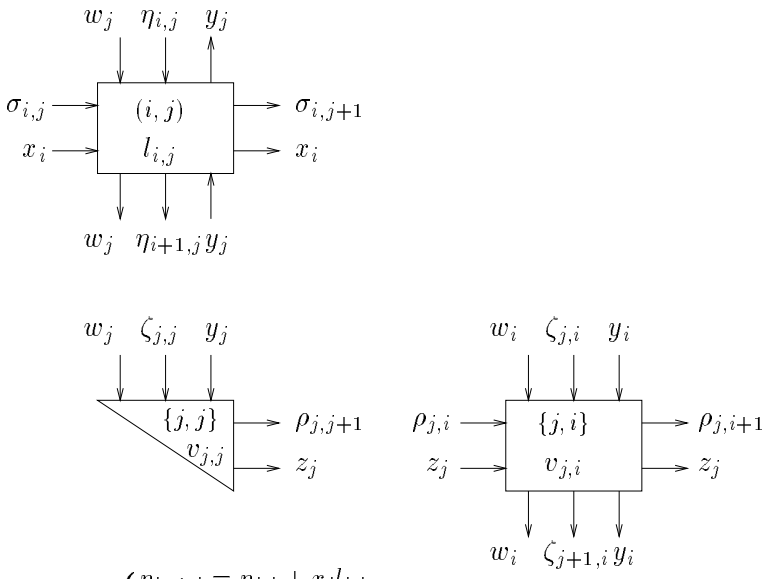


Figure 8.7: Signal flow graph for the LU based version of algorithm C. Global picture, input, output and boundary conditions.



$$
(i,j): \begin{cases} \eta_{i+1,j} = \eta_{i,j} + x_i l_{i,j} \\ \sigma_{i,j+1} = \sigma_{i,j} - l_{i,j} w_j \\ \dot{l}_{i,j} = \sigma_{i+1,j} y_j \end{cases}
$$

$$
\{j,j\}: \begin{cases} \rho_{j,j+1} = v_{j,j} w_j \\ z_j = \zeta_j / v_{j,j} \\ \dot{v}_{j,j} = -\rho_{j,j+1} y_j + z_j \zeta_{j,j} \end{cases}
$$

$$
\{j,i\}: \begin{cases} \zeta_{j+1,i} = \zeta_{j,i} - z_j v_{j,i} \\ \rho_{j,i+1} = \rho_{j,i} + v_{j,i} w_i \\ \dot{v}_{j,i} = -\rho_{j,i+1} y_i + z_j \zeta_{j,i} + z_j \zeta_{j+1,i} \end{cases}
$$

Figure 8.8: Signal flow graph for the LU based version of algorithm C. Cells (i,j), $\{j,j\}$ and $\{j,i\}$.

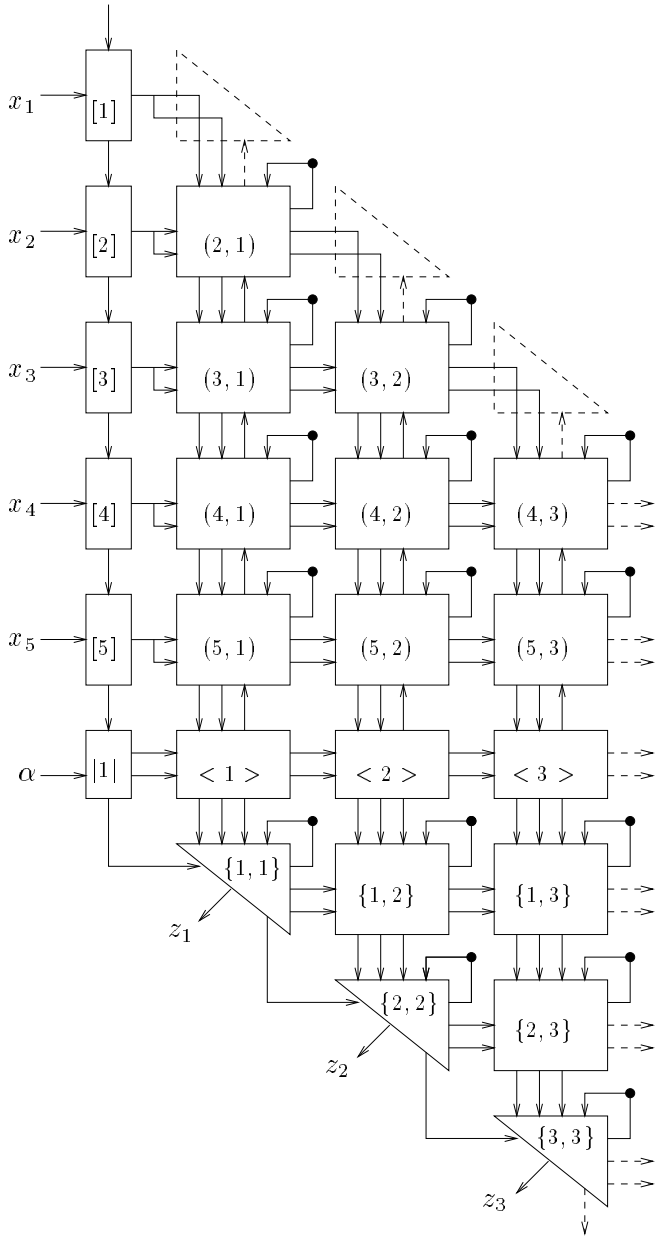


Figure 8.9: Signal flow graph for the unpipelined LU based version of algorithm D. Global picture, input, output and boundary conditions.

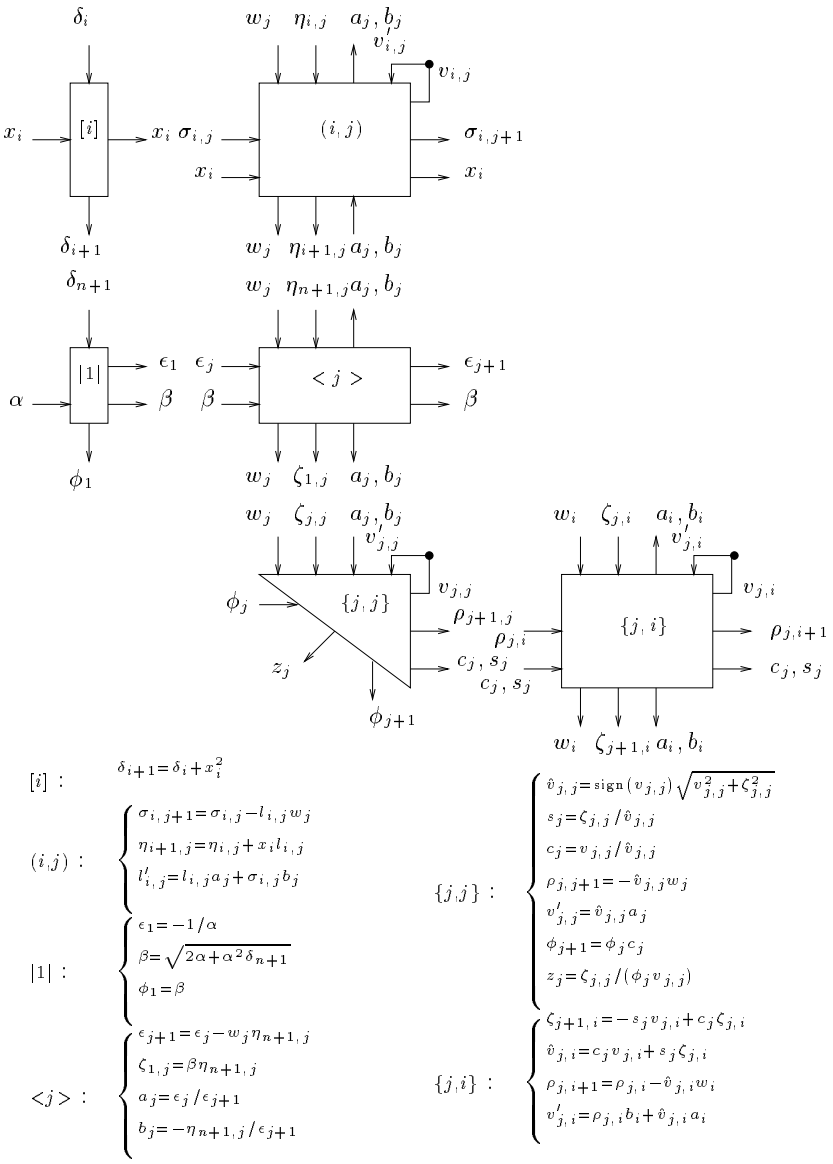


Figure 8.10: Signal flow graph for the unpipelined LU based version of algorithm D. Cells $[i]$, (i,j), $|1|$, $< j >$, $\{j,j\}$ and $\{j,i\}$.

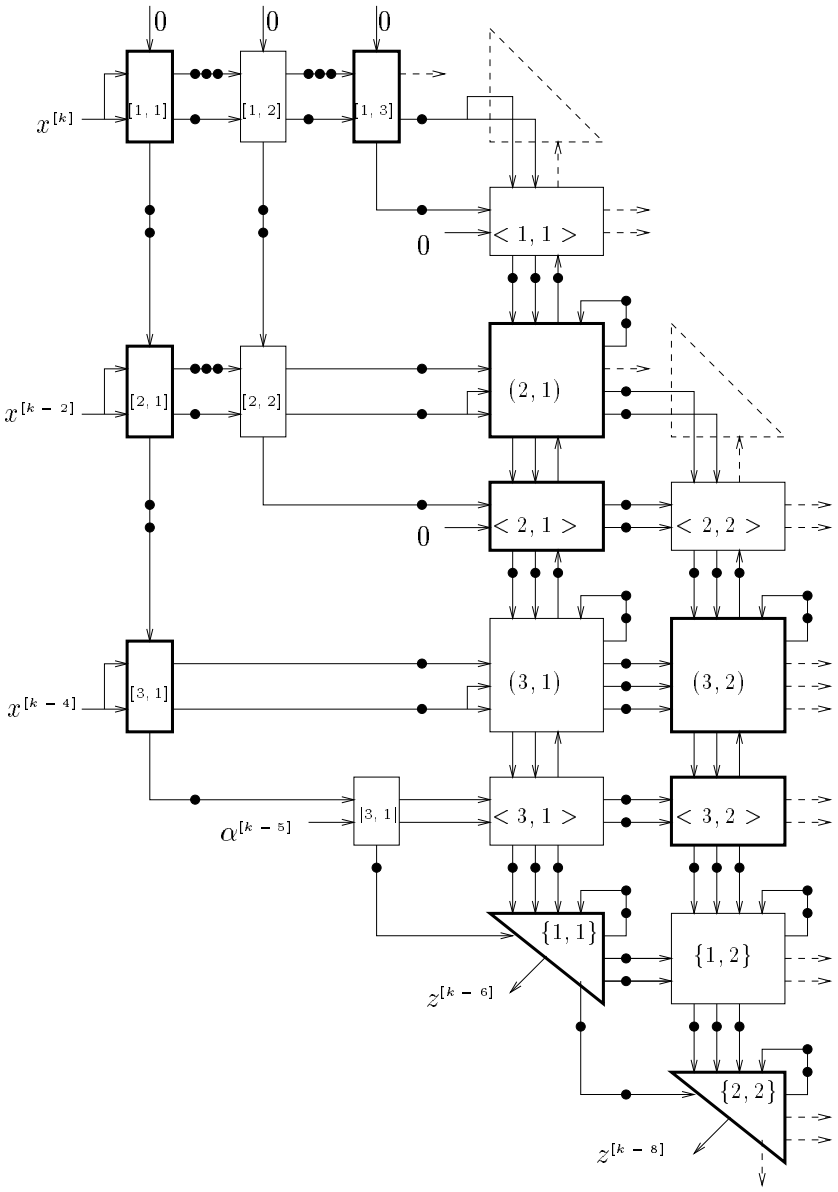


Figure 8.11: Signal flow graph for the pipelined LU based version of algorithm D. Global picture, input, output and boundary conditions. Thick cells are active at even times. Thin cells are active at odd times.

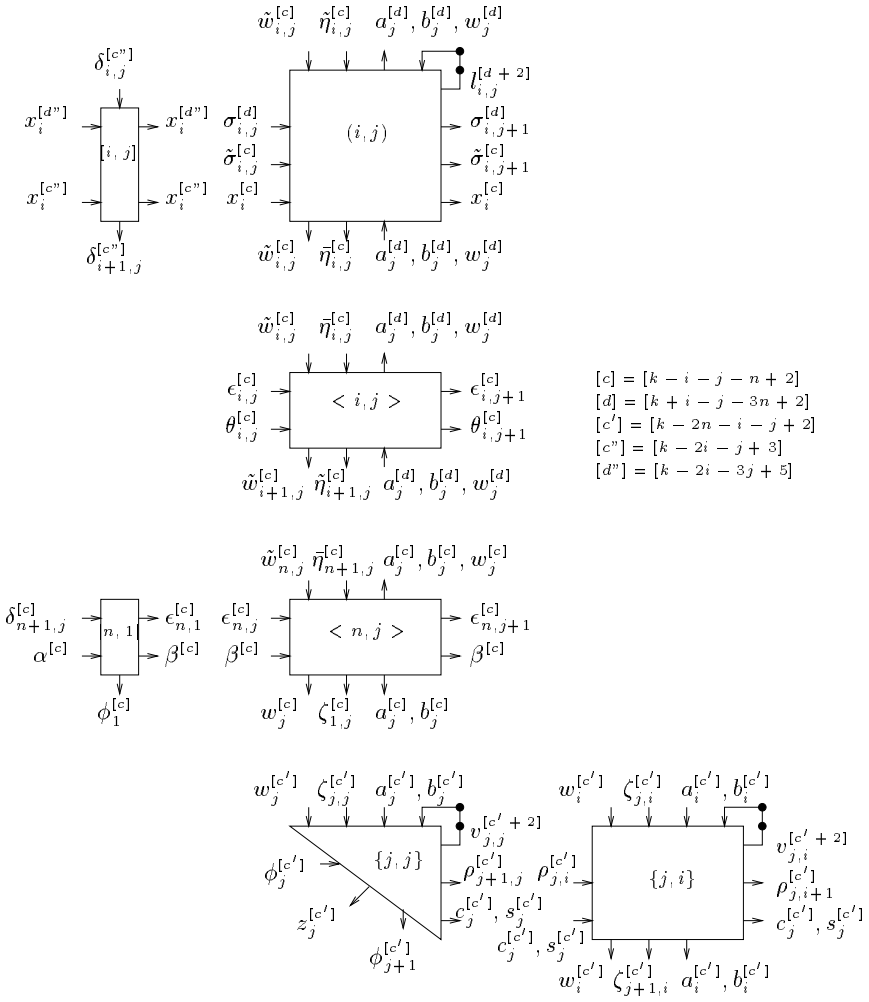


Figure 8.12: Signal flow graph for the pipelined LU based version of algorithm D. Cells $[i,j]$, (i,j), $< i,j >$, $< n,j >$, $|n,1|$, $\{j,j\}$ and $\{j,i\}$. Also see figure 8.13.

$[i,j]$:

$$\delta_{i+1,j}^{[c'']} = \delta_{i,j}^{[c'']} + x_i^{[d'']} x_i^{[c'']}$$

(i,j) :

$$\begin{cases} \sigma_{i,j+1}^{[d]} &= \sigma_{i,j}^{[d]} - l_{i,j}^{[d]} w_j^{[d]} \\ \tilde{\sigma}_{i,j+1}^{[c]} &= \tilde{\sigma}_{i,j}^{[c]} - l_{i,j}^{[d]} \tilde{w}_{i,j}^{[c]} \\ \bar{\eta}_{i,j}^{[c]} &= \tilde{\eta}_{i,j}^{[c]} + l_{i,j}^{[d]} x_i^{[c]} \\ l_{i,j}^{[d+2]} &= a_j^{[d]} l_{i,j}^{[d]} + b_j^{[d]} \sigma_{i,j}^{[d]} \end{cases}$$

$< i,j >$:

$$\begin{cases} \epsilon_{i,j+1}^{[c]} &= \epsilon_{i,j}^{[c]} - \bar{\eta}_{i,j}^{[c]} w_j^{[d]} \\ \tilde{\eta}_{i+1,j}^{[c]} &= \bar{\eta}_{i,j}^{[c]} a_j^{[d]} + \epsilon_{i,j}^{[c]} b_j^{[d]} \\ \theta_{i,j+1}^{[c]} &= \theta_{i,j}^{[c]} a_j^{[d]} - \tilde{w}_{i,j}^{[c]} b_j^{[d]} \\ \tilde{w}_{i+1,j}^{[c]} &= \tilde{w}_{i,j}^{[c]} + \theta_{i,j}^{[c]} w_j^{[d]} \end{cases}$$

$< n,j >$:

$$\begin{cases} w_j^{[c]} &= \tilde{w}_{n,j}^{[c]} \\ \epsilon_{n,j+1}^{[c]} &= \epsilon_{n,j}^{[c]} - w_j^{[c]} \bar{\eta}_{n+1,j}^{[c]} \\ \zeta_{1,j}^{[c]} &= \beta^{[c]} \bar{\eta}_{n+1,j}^{[c]} \\ a_j^{[c]} &= \epsilon_{n,j}^{[c]} / \epsilon_{n,j+1}^{[c]} \\ b_j^{[c]} &= -\bar{\eta}_{n+1,j}^{[c]} / \epsilon_{n,j+1}^{[c]} \end{cases}$$

$|n,1|$:

$$\begin{cases} \epsilon_{n,1}^{[c]} &= -1/\alpha^{[c]} \\ \beta^{[c]} &= \sqrt{2\alpha^{[c]} + \alpha^{[c]2} \delta_{n+1,j}^{[c]}} \\ \phi_1^{[c]} &= \beta^{[c]} \end{cases}$$

$\{j,j\}$:

$$\begin{cases} \hat{v}_{j,j}^{[c']} &= \mathrm{sign}(v_{j,j}^{[c']}) \sqrt{v_{j,j}^{[c']2} + \zeta_{j,j}^{[c']2}} \\ s_j^{[c']} &= \zeta_{j,j}^{[c']} / \hat{v}_{j,j}^{[c']} \\ c_j^{[c']} &= v_{j,j}^{[c']} / \hat{v}_{j,j}^{[c']} \\ \rho_{j+1,j}^{[c']} &= -\hat{v}_{j,j}^{[c']} w_j^{[c']} \\ v_{j,j}^{[c'+2]} &= \hat{v}_{j,j}^{[c']} a_j^{[c']} \\ \phi_{j+1}^{[c']} &= \phi_j^{[c']} c_j^{[c']} \\ z_j^{[c']} &= \zeta_{j,j}^{[c']} / (\phi_j^{[c']} v_{j,j}^{[c']}) \end{cases}$$

$\{j,i\}$:

$$\begin{cases} \zeta_{j+1,i}^{[c']} &= -s_j^{[c']} v_{j,i}^{[c']} + c_j^{[c']} \zeta_{j,i}^{[c']} \\ \hat{v}_{j,i}^{[c']} &= c_j^{[c']} v_{j,i}^{[c']} + s_j^{[c']} \zeta_{j,i}^{[c']} \\ \rho_{j,i+1}^{[c']} &= \rho_{j,i}^{[c']} - \hat{v}_{j,i}^{[c']} w_i^{[c']} \\ v_{j,i}^{[c'+2]} &= \hat{v}_{j,i}^{[c']} a_i^{[c']} + \rho_{j,i}^{[c']} b_i^{[c']} \end{cases}$$

Figure 8.13: Signal flow graph for the pipelined LU based version of algorithm D. Formulas for the cells of figure 8.12.

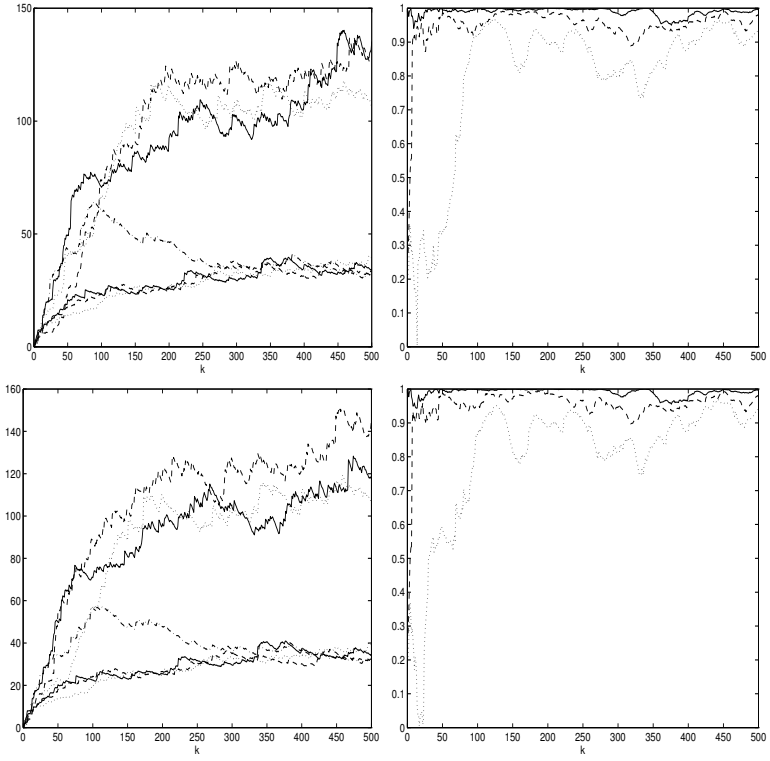


Figure 8.14: Numerical experiment with algorithm D to justify the use of older estimates of $y_1, \ldots, y_m$. Input $x \in \mathbb{R}^7$ generated by $x = Ms+n$ with $E\{ss^T\} = I$, $E\{nn^T\} = (.5)^2 I$ and a slowly time-varying orthogonal $M \in \mathbb{R}^{7\times 3}$ with an average maximal canonical angle of 1.5 degrees between consecutive values of subspaces $\mathrm{col}(M)$. Forgetting factor $\rho = 0.99$. Left: Estimated eigenvalues of $E\{xx^T\}/(1-\rho)$. Right: cosines of canonical angles between signal subspace estimate and generating signal subspace. Top: with spherical subspace step size (8.4) Bottom: with approximated step size using older estimates of y to estimate the eigenvalues. There is no qualitative difference in performance.

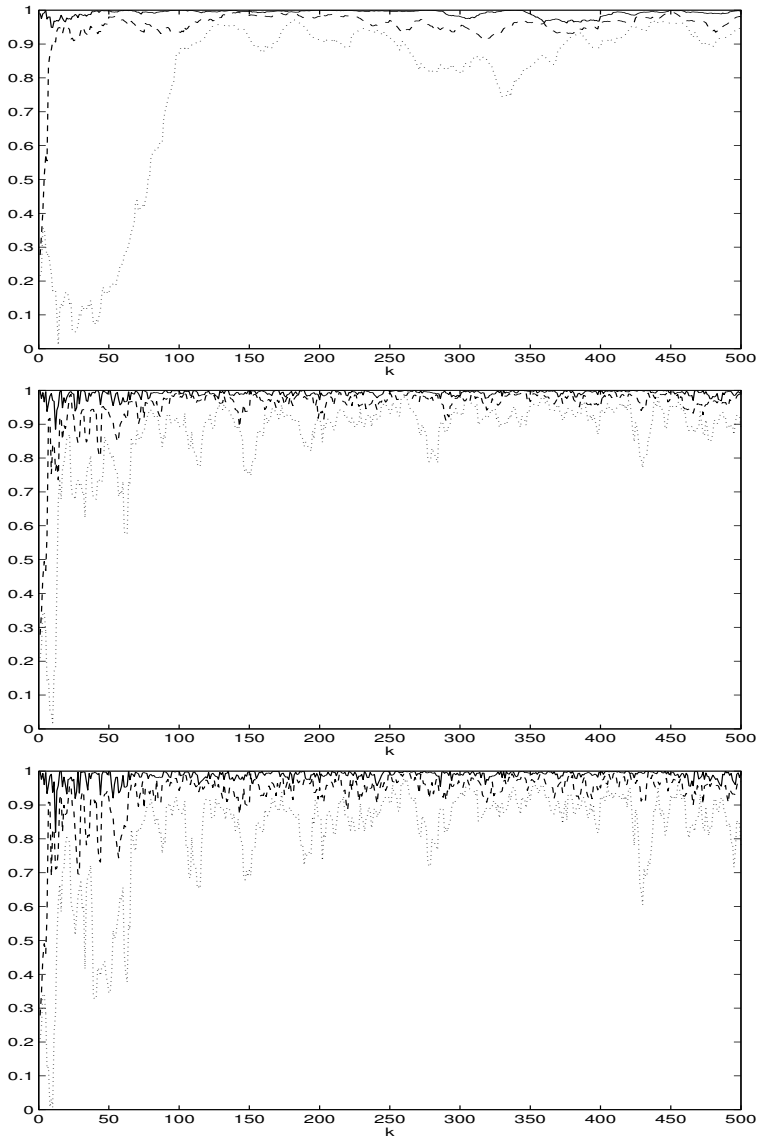


Figure 8.15: Numerical experiment for algorithm D with different step sizes as in (8.6) with different values of K. Input and forgetting factor as in figure 8.14. Cosines of canonical angles between signal subspace estimate $\mathrm{col}(A)$ and generating signal subspace $\mathrm{col}(M)$. Top: $K = 1$. Middle $K = 10$. Bottom $K = 20$. The best results here are obtained for $K = 10$. In cases with less noise even more substantial improvements can be observed.

Appendix

8.A Derivation of the signal flow graph of algorithm 8.5

In this appendix we describe the derivation of the signal flow graph for the LU based version of algorithm D. The unpipelined signal flow graph is shown in figures 8.9 and 8.10. The pipelined signal flow graph is shown in figures 8.11, 8.12 and 8.13. For the derivation, we will think in terms of the case with $m = n$. The figures show signal flow graphs for the case with $m < n$. They are obtained from the case with $m = n$ by simply omitting the $n - m$ last columns.

To derive the unpipelined signal flow graph of figures 8.7 and 8.8, we have to implement right multiplication of L with $\mathrm{lf}(I + \alpha w y^T)$, right multiplication of V with $\mathrm{iuf}(I + \alpha w y^T)$ and left multiplication of V with $\mathrm{cf}(I + \beta z z^T)$. The multiplication with $\mathrm{cf}(I + \beta z z^T)$ amounts to applying Givens rotations as for the discrete-time GK algorithm (algorithm 6.16) with input $\sqrt{\beta}y$. Substituting R with V and x with $\sqrt{\beta}y$ in the second description of algorithm 6.16, it follows that

$$\begin{aligned} \mathrm{cf}(I + \beta z z^T)V &= \mathrm{cf}(V^T V + \beta y y^T) \\ &= \left[\begin{array}{c|c} I_{n+1} & 0_{n+1\times 1} \end{array}\right] G^{n+1|n+2}(\phi_{n+1}) \ldots G^{1|n+2}(\phi_1) \left[\begin{array}{c} V \\ \hline \sqrt{\beta}y \end{array}\right]. \end{aligned}$$

The multiplication with $\mathrm{cf}(I+\beta z z^T)V$ can therefore be implemented on the part storing V in the same way as the GK array (figure 6.2). To obtain $\sqrt{\beta}y$, $y = L^T x$ is calculated in the part storing L by the signals $\eta_{i,j}$ as in the continuous-time case, $\sqrt{\beta}$ is calculated in cell $|n, 1|$ (similar to cell $|1, 1|$ of algorithm D), and the multiplication is done in the cells $< n, j >$, $j = 1, \ldots, n$.

For the multiplications with $\mathrm{lf}(I + \alpha w y^T)$ and $\mathrm{iuf}(I + \alpha w y^T)$ we proceed in a similar way as for the GK array and the MM array (figures 6.2, 6.5 and 6.6). The signals $\sigma_{i,j}$ carrying partial sums (see chapter 5) in the continuous-time signal flow graph, are again thought of as an extra column appended to L for the discrete-time signal flow graph. This extra column affects and is affected by the successive columns of L as it travels through the array. In a similar way $\rho_{i,j}$ is thought of as an extra column of V. The updating of the columns of L and V and of the extra columns is described by the matrices $J_{(j)}$, $j = 1, \ldots, n$, playing a similar role as the Givens rotations.

$$J_{(j)} = a_j E^{(j,j)} + b_j E^{(n+1,j)} + c_j E^{(j,n+1)} + d_j E^{(n+1,n+1)} + \sum_{i=1, i\neq j}^{n} E^{(i,i)}. \quad (8.7)$$

(Note that c_j is distinct from $c_j = \cos(\phi_j)$ in the Givens rotations mentioned above. Below we will find that c_j in (8.7) equals $-w_j$, and no longer use c_j in this context).

Writing

$$L' = L\,\mathrm{lf}(I + \alpha w y^T) = \mathrm{lf}(L + \alpha x y^T) = \mathrm{lf}([L \mid x] J J^{-1} \left[\frac{I}{\alpha y^T}\right]),$$

where $J = J_{(1)} \cdots J_{(n)}$, and requiring that

$$J^{-1} \left[\frac{I}{\alpha y^T}\right] \text{ is upper triangular,} \tag{8.8}$$

and

$$[L \mid x]J \text{ is lower triangular with unit diagonal,} \tag{8.9}$$

one finds that

$$[L' \mid 0] = [L \mid x]J \tag{8.10}$$

and

$$J^{-1} \left[\frac{I}{\alpha y^T}\right] = \left[\frac{\mathrm{uf}(I + \alpha w y^T)}{0}\right].$$

To find $\mathrm{iuf}(I + \alpha w y^T)$, we add a dummy column and invert. Inversion of

$$J^{-1} \begin{bmatrix} I & 0 \\ \alpha y^T & -1 \end{bmatrix} = \begin{bmatrix} \mathrm{uf}(I + \alpha w y^T) & u \\ 0 & v \end{bmatrix}$$

yields

$$\begin{bmatrix} I & 0 \\ \alpha y^T & -1 \end{bmatrix} J = \begin{bmatrix} \mathrm{iuf}(I + \alpha w y^T) & -u/v \\ 0 & 1/v \end{bmatrix}. \tag{8.11}$$

As explained above, we denote the intermediate results for the extra column appended to L after multiplication with the successive operations $J_{(j)}$ by $\sigma_{i,j}$. That is,

$$\left[\begin{array}{c|c} L & x \end{array}\right] J_{(1)} \cdots J_{(j-1)} = \left[\begin{array}{c|c|c} L'_{\cdot,<j} & L_{\cdot,\geqslant j} & \begin{matrix} 0_{(j-1)\times 1} \\ \sigma_{\geqslant j,j} \end{matrix} \end{array}\right]$$

or

$$\begin{bmatrix} l'_{i,j} & \sigma_{i,j+1} \end{bmatrix} = \begin{bmatrix} l_{i,j} & \sigma_{i,j} \end{bmatrix} \begin{bmatrix} a_j & c_j \\ b_j & d_j \end{bmatrix}. \tag{8.12}$$

The entries a_j, b_j, c_j and d_j are found from the conditions (8.8) and (8.9). The operations $J_{(j)}$ have to zero the successive entries of x in (8.9), and the operations $J_{(j)}^{-1}$ in (8.8) or $J_{(j)}$ in (8.11) have to zero the entries of y. To this end, a_j, b_j, c_j and d_j should satisfy

$$\begin{bmatrix} 1 & \sigma_{j,j} \end{bmatrix} \begin{bmatrix} a_j & c_j \\ b_j & d_j \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{8.13}$$

and

$$\begin{bmatrix} y_j & \epsilon_j \end{bmatrix} \begin{bmatrix} a_j \\ b_j \end{bmatrix} = 0, \tag{8.14}$$

where ϵ_j keeps track of the right most entry of $[y^T \mid -1/\alpha]$ in (8.11) while being affected by the operations $J_{(j)}$. That is,

$$[y^T \mid -1/\alpha] J_{(1)} \cdots J_{(j-1)} = [0_{1\times(j-1)} \mid y_{\geqslant j}^T \mid \epsilon_j]$$

or

$$\epsilon_{j+1} = y_j c_j + \epsilon_j d_j . \tag{8.15}$$

The conditions (8.13) and (8.14) leave an extra degree of freedom for the choice of a_j, b_j, c_j and d_j. Choosing $\det J_{(j)} = 1$, $j = 1, \ldots, n$, one obtains

$$\begin{array}{rcl} a_j &=& \epsilon_j/(\epsilon_j - \sigma_{j,j} y_j) \\ b_j &=& -y_j/(\epsilon_j - \sigma_{j,j} y_j) \\ c_j &=& -\sigma_{j,j} \\ d_j &=& 1. \end{array} \tag{8.16}$$

Eliminating c_j and d_j, and observing that $\sigma_{j,j}$ is equal to the component w_j of $w = L^{-1}x$ (as in the continuous-time case), the equations (8.12), (8.15) and (8.16) can be recast as

$$\begin{array}{rcl} l'_{i,j} &=& l_{i,j} a_j + \sigma_{i,j} b_j \\ \sigma_{i,j+1} &=& \sigma_{i,j} - l_{i,j} w_j \\ \epsilon_{j+1} &=& \epsilon_j - y_j w_j \\ a_j &=& \epsilon_j/\epsilon_{j+1} \\ b_j &=& -y_j/\epsilon_{j+1}. \end{array} \tag{8.17}$$

Finally, $\hat{V} = \mathrm{cf}(I + \beta z z^T)V$ has to be multiplied with $\mathrm{iuf}(I + \alpha w y^T)$ to obtain V'. From (8.11), this means adding a zero column, applying the operations J and dropping the extra column. The extra column is realized by the signals $\rho_{i,j}$. Putting everything together leads to the signal flow graph of figures 8.9 and 8.10.

Unlike in the continuous-time signal flow graph, the output $z = V^{-T} L^T x$ (remember that z corresponds to $y = A^T x$ in algorithm D) is not available directly in the signal flow graph for updating L and V. However it can be extracted by means of extra signals ϕ_j, $j = 1, \ldots, n$ running along the diagonal cells of V and accumulating the products $\phi_j = \prod_{i=1}^{j} c_i$. One finds $z_j = \zeta_{j,j}/(\phi_j c_j)$. This can be proved by induction, and is similar to the way in which RLS residuals can be extracted from a GK array as described by [75].

Next, we have to pipeline this signal flow graph. We first consider vertical pipelining of the part storing L. We follow the same strategy as for algorithm D in section 8.2. Delays are introduced on lines going upwards, older versions of L are stored in more upward rows, and corrections are made on lines going downwards. To this end the signals $\eta_{i,j}$ are replaced by $\bar{\eta}_{i,j}$ (the uncorrected signal) and $\tilde{\eta}_{i,j}$ (the corrected signal), defined as follows,

$$\begin{array}{rcl} \tilde{\eta}_{i,j}^{[k]} &=& x_{<i}^{[k]T} L_{<i,j}^{[k+i-n]}, \\ \bar{\eta}_{i,j}^{[k]} &=& x_{\leqslant i}^{[k]T} L_{\leqslant i,j}^{[k+i-n]}. \end{array}$$

The correction to obtain $\tilde{\eta}_{i+1,j}$ follows from

$$\begin{aligned}
\tilde{\eta}^{[k]}_{i+1,\cdot} &= x^{[k]}_{<i+1}{}^{T} L^{[k+i+1-n]}_{<i+1,\cdot} \\
&= x^{[k]}_{\leqslant i}{}^{T} \left[\; L^{[k+i-n]}_{\leqslant i,\cdot} \;\middle|\; x^{[k+i-1]}_{\leqslant i} \;\right] J^{[k+i-n]}_{(1)} \cdots J^{[k+i-n]}_{(n)} \left[\frac{I}{0}\right] \\
&= \left[\; \bar{\eta}_{i,\cdot} \;\middle|\; x^{[k]}_{\leqslant i}{}^{T} x^{[k+i-n]}_{\leqslant i} \;\right] J^{[k+i-n]}_{(1)} \cdots J^{[k+i-n]}_{(n)} \left[\frac{I}{0}\right].
\end{aligned}$$

That is, $\tilde{\eta}_{i+1,\cdot}$ is obtained from $\bar{\eta}_{i,\cdot}$ by adding an extra entry $x^{[k]}_{\leqslant i}{}^{T} x^{[k+i-n]}_{\leqslant i}$, applying the operations $J^{[k+i-n]}$ and dropping the extra entry. The extra entry is realized by the signals $\epsilon_{i,j}$. To calculate $\epsilon_{i,1} = x^{[k]}_{\leqslant i}{}^{T} x^{[k+i-n]}_{\leqslant i}$, an extra triangular part is added to the array, identical to the extra part that was needed for the vertically pipelined version of algorithm D (but upside down).

Also the signals w_j going downwards are replaced by signals $\tilde{w}_j$, calculated with an older version of L and corrected at each new row. We define

$$\tilde{w}^{[k]}_{i,\leqslant i} = x^{[k]}_{\leqslant i}{}^{T} L^{[k+i-n]-T}_{\leqslant i,\leqslant i}.$$

The corrected signals are

$$\tilde{w}_{i+1,\leqslant i} = x^{[k]}_{\leqslant i}{}^{T} L^{[k+i+1-n]-T}_{\leqslant i,\leqslant i}. \tag{8.18}$$

To obtain this correction in terms of the operations $J_{(j)}$ we add a dummy row to 8.10, and take the inverse transpose. From

$$\begin{bmatrix} L^{[k+i+1-n]} & 0 \\ f^{T} & \varphi \end{bmatrix} = \begin{bmatrix} L^{[k+i-n]} & x^{[k+i-n]} \\ 0 & -1 \end{bmatrix} J^{[k+i-n]}_{(1)} \cdots J^{[k+i-n]}_{(n)},$$

one finds

$$\begin{bmatrix} L^{[k+i+1-n]-T} & -f/\varphi \\ 0 & 1/\varphi \end{bmatrix} =$$

$$\begin{bmatrix} L^{[k+i-n]-T} & 0 \\ x^{[k+i-n]T} L^{[k+i-n]-T} & -1 \end{bmatrix} J^{[k+i-n]-T}_{(1)} \cdots J^{[k+i-n]-T}_{(n)}.$$

Plugging the result for $L^{[k+i+1-n]-T}$ in (8.18), yields

$$\tilde{w}_{i+1,\leqslant i} = [\tilde{w}^{[k+i-n]}_{i,\leqslant i} \mid 0] J^{[k+i-n]-T}_{(1)} \cdots J^{[k+i-n]-T}_{(i)} \left[\frac{I}{0}\right].$$

That is, the correction to $\tilde{w}_{i,\leqslant i}$ to obtain $\tilde{w}_{i+1,\leqslant i}$ is realized by adding an extra entry 0, applying the operations $J^{-T\,[k+i-n]}$ and dropping the extra entry. The extra entry is realized by the signals $\theta_{i,j}$. The operations $J^{-T\,[k+i-n]}_{(j)}$ are given by

$$J^{-T\,[k+i-n]}_{(j)} = \begin{bmatrix} a_j & -w_j \\ b_j & 1 \end{bmatrix}^{-T} = \begin{bmatrix} 1 & -b_j \\ w_j & a_j \end{bmatrix}$$

Next to $\tilde{w}_{i,j}^{[k]}$, also the original $w_j^{[k+i-n]}$ is needed at cell (i,j) for the application of the operation $J_{(j)}^{[k+i-n]}$, this signal can be passed upwards together with $a_j^{[k+i-n]}$ and $b_j^{[k+i-n]}$.

Finally, vertical pipelining of the part storing L is obtained by two-slowing and cut-set retiming along horizontal cuts. Vertical pipelining of the part storing V and horizontal pipelining of the whole array are obtained by simple cut-set retiming, adding delays on lines going downwards or to the right. The result is shown in figures 8.11, 8.12 and 8.13. The correctness of this signal flow graph has been verified experimentally.

Chapter 9

Neural network adaptation laws for continuous-time adaptive signal processing.

In this chapter and chapter 10 we consider continuous-time algorithms as a framework for studying neural adaptation laws. In this chapter we give a neural interpretation to the class of uniform parallel algorithms defined in definition 5.1, and the examples discussed in chapters 6, 7 and 8. In chapter 10 we will consider adaptation laws for recurrent neural networks, used as memories.

For general introductions to artificial neural networks and their adaptation laws we refer to [49, 112, 66].

The main results of this chapter were published in [34, 27, 26].

9.1 General concepts

Two general ideas are borrowed from biology. The *first* idea is that complex behavior can be obtained by grouping together a number of *similar, simple* elements (neurons), and making them interact. Roughly speaking neurons *weigh and sum* information from other neurons and send the result to other neurons. The *second* idea is that the weights by which neurons weigh incoming information, *adapt* during use, influenced by *local* information, obeying laws that are often *uniform* over (parts of) the network.

The first idea is reflected as follows in *forward* and *recurrent* neural models.

Common *forward* (static) models define the state of a neuron i by

$$z_i = f(\sum_{j \in \mathcal{N}_i} a_{i,j} z_j), \tag{9.1}$$

where $j \in \mathcal{N}_i$ runs over all neurons that provide input to neuron i. The quantities $a_{i,j}$ are called weights, and are attached to the connections (synapses) between neurons (figure 9.1). And $f : \mathbb{R} \to \mathbb{R}$ is a function that introduces nonlinearity, typically f has a shape as in figure 9.2. In *forward* models information flows in one direction only. If neuron i provides direct or indirect input to neuron j, neuron j does not provide direct or indirect input to neuron i.

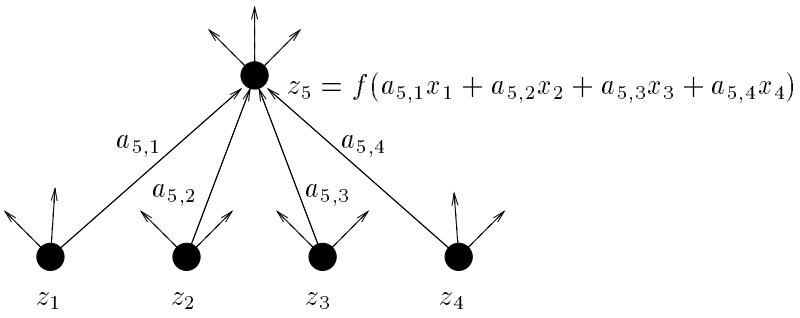


Figure 9.1: A neuron weighs and sums input from other neurons.

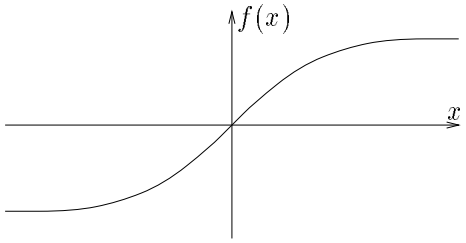


Figure 9.2: Typical nonlinearity of a neuron.

Common *recurrent* (dynamic) models, define the state of a neuron i in discrete time by

$$z_i^{[k+1]} = f(\sum_{j \in \mathcal{N}_i} a_{i,j} z_j^{[k]}),$$

or in continuous time by

$$\dot{z}_i(t) = -z(t) + f(\sum_{j \in \mathcal{N}_i} a_{i,j} z_j(t)).$$

In these models a neuron can influence its own state at a later moment, or neurons can *mutually* influence each other directly or indirectly. Note that in equilibrium the state of a neuron corresponds to the forward model (9.1).

The *second* idea, of *local adaptation laws*, introduces dynamics in forward models, or adds a 'second level' of dynamics to recurrent models. If necessary we will distinguish both levels as *operation level* and *adaptation level*. The networks discussed in this chapter are forward networks with adaptation dynamics, the networks of chapter 10 possess both levels of dynamics.

Our interest in neural networks focuses on obtaining interesting behavior from simple neural networks with uniform local adaptation laws. By *uniform*, we mean that all weights obey the same law. In a less strict sense, we will also allow a small number of different laws. But the notion is opposite to directly designing different elements for differing subtasks. By *local* we mean that the information that determines the adaptation of a weight of a certain connection can be found locally in that connection or the neurons connected by it. But also for the use of the qualification 'local', we will not be very restrictive, and sometimes allow additional, uniform (operation-level-like) information processing, to calculate information in the neurons for adaptation of the weights. In this sense the label 'local' should be interpreted simply as 'implementable on the network structure', as is common practice in neural network literature. Nevertheless, the neural interpretation of the Gentleman-Kung algorithm, and the local version of algorithm C (algorithm 9.1) are 'more local' than most adaptation laws in literature, in that they only use information that is already present in the network for the operation level. Finally, to avoid confusion, note that the notion of 'local adaptation' is distinct from 'local connectivity', meaning that neurons are only connected to nearby other neurons.

We also point out features that do not correspond to biology. *First*, in many artificial neural models, connections are *bidirectional*, that is, used in two directions. In some models neurons influence each other mutually with the same weight (on the operation level). In other models, information to adapt weights (adaptation level) is obtained through signals that use the connections in the opposite direction than on the operation level. It seems that most biological connections are unidirectional. In this work both unidirectional and bidirectional models are used. An advantage of unidirectional models is also that often they give rise to simpler signal flow graphs and possibly analog hardware realizations.

Secondly, the notion of uniformness should not be interpreted too strictly when applied to biological systems. A great variety of different adaptation mechanisms exist. Nevertheless, this notion is applicable to parts of biological systems, and even when different subsystems perform differing subtasks of the processing, they can evolve by essentially the same adaptation laws. However it is not our aim to investigate these type of questions.

Finally, in studying uniform local adaptation laws, we have chosen to restrict ourselves to systems that can be studied by rigorous mathematics. This has brought us to the systems of definition 5.1 which, for some readers, may be too nicely behaved or 'too linear' to be called neural networks. However, we believe that many ideas can be generalized to a broader range of systems, even though the mathematical description may be less thorough.

9.2 Neural interpretation of uniform parallel algorithms of types I and II

In this section and section 9.3 we give a neural network interpretation to the uniform parallel continuous-time matrix algorithms of definition 5.1. Terms of type I use information in the neurons in a Hebbian way. Terms of type II are generalized Hebbian in the sense of [91]. Terms of type III are a new generalization, and are considered in the next section.

In this chapter, the matrix A of the algorithms 5.1, corresponds to the weight matrix of a fully connected one-layer forward neural network. In chapter 10 we will also consider fully connected *recurrent* models. The explanation below is given in terms of forward nets. In the next chapter we will also consider terms of type I as adaptation laws for recurrent networks. They fit in the scheme given below, by identifying the first and second layer of the forward network.

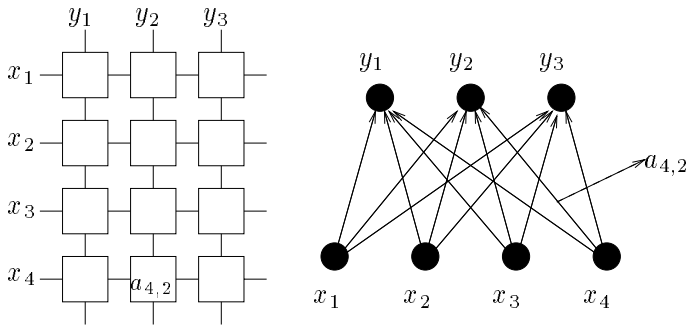


Figure 9.3: Correspondence between array representation and network representation.

Figure 9.3 shows the correspondence between the array representation used in chapter 5 and the network representation that is more common in neural network literature. Neurons of the first layer correspond to rows of the array, neurons of the second layer correspond to columns, and connections between neurons are located at the intersection of rows and columns. The vectors x, c, and e in

definition 5.1 are located in the first layer, components x_i, c_i or e_i in neuron i of the first layer. The vectors y, d and f are located in the second layer, components x_j, d_j or f_j in neuron j. For systems of type III, the resulting network will no longer be a single layer network in the classical sense, and some dynamics will be introduced in the neurons.

The matrix vector multiplication to obtain $c = A^T x$ from x, corresponds to the normal operation of a linear forward network (figure 9.3). The multiplications with A instead of A^T, for instance to obtain $d = Ac = AA^T x$, use the connections in the other direction. Linear neurons simply make a linear combination of incoming information (f is the identity map in (9.1)). For most networks in this chapter the neurons will be linear neurons. The nonlinearity of the systems studied is situated entirely in the adaptation laws.

The oldest and most famous local adaptation law is the Hebb law [51]. It says that the weight $a_{i,j}$ of the connection between neurons i and j adapts proportionally to the product of the activations z_i and z_j of these neurons (figure 9.4). *Systems of type I* in definition 5.1 generalize this idea by not only allowing acti-

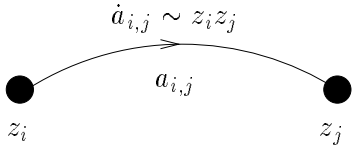


Figure 9.4: Classical Hebbian adaptation law.

vations of neurons but all vectors c, d that can be obtained in the neurons by a (small) number of matrix vector multiplications. Figure 9.5 shows a forward net calculating $d = A^T x$ and adapting A by $\dot{a}_{i,j} = x_i d_j$. This realizes the system $\dot{A} = xx^T A$ of type I.

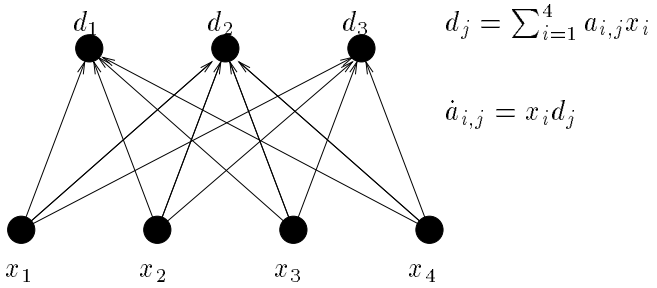


Figure 9.5: Neural network representation of the system $\dot{A} = xx^T A$ of type I.

As pointed out in chapter 5, *systems of type II* are related to systems of type I but

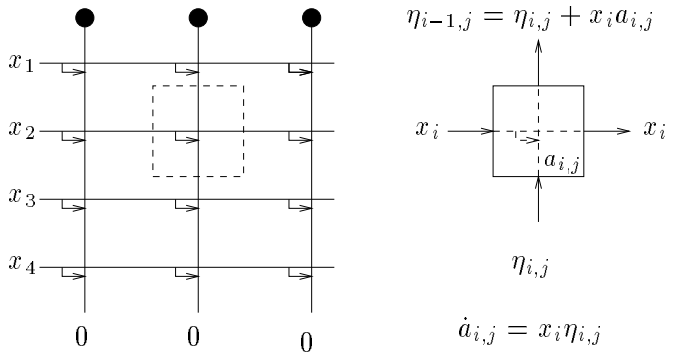


Figure 9.6: Neural network representation of the system $\dot{A} = \text{upp}(xx^T)A$ of type II.

use partial sums, obtained as intermediate results for the calculation of vector products, to adapt the weights. Figure 9.6 shows a neural representation for the system $\dot{A} = \text{upp}(xx^T)A$ of type II. In order to show the partial sums, we have chosen a picture that is closer to the array picture. In biological terms, vertical lines can be thought of as axons, accumulating incoming information. Every connection (synapse) adds its share. Horizontal lines are dendrites, sending information to other neurons. Actually, also type II adaptation laws for the connections are very close to Hebb's original idea. But instead of information in a neuron, the partial sums, $\eta_{i,j}$, $\eta_{i+1,j}$ or $\frac{1}{2}(\eta_{i,j} + \eta_{i+1,j})$ (for tri = upp, tri = upps or tri = upph), that can be found at the connection, are used. In the representation of figure 9.6, the use of these partial sums actually makes the adaptation law 'more local' than the use of the total sum as in systems of type I.

Recall that algorithm C and continuous-time versions of Sanger's related generalized Hebbian algorithm [91], and the continuous-time MM algorithm (making abstraction of the forgetting mechanism) are systems of type II. However, all of them are more complicated than the system of figure 9.6 in that they use the weights in two directions.

9.3 Neural interpretation of uniform parallel algorithms of type III and related algorithms

In this section we consider the neural interpretation of systems of type III in definition 5.1 and related systems. First, we consider the case of the continuous-time algorithms of chapter 6. For one of the LDL based algorithms a simple

interpretation for the connections is obtained. We also show how linearly constrained minimum variance beamforming (LCMV) can be realized by the same network with an extra linear layer. Next we derive a new form of algorithm C whose neural realization is the same network as for the LCMV beamforming, now with an adaptive first layer of type I, and which uses only information that is available already at the operation level, to adapt the weights.

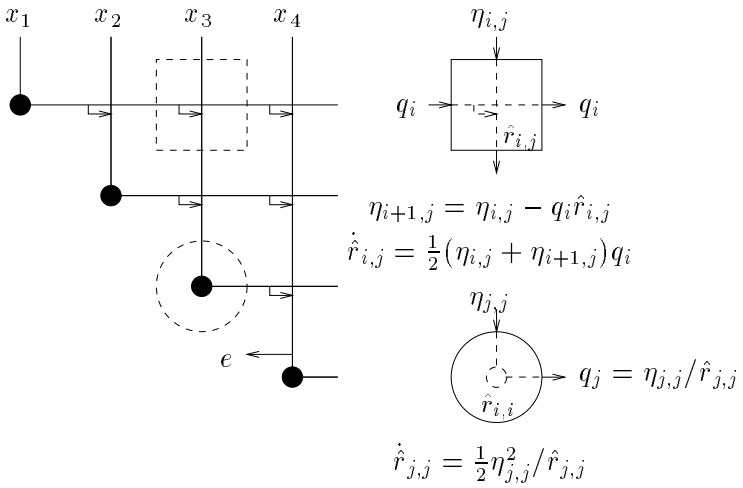


Figure 9.7: Neural network representation of the continuous-time Gentleman Kung algorithm (with $\lambda = 0$).

Figure 9.7 shows the neural interpretation of the continuous-time Gentleman Kung algorithm (algorithm 6.10) with $\lambda = 0$, given by

$$\dot{\hat{R}} = \text{upph}(\hat{R}^{-T} x x^T \hat{R}^{-1}) \hat{R},$$

where we have used the notation of chapter 6. To get correspondence with the notation of definition 5.1 one should set $A = \hat{R}$ and $y = x$ (as the input is supplied to the columns of the array).

To implement the back-substitution scheme for the calculation of $b = A^{-T} y$ (equations 5.1 and 5.2, figure 5.4), three new elements are added to the systems of type II.

(i) The partial sums $\eta_{i,j}$ in (5.2) are calculated by subtracting information from the input y instead of adding information starting from 0.

(ii) To calculate b_i from the last partial sum a division is needed.

(iii) The component b_i is already needed for the calculation of components b_j, $j > i$.

Due to (iii), the resulting network is no longer a one-layer network, but neurons get input from all previous neurons. Property (ii) gives rise to different interpretations for the neurons. In figure 9.7 we have interpreted $r_{i,i}$ as an internal state of the neuron, and $b_{i,i} = \eta_{i,i}/r_{i,i}$ as the output of the neuron (although for the application of recursive least squares estimation $\eta_{i,i}$ itself is needed as output of the network). Alternatively, one can consider $r_{i,i}$ as a connection from neuron i to itself, and determine b_i such that $\eta_{i+1,i} = 0$ (after this connection). This can be realized by the feed back mechanism, explained in section 6.6. In that case the neuron corresponds to the amplifier.

If one considers the LDL-based algorithm 6.21 (with free adaptation rates for the LMS cells), neurons are again simple linear neurons. And also the connections, corresponding to the LMS cells, get a simpler interpretation. It is well known that the LMS algorithm has a neural interpretation [104]. But in the case of only one input (and one reference input) the LMS algorithm is simply realized by one synapse, adapted using the partial sum obtained after that synapse has contributed its share to the input of the neuron.

In some applications of RLS estimation, the input vector is first multiplied by a fixed rectangular matrix, before being supplied to the GK algorithm (or any linear estimation algorithm). In neural network terms this simply corresponds to adding a neural network layer with fixed weights in front of the RLS network. In LCMV beamforming for example [100], the inputs come from a linear array of n antennas, receiving a planar wave. The beamformer linearly combines these inputs in an adaptive way, minimizing the output power subject to one or more constraints. A typical constraint requires that signals from a given direction are passed unchanged. The constraints determine the fixed preprocessing matrix. Figure 9.8 shows a neural network version of such a beamformer.

Also a subspace tracking algorithm can be implemented on the same network structure, now by making the first layer adaptive. This is realized by implementing algorithm C, storing B and $R = S^{-1}$, defined as in theorem 8.1 (but not for constant x), instead of A. One obtains

Algorithm 9.1

$$\begin{aligned} \dot{B} &= xx^T B, \\ \dot{R} &= 2\,\mathsf{upph}(R^{-T}B^T xx^T BR^{-1})R. \end{aligned}$$

Setting $A = BR^{-1}$ one finds algorithm C again. The equation for $\dot{B}$ is the simple Hebbian law of type I of figure 9.5. The equation for $\dot{R}$ is the GK algorithm of figure 9.7, taking the output $B^T x$ of the part realizing B as input. However, a disadvantage of this approach is that B can get large and close to singular, such that in practice some extra scheme will be needed, bringing B closer to

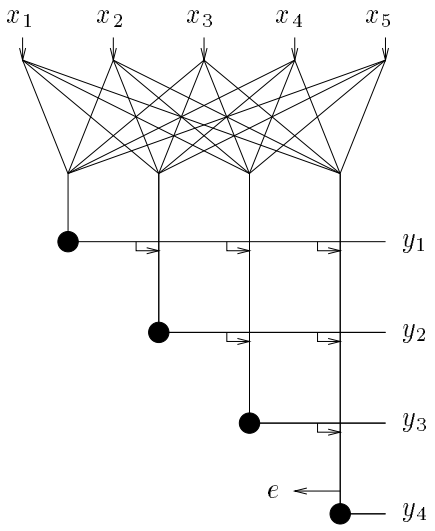


Figure 9.8: Neural network representation of a system for LCMV beamforming or subspace tracking. For LCMV beamforming, the top part has fixed weights, and the output is e. For subspace tracking, the top part is Hebbian (as in figure 9.5, upside down) and the outputs are y_1 to y_4.

orthogonality again (and adapting R accordingly to leave BR^{-1} unaffected). This may make the network more complicated again.

9.4 Conclusion

We have interpreted the algorithms of definition 5.1 in the context of neural networks. They form a class of adaptation laws that can be implemented on the network, and at the same time allow for a mathematical analysis of the global behavior. In this chapter we have considered adaptation laws for forward neural networks for recursive least squares estimation and subspace tracking, based on the algorithms of chapters 6 and 7. In the next chapter, we consider adaptation laws for recurrent neural networks for associative-memory applications.

Chapter 10

Continuous-time adaptation laws for neural network memories

In this chapter we discuss continuous-time adaptation laws for recurrent neural networks, used as associative memories.

We consider three related fully connected recurrent models, the Hopfield model [58, 59], the brain-state-in-a-box model (BSB) [1], and linear systems in saturated mode (LSSM) [72]. These models are defined in section 10.1 and discussed more extensively in appendix 10.A.

Although these models suffer from limited capacity and although it is difficult to avoid unwanted stable states (stored patterns), they are still a topic of actual research. One reason is that the underlying idea of realizing memory with dynamical systems remains an appealing idea. The main idea is that patterns can be retrieved from memory, when only supplying perturbed or incomplete patterns. Another advantage is that the resulting memories possess the property of graceful degradation. Damage to a limited number of connections does not severely affect the operation of the memory. For a more general discussion of associative memories in different neural forms, we refer to [66].

The main contribution of this chapter is the derivation of adaptation laws, which can be realized on the net. These laws store patterns in the memory, that are shown by a teaching signal. As opposed to other adaptation laws, the number of extra stored patterns is kept low and can be characterized mathematically. The obtained result is equivalent to a known (off-the-net) design algorithm.

Another contribution consists in reducing the complexity of the attraction mechanism of section 10.3 by exchanging spatial complexity with time complexity in a way that is susceptible to generalization.

This chapter is organized as follows. In sections 10.1 we discuss three neural network models, and their application as associative memories. The next three sections introduce new continuous-time algorithms, apart from the memory application. Section 10.2 discusses a nonautonomous variant of the double bracket flow, that fits in the class of uniform parallel algorithms of definition 5.1. To preserve the isospectrality of the algorithm in spite of inaccurate realizations, section 10.3 considers modifications to stabilize the spectrum. In sections 10.4 these modifications are adapted to fit in the class of uniform parallel systems (definition 5.1).

Section 10.5 discusses the application of these algorithms as adaptation laws for neural associative memory. Section 10.6 compares them with other laws, and section 10.7 gives some simulation results.

The main results of this chapter were published in [35, 33, 32].

10.1 Associative memory

The memory application discussed in this chapter will be developed for the brain-state-in-a-box type model (BSB) [1], but can also be applied to the related Hopfield type model [58, 59] with piecewise linear limiters and to linear systems in saturated mode (LSSM) [72]. In this section we define these three models, briefly discuss how they can be used as memories and describe a known design algorithm [72] for the weights.

Definition 10.1
The *brain-state-in-a-box model* (BSB) is described by

$$\dot{z} = -z + f(Az + b), \tag{10.1}$$

$z \in \mathbb{R}^n$ is the *state*,
$b \in \mathbb{R}^n$ is the *external input*,
$A \in \mathcal{S}_{symm}(n)$ is the symmetric *weight matrix*,
and $f : \mathbb{R}^n \to \mathbb{R}^n$ is defined by

$$\begin{array}{rcl} f_i(z) & = & -1 \text{ if } z_i \leqslant -1, \\ f_i(z) & = & z_i \text{ if } -1 \leqslant z_i \leqslant 1, \\ f_i(z) & = & 1 \text{ if } z_i \geqslant 1. \end{array} \tag{10.2}$$

Definition 10.2
The *Hopfield model* (with piecewise linear limiters) is described by

$$\begin{aligned} \dot{z} &= -z + Ay + b, \\ y &= f(z). \end{aligned} \tag{10.3}$$

$z \in \mathbb{R}^n$ is the *state*,
$y \in \mathbb{R}^n$ is the *output*,
$b \in \mathbb{R}^n$ is the *external input*,
$A \in \mathcal{S}_{symm}(n)$ is the symmetric *weight matrix*,
and f is the *limiter* and is defined as in the BSB model (definition 10.1).

Definition 10.3
Linear systems in saturated mode (LSSM) are described by

$$\dot{z} = (A - I)z + b, \text{ constrained by } -1 \leqslant z_i \leqslant 1, \tag{10.4}$$

$z \in \mathbb{R}^n$ is the *state*,
$b \in \mathbb{R}^n$ is the *external input*,
$A \in \mathcal{S}_{symm}(n)$ is the *weight matrix*.

The constraint $-1 \leqslant z_i \leqslant 1$ should be interpreted as follows. If for some component $|z_i| = 1$ and $z_i((A - I)z + b)_i > 0$, that is, if the component is about to leave the interval $[-1, 1]$, then $\dot{z}_i$ is set to 0, and the component is kept at $z_i = 1$ or $z_i = -1$, until $z_i((A - I)z + b)_i < 0$.

A more detailed study of the relations between these three models, is given in appendix 10.A at the end of this chapter, also see [30].

These models have been proposed as models for associative memory in the following way. The stable equilibria of the network are stored memory patterns. The three models are suited to store "binary" patterns $p \in \{-1, 1\}^n$ as equilibria. A pattern is retrieved by supplying a perturbed (or incomplete) pattern as an initial state and letting the network evolve towards one of its stable patterns. The behavior of this memory is determined by the weights of the connections. If the weight matrix A is symmetric, all three models converge to equilibria from any initial condition. The main problem is to find an appropriate set of weights, given a set of binary patterns $\mathcal{P} = \{p_i\}_{i=1}^m \subset \{-1, 1\}^n$ to be stored in the memory. The notion of appropriate weights can be interpreted in many ways. In the minimal sense it means that the given patterns should be stable equilibria. In addition, requirements can be imposed on the presence of other (unwanted) stable patterns and the nature of the attraction basins of equilibria, that is, which patterns are attracted to which equilibrium. In this chapter we will pay some attention to the problem of unwanted patterns. We do not prove results about attraction basins.

We will call a *design algorithm*, any algorithm for calculating appropriate weights for a set of patterns $\mathcal{P}$. We will call it an *adaptation law* or *adaptation algorithm*, (sometimes also called 'learning rule') if the algorithm prescribes a gradual adaptation of the weights, governed by some input signal. All adaptation algorithms considered will be on-the-net algorithms. This means that their signal flow graph can be mapped onto the network. In the remainder of this section we discuss a design algorithm (algorithm 10.6), that was originally introduced for the LSSM model. In section 10.5 we will discuss *adaptation* laws that converge to an appropriate weight matrix when supplied with an input signal, showing the patterns to be stored. The design algorithm 10.6 can also be applied to the other models mentioned, and we will prove its main property for the BSB model, as theorem 10.8.

An important feature of the design algorithm 10.6 is that it is suited for a theoretical analysis, allowing provable statements about possible unwanted stable states. In this work we want to go further down the same line, emphasizing *provable* results that can be obtained for *adaptation* laws.

Before introducing the design algorithm 10.6, the following lemma states conditions for a pattern to be stored in the memory (that is, to be an asymptotically stable equilibrium).

Lemma 10.4
(i) A binary pattern $p \in \{-1,1\}^n$ is an equilibrium of the BSB model (definition 10.1) iff

$$p_i(Ap+b)_i \geq 1 \quad i = 1,\ldots,n. \tag{10.5}$$

It is asymptotically stable iff

$$p_i(Ap+b)_i > 1 \quad i = 1,\ldots,n. \tag{10.6}$$

(ii) If $b = 0$ and the binary pattern $p \in \{-1,1\}^n$ is an eigenvector of A with eigenvalue $\rho > 1$, then p is an asymptotically stable equilibrium of the BSB model.

Proof:
(i) A pattern $p \in \{-1,1\}^n$ is an equilibrium of the BSB model iff $-p+f(Ap+b) = 0$, or $p_i = f_i(Ap+b)$, for $i = 1,\ldots,n$. From the definition of f (10.2), this is equivalent to $(Ap+b)_i \geq 1$ if $p_i = 1$, and to $(Ap+b)_i \leq -1$ if $p_i = -1$. These conditions can be recast as (10.5).
Under the conditions (10.6), p is asymptotically stable because we can choose a neighborhood $\mathcal{U}$ of p in which $p_i(Az+b)_i > 1$ for all $z_i \in \mathcal{U}$, implying $f(Az+b) = p$. In $\mathcal{U}$, z follows the linear system $\dot{z} = -z + p$, which implies a unique (in $\mathcal{U}$) stable equilibrium at $z = p$.
(ii) This follows immediately from (i), by observing that $p_i(Ap+b)_i = \rho p_i^2 = \rho > 1$, under the given conditions.
□

Remark 10.5
We will further assume that $b = 0$. At the end of section 10.5 we will discuss a possible role for b. As a consequence of the choice of $b = 0$, a pattern p and its inverse $-p$ are always together stable or unstable for a given A.

We are now in a position to introduce the design algorithm 10.6, which is a variant of a design algorithm for the LSSM model [72], reformulated for the BSB model.

Algorithm 10.6
Let $\mathcal{P} = \{p_i\}_{i=1}^k \subset \{-1, 1\}^n$ be a given set of binary patterns to be stabilized and let $\{u_i\}_{i=1}^n$ be an orthonormal basis of the state space $\mathbb{R}^n$, such that $\{u_i\}_{i=1}^k$ is a basis of $\mathcal{X} = \text{span}(\mathcal{P})$.

Then choose

$$A = A_{\rho,\tau} = \rho \sum_{i=1}^{k} u_i u_i^T - \tau \sum_{i=k+1}^{n} u_i u_i^T, \tag{10.7}$$

where $\rho > 1$ and $\tau > 0$ are parameters.

That is, choose A to be the symmetric matrix with an eigenvalue ρ in the eigenspace $\mathcal{X} = \text{span}(\mathcal{P})$ and an eigenvalue $-\tau$ in the orthogonal complement $\mathcal{X}^\perp$.

Remark 10.7
In the original formulation [72], a matrix $T_{\tau'} = \sum_{i=1}^k u_i u_i^T - \tau' \sum_{i=k+1}^n u_i u_i^T$ was proposed, where T corresponds with $A - I$. Positive multiples rT_τ, where $r > 0$, can be considered also, as the multiplication with r amounts to a time scaling for the LSSM model ($b = 0$). The correspondence with our notation (in terms of A instead of $A - I$) is given by $\rho = r + 1$ and $\tau' = (\tau + 1)/(\rho - 1)$.

The main property of this design algorithm is reflected in the following theorem.

Theorem 10.8
If $\tau' = (\tau+1)/(\rho-1)$ is large enough, all and only those binary patterns which lie in $\mathcal{X}$ are asymptotically stable equilibria of the BSB model (definition 10.1), designed with algorithm 10.6.

Proof: Since a binary pattern $p \in \{-1, 1\}^n \cap \mathcal{X}$ is an eigenvector of A with eigenvalue $\rho > 1$, it is an asymptotically stable equilibrium by lemma 10.4(ii). In order to prove that a binary pattern $q \notin \mathcal{X}$ is not an equilibrium when τ' is large enough, we have to show (lemma 10.4(i)), that for some i, $q_i(Aq)_i < 1$, or equivalently $q_i((A - I)q)_i < 0$, from $q_i^2 = 1$. We find

$$\min_j q_j((A - I)q)_j \leq \frac{1}{n} \sum_{j=1}^{n} q_j((A - I)q)_j = \frac{1}{n} q^T (A - I)q,$$

and from (10.7)

$$q^T(A-I)q = (\rho-1)\sum_{j=1}^{k}(q^Tu_j)^2 - (\tau+1)\sum_{j=k+1}^{n}(q^Tu_j)^2,$$

and since $q \notin \mathcal{X}$, $q^Tu_j \neq 0$ for some j, with $k+1 \leq j \leq n$. Therefore, if $\tau' = (\tau+1)/(\rho-1)$ is large enough, $\min_j q_j((A-I)q)_j < 0$, which proves the theorem.
□

The complete characterization of the set of stored patterns is a positive feature of design algorithm 10.6. Moreover, for a random set of patterns with about half as many patterns as neurons, the number of unwanted patterns is often still zero, as opposed to what one expects from low-dimensional examples.

No lower-bound for τ' is provided in [72], but some simulation results are given. One can also easily prove that the number of stable states $q \notin \mathcal{X}$ decreases with increasing τ. Further on we will describe how realizations of the adaptation law can be obtained for which the value of τ' can be manipulated easily.

10.2 A nonautonomous variant of the double bracket flow

In the following sections we discuss some continuous-time algorithms, and only return to the neural application in section 10.5. In this section, we consider a nonautonomous variant of the double bracket flow, given by substituting $N = x(t)x(t)^T$ in algorithms 4.11,

Algorithm 10.9

$$\dot{H} = [H,[H,xx^T]] = H^2xx^T - 2Hxx^TH + xx^TH^2.$$

We will assume that x is sufficiently smooth when necessary.

This is a system of type I in definition 5.1, since $H = H^T \in \mathcal{S}_{symm}(n)$. Recall that in the neural interpretation of chapter 9, for recurrent networks, the first and second layer are identified.

Theorem 10.11 gives the main result for this algorithm. With lemma 10.10, we first show the close relation with theorem 7.29.

Lemma 10.10
(i) If H follows algorithm 10.9, H has a set of eigenvectors satisfying

$$\dot{P} = -[H, xx^T]P.$$

(ii) If $\bar{H} = aH + bI$ has EVD $\bar{P}\bar{\Lambda}\bar{P}^T$,

$$\dot{\bar{H}} = \frac{1}{a}[\bar{H}, [\bar{H}, xx^T]],$$

$$\dot{\bar{P}} = -\frac{1}{a}[\bar{H}, xx^T]\bar{P}.$$

Proof:
(i) The proof of theorem 4.12(i) is also valid for the nonautonomous case.
(ii) $\dot{G} = a\dot{H} = a[(G - bI)/a, [(G - bI)/a, xx^T]] = [G, [G, xx^T]]/a$
□

For the case where H has only two distinct eigenvalues, 1 and 0, the nonautonomous double bracket flow relates to algorithm C (algorithm 7.10), through interpretation 7.22. In that case, the result of theorem 10.11 corresponds to corollary 7.33 (the noiseless stationary case) of theorem 7.29. With lemma 10.10, this is easily extended to any case where H has only two distinct eigenvalues. This will also be the most important case for the applications of section 10.5. However, the geometric interpretations 10.12 and 10.13 below suggest a generalization with some relevance for the applications, given as theorem 10.11. The generalization is valid for all matrices H with a principal eigenspace of dimension m. Eigenvalues corresponding to other eigenvectors need not be equal.

Theorem 10.11
Consider the system (10.9), $\dot{H} = [H, [H, xx^T]]$, where $x(t)$ is a continuously differentiable vector signal, evolving in an m-dimensional subspace $\mathcal{X} \subset \mathbb{R}^n$, and not converging to a smaller subspace of $\mathcal{X}$ (in the sense of interpretation 7.34).
Let the initial state H_0 have a principal eigenvalue λ with multiplicity m, and let no principal eigenvector of H_0 be orthogonal to $\mathcal{X}$.

Then, $H(t)$ converges to a matrix H^{st} with the same eigenvalues as H_0 and with principal eigenspace $\mathcal{X}$.

Proof: (also see interpretations 10.12 and 10.13 below.)
Let P be a set of eigenvectors as in lemma 10.10(i). Let α be the set of indices of the m eigenvectors with the principal eigenvalue λ, and let β contain the other indices. Let $D = (\lambda I - \Lambda_{\beta,\beta})^{\frac{1}{2}}$ and let $X \in \mathrm{St}(n, m)$ with $\mathrm{col}(X) = \mathcal{X}$ and $Y \in \mathrm{St}(n, n-m)$ with $\mathrm{col}(Y) = \mathcal{X}^\perp$. Note that $XX^T + YY^T = I$, $XX^T x = x$ and $Y^T x = 0$.

We will consider the evolution of the singular values of $Z = Y^T P_{\cdot,\beta} D$. (The main difference with theorem 7.29 is that we introduce D and consider the *orthogonal complements* of $\mathcal{X}$ and $\mathrm{col}(P_{\cdot,\alpha})$).

From $\dot{P} = -[H, xx^T]P = -[H - \lambda I, xx^T]P$, we have

$$\dot{P}_{\cdot,\beta} = [P_{\cdot,\beta} D^2 P_{\cdot,\beta}{}^T, xx^T] P_{\cdot,\beta}$$

and with $Y^T x = 0$,

$$\dot{Z} = Z(DP_{\cdot,\beta}{}^T x)(DP_{\cdot,\beta}{}^T x)^T.$$

With lemma 3.9(ii), one finds that

$$\dot{\sigma_i} = \sigma_i (v_i^T DP_{\cdot,\beta}{}^T x)^2, \qquad i = 1, \ldots, n-m, \tag{10.8}$$

for the singular values of Z. The vectors v_i denote corresponding singular vectors.

Therefore $\sigma \geq 0$ is non-increasing. Moreover σ cannot be 0, by the assumption that no principal eigenvector of H_0 be orthogonal to $\mathcal{X}$ (rank deficiency of Z would imply a principal eigenvector, orthogonal to $\mathcal{X}$). Because Y and $P_{\cdot,\beta}$ have orthogonal columns, the singular values σ_i of $Z = Y^T P_{\cdot,\beta} D$ are upper-bounded by the maximal diagonal entry of D. Therefore σ_i must converge and $v_i^T DP_{\cdot,\beta}{}^T x$ must go to 0. This implies that $P_{\cdot,\beta}{}^T x$ goes to 0.

Defining the SVD of $X^T P_{\cdot,\beta}$ as $\tilde{U}\tilde{\Sigma}\tilde{V}$, we also find with $x = XX^T x$ that $\tilde{\Sigma}\tilde{U}^T X^T x$ goes to 0, or with $\mu_i = \tilde{u}_i^T X^T x$, that $\mu_i \tilde{\sigma}_i$ goes to 0, $i = 1, \ldots, n$. If no μ_i goes to 0, all $\tilde{\sigma}_i$ must go to 0 and therefore all canonical angles between $\mathcal{X}$ and $\mathrm{col}(P_{\cdot,\beta})$ go to $\pi/2$, and the principal eigenspace $\mathrm{col}(P_{\cdot,\alpha})$ of H converges to $\mathcal{X}$.

As in interpretation 7.34 the condition that no μ_i goes to 0 can again be interpreted in a wide sense as 'x does not converge to a subspace of $\mathcal{X}$'. For, since $P_{\cdot,\beta}{}^T x$ goes to 0 also $\frac{d}{dt} X^T P_{\cdot,\beta}$ goes to 0, and with lemma 3.9(ii) the velocity of $\mathrm{col}(\tilde{U}_\eta)$ (where η contains the indices i for which $\mu_i \not\to 0$) goes to 0. As x converges to $\mathrm{col}(\tilde{U}_\eta)^\perp$, x converges to a subspace with velocity going to 0.
□

Interpretation 10.12
In the local basis of lemma 3.15, the component affecting eigenvectors p_i and p_j of H, has angular velocity

$$\omega_{i,j} = (\lambda_i - \lambda_j)(p_i^T x)(p_j^T x).$$

Inspecting the sign of $\omega_{i,j}$, one finds that the eigenvector with the largest eigenvalue, say p_i (or $-p_i$), rotates towards x (figure 10.1).

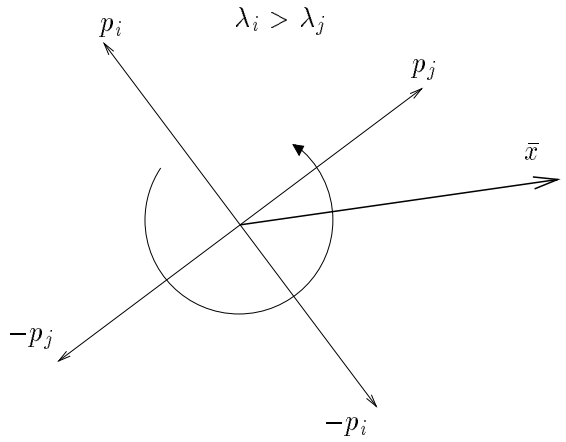


Figure 10.1: Rotation of p_i and p_j, $\lambda_i > \lambda_j$. The direction of p_i rotates towards $\bar{x}$, the projection of x on $\text{span}\{p_i, p_j\}$.

Interpretation 10.13
Consider again the cylindrical coordinate picture of the double bracket flow in $\mathcal{S}_{symm}(2)$ (figure 4.3), to be read as 'N attracts the point H on the circle to a point in the same vertical plane as N'. Matrices N in one half attract H to one equilibrium. Matrices N in the other half attract H to the other equilibrium. Time-varying N staying at one side still attract N to the same equilibrium. In the case of the theorem, one should think of the circle as the manifold $\mathcal{I}_{H_0}$ of matrices with the same spectrum as H_0. And the normal space through H^{st} contains all matrices commutable with H^{st} (proposition 2.19(ii)). It is now divided in more than two parts, corresponding to different ways in which the eigenvalues of H^{st} can be matched with the eigenvalues of N. All $N = xx^T$, with $x \in \mathcal{X}$, have the eigenvalues in the same order as H^{st} (though not strictly due to repeated eigenvalues). Constant xx^T would only attract a one-dimensional subspace of the principal eigenspace of H to lie in $\mathcal{X}$ (making H commute already). But if x does not converge to a subspace of $\mathcal{X}$, H converges to H^{st}.

Remark 10.14
This interpretation can be formalized by showing that $\|H - XX^T\|_F^2$ is steadily non-increasing during evolution of H, by analogy with the proof of the double bracket flow (theorem 4.12). One can show (using $H = \lambda I - P_{\cdot,\beta} D^2 P_{\cdot,\beta}{}^T$ and $XX^T = I - YY^T$) that $\|H - XX^T\|_F^2$ is equal up to a constant to $-2\|Y^T P_{\cdot,\beta} D\|_F^2$. The singular values of $Y^T P_{\cdot,\beta} D$ are only needed in the proof of theorem 10.11 to settle some details.

10.3 Stabilization of the spectrum

Since analog realizations are necessarily inaccurate, isospectrality will not be guaranteed by any practical realization. Any deviation of the spectrum, simply results in the (exact) system preserving the new spectrum. Also round-off errors or discretization errors in discrete implementations would accumulate. We introduce some modifications (10.11) to overcome this loss. These modifications do not belong to the class of uniform parallel systems of definition 5.1. In the next section we overcome this problem as well.

To stabilize the spectrum we will introduce an extra 'radial' velocity term (lemma 3.15), influencing the eigenvalues and not the eigenvectors.

Theorem 10.15
Let $v(\lambda)$ be a polynomial in λ and let $v'(\lambda)$ be its derivative. Consider the algorithm

$$\dot{H} = v'(H), \tag{10.9}$$

where H is a symmetric matrix,

Then,
(i) the eigenvectors of H are constant and the eigenvalues λ_i of H all change according to the same equation

$$\dot{\lambda}_i = v'(\lambda_i) \qquad\qquad i = 1, \ldots, n, \tag{10.10}$$

which describes the (one-dimensional) gradient ascent flow of $v(\lambda)$.

(ii) The algorithm is the gradient ascent flow of $\mathrm{Tr}(v(H))$.

(iii) If $v(\lambda)$ is upper-bounded, all λ_i converge to a local maximum of $v(\lambda)$. Each local maximum is attracted from within the attraction interval between the nearest left local minimum (or $-\infty$) and the nearest right local minimum (or $+\infty$).

(iv) When started from an initial state H_0 with eigenvalues $\lambda_1, \ldots \lambda_n$ in the attraction intervals (see (iii)) of the local maxima $\lambda_1^{(M)}, \ldots, \lambda_n^{(M)}$, the system

$$\dot{H} = [H, [H, N]] + v'(H) \tag{10.11}$$

converges to an equilibrium of the system

$$\dot{H} = [H, [H, N]] \tag{10.12}$$

on $\mathcal{I}_{\Lambda^{(M)}}$, where $\lambda^{(M)} = \mathrm{diag}\{\lambda_1^{(M)}, \ldots, \lambda_n^{(M)}\}$.

Proof: (i) This follows from lemma 3.9, with $M = P^T \dot{H} P = P^T v'(P \Lambda P^T) P = v'(\Lambda)$.

(ii) Let $v(H) = \sum_{i=0}^{n} a_i H^i$. Then $\frac{d}{dt}v(H) = \sum_{i=1}^{k}\sum_{j=0}^{i} H^j \dot{H} H^{i-j-1}$. And with lemma 2.11(vii), $\frac{d}{dt}\operatorname{Tr}(v(H)) = \operatorname{Tr}(\sum_{i=1}^{k}(i-1)H^{i-1}\dot{H}) = \langle v'(H), \dot{H}\rangle$. Property (ii) follows from the definition of the gradient (definition 3.21).
(iii) This follows directly from the evolution of the eigenvalues, as described by (10.10).
(iv) Since $\operatorname{Tr}(v(H))$ is not altered by an isospectral change of H, it follows from (ii) that it is still a weak Lyapunov function of (10.11) (in the sense that it does not decrease during evolution and has compact superlevel sets. This last property follows from the fact that the eigenvalues are bounded by (i) if $v(\lambda)$ is an upper-bounded polynomial. Moreover, $\frac{d}{dt}v(H) = 0$ only if $v'(H) = 0$. Therefore, the limit set of (10.11) is a subset of $\{H \mid v'(H) = 0\}$. As the term $v'(H)$ vanishes, one can expect convergence to the same equilibrium as when starting on the manifold. This is confirmed by considering the function $\langle H^{(M)}, N\rangle$, where $H^{(M)}$ takes the eigenvectors of H and the eventual eigenvalues $\Lambda^{(M)}$ (in the same order as in H_0): $H^{(M)} = P\Lambda^{(M)}P^T$, with P satisfying

$$\dot{P} = -[H, N]P,$$

as in theorem 4.12.
One finds

$$\begin{aligned}
\langle H^{(M)}, N\rangle &= \langle [H^{(M)}, [H, N]], N\rangle \\
&= \langle [H^{(M)}, N], [H, N]\rangle \\
&= \langle [\Lambda^{(M)}, P^T N P], [\Lambda, P^T N P]\rangle \\
&= \langle C_{\Lambda^{(M)}} . * (P^T N P), C_\Lambda . * (P^T N P)\rangle,
\end{aligned}$$

with $C_{\Lambda^{(M)}}$ and C_Λ as in lemma 2.11(ii). It follows from (i) that the eigenvalues of H retain their original order. Therefore $\frac{d}{dt}\langle H^{(M)}, N\rangle \geqslant 0$ and can only be equal to 0 if $[H, N] = 0$. This proves (iv).
□

Remark 10.16
Also the stability analysis of theorem 4.12 can be copied with minor changes. One finds the same stable equilibria on $\mathcal{I}_\Lambda^{(M)}$ as for the double bracket flow.

In other words, the second term in $\dot{H} = [H, [H, N]] + v'(H)$ is a radial velocity component (see section 3.2.3), responsible for ascending $\operatorname{Tr}(v(H))$, the first term is an isospectral velocity component turning H towards N. The system generically converges to a matrix with eigenvalues which locally maximize $v(\lambda)$, associated to eigenvectors of N in the same way as for the double bracket flow. The initial matrix H_0 determines which local maxima will be attained. The presence of an eigenvalue of H_0 in the attraction interval of the local maximum $\lambda_i^{(M)}$ of $v(\lambda)$ for the system (10.10) will result in an eigenvalue $\lambda_i^{(M)}$ for the equilibrium state. If all eigenvalues of H_0 are local maxima of $v(\lambda)$ the second term of (10.11) can restore small deviations from the isospectral trajectory, caused by an inaccurate implementation. The case with two eigenvalues λ_1 and λ_2, for

example, can be modified in this sense by choosing $v(\lambda) = -\frac{1}{2}(\lambda-\lambda_1)^2(\lambda-\lambda_2)^2$, resulting in $v'(H) = -(H-\lambda_1 I)(H-\lambda_2 I)(2H-\lambda_1-\lambda_2)$.

Remark 10.17
Without appropriate choice of the initial state, this modification only allows to restrict the possible values which each single eigenvalue can reach, to the maxima of the function $v(\lambda)$. In order to fix the spectrum as a whole, (i.e. to fix the stable spectra of (10.9) up to within a permutation only), gradient ascent flows of more complex functions of the spectrum are needed. For example, the system

$$\dot{H} = -\,\mathrm{grad}_H \sum_{k=1}^{n} (C_k(H) - C_k(\Lambda))^2,$$

where $C_k(H)$, $k = 1, \ldots, n$ are the coefficients of the characteristic polynomial of H, $\det(H - xI) = \sum_{k=1}^{n} C_k(H) x^k$. This results in radial velocities, making the coefficients $C_k(H)$ converge to the coefficients $C_k(\Lambda)$, causing H to get the same characteristic polynomial and therefore the same spectrum as Λ. However the computational effort is too high in view of the fact that we only need to get the eigenvalues of H in the right attraction interval for (10.11) to converge to a fixed spectrum.

10.4 Stabilization of the spectrum with laws of type I

We now concentrate again on (10.11) for the case where $\Omega = [H, xx^T]$ as in algorithm 10.9. The calculation on the network of the powers of H in $v'(H)$ would need some further sophistication, for example by calculating the column $H^{k+1}_{\cdot i}$ of H^{k+1} as $HH^k_{\cdot i}$ for all $i = 1, \ldots, n$. We can however introduce modifications which reduce the calculations again to a system of type I of definition 5.1. To this end, we introduce an extra 'input' signal, that can simply be a noise source, or any signal that does not converge to a subspace of $\mathbb{R}^n$. By this mechanism the spatial complexity of the stabilization terms is reduced, at the expense of an increased time complexity (slower convergence of the spectrum). The way in which this is obtained is susceptible to generalization. Actually, the nonautonomous case of the double bracket flow studied above (algorithm 10.9) can be seen as a lower complexity version of the double bracket flow. Other low complexity versions can be thought of also.

Theorem 10.18
Let $v(\lambda)$ be an upper-bounded polynomial and $v'(\lambda)$ its derivative. Let $v_1(\lambda)$ and $v_2(\lambda)$ be two polynomials satisfying $v_1(\lambda)v_2(\lambda) = v'(\lambda)$. Let $s(t)$ be a continuouly differentiable vector signal evolving in $\mathbb{R}^n$. Let $\Omega \in \mathcal{S}_{skew}(n)$.

Consider the algorithm

$$\begin{aligned} \dot{H} &= [H,\Omega] + v_1(H)ss^T v_2(H) + v_2(H)ss^T v_1(H), \\ &= [H,\Omega] + (v_1(H)s)(v_2(H)s)^T + (v_2(H)s)(v_1(H)s)^T, \end{aligned} \tag{10.13}$$

then
(i) H ascends $\mathrm{Tr}(v(H))$ (in a non-steepest way).

(ii) each eigenvalue λ_i of H satisfies

$$\dot{\lambda_i} = 2v'(\lambda_i)(p_i^T s)^2, \tag{10.14}$$

where p_i is a corresponding eigenvector.

(iii) Let $s(t)$ evolve in an m-dimensional subspace $\mathcal{S}$ of $\mathbb{R}^n$, without converging to a lower-dimensional subspace, then, each eigenvalue λ of H, corresponding to an eigenvector p, which does not converge to a direction orthogonal to $\mathcal{S}$, converges to a local maximum of $v(\lambda)$, from within the attraction interval between the nearest left local minimum (or -∞) and the nearest right local minimum (or $+\infty$).

(iv) Let $\Omega = [H, xx^T]$ with $x(t)$ as in theorem 10.11. Let $v_1(\lambda_i^{(M)}) = 0$ for any local maximum $\lambda_i^{(M)}$ of $v(\lambda)$. And let the initial state H_0 have eigenvalues $\lambda_1, \ldots \lambda_n$ in the attraction intervals (see (iii)) of the local maxima $\lambda_1^{(M)}, \ldots, \lambda_n^{(M)}$, with m of them in the attraction interval around $\lambda = \max_i \lambda_i^{(M)}$.
Let $s(t)$ be a continuously differentiable signal evolving in $\mathbb{R}^n$ which does not converge to a subspace of $\mathbb{R}^n$.

Then, if

$$\dot{H} = [H,[H,xx^T]] + v_1(H)ss^T v_2(H) + v_2(H)ss^T v_1(H), \tag{10.15}$$

the eigenvalues of H converge to $\lambda_1^{(M)}, \ldots, \lambda_n^{(M)}$, and the principal eigenspace of H generically converges to $\mathcal{X}$.

Proof:

$$\begin{aligned} \text{(i) } \tfrac{d}{dt}\mathrm{Tr}(v(H)) &= \langle \mathrm{grad}\ \mathrm{Tr}(v(H)), \dot{H} \rangle \\ &= \langle v'(H), [H,\Omega] + v_1(H)ss^T v_2(H) + v_2(H)ss^T v_1(H) \rangle \\ &= \langle [H, v'(H)], \Omega \rangle + \langle v'(H), v_1(H)ss^T v_2(H) + v_2(H)ss^T v_1(H) \rangle \\ &= 2s^T v'(H)^2 s \\ &\geqslant 0. \end{aligned}$$

This proves that $\mathrm{Tr}(v(H))$ is ascended.
(ii) This follows from lemma 3.9, with

$$M = P^T \dot{H} P = [\Lambda, P^T \Omega P] + (v_1(\Lambda)P^T ss^T P v_2(\Lambda) + v_2(\Lambda)P^T ss^T P v_1(\Lambda)).$$

For distinct eigenvalues, one finds

$$\dot{\lambda_i} = m_{i,i} = p_i^T (v_1(H)ss^T v_2(H) + v_2(H)ss^T v_1(H))p_i = 2v'(\lambda_i)(p_i^T s)^2.$$

The case of multiple eigenvalues can be treated like the case of multiple singular values in theorem 7.29.

(iii) As the eigenvalues follow (10.14),

$$\dot{v}(\lambda_i) = v'(\lambda_i)^2 (p_i^T s)^2 \geqslant 0.$$

It follows that $v(\lambda_i)$ is steadily non-decreasing. Since $v(\lambda_i)$ is bounded, $\dot{v}(\lambda_i)$ must go to 0. If $p_i(t)$ does not converge to a direction orthogonal to $\mathcal{S}$, $p_i^T s$ cannot converge to 0, for this would mean that $s(t)$ converges to a subspace of $\mathcal{S}$. We conclude that $v'(\lambda_i)$ must go to 0. This ensures that $\lambda_i(t)$ converges to a local maximum of $v(\lambda_i)$.
(iv) Because $s(t)$ does not converge to a subspace of $\mathbb{R}^n$, it follows from (iii) that all eigenvalues converge to $\lambda_1^{(M)}, \ldots, \lambda_n^{(M)}$. Because in that case the terms $v_1(H)ss^T v_2(H) + v_2(H)ss^T v_1(H)$ vanish, one can expect that (10.15) converges to the same equilibrium as $\dot{H} = [H, [H, xx^T]]$.

Let X, Y, P, Λ, D, α and β be defined as in theorem 10.11 and consider the evolution of $\langle H, XX^T \rangle$.

$$\begin{aligned}
\tfrac{d}{dt}\langle H, XX^T\rangle &= \langle H^2 xx^T - 2Hxx^T H + xx^T H^2 \\
&\quad + v_1(H)ss^T v_2(H) + v_2(H)ss^T v_1(H), XX^T \rangle \\
&= 2(x^T H^2 x - x^T HXX^T Hx + s^T v_1(H) XX^T v_2(H) s) \\
&= 2(x^T HYY^T Hx + s^T v_1(H) XX^T v_2(H) s) \\
&= 2\|Y^T Hx\|^2 + 2s^T v_1(H) XX^T v_2(H) s.
\end{aligned}$$

Because $\langle H, XX^T \rangle$ is bounded, $\|Y^T Hx\|^2 \geqslant 0$, and $v_1(H) \rightarrow 0$, $\|Y^T Hx\|$ must converge to 0.

With $Y^T x = 0$, we also find that $\|Y^T(\lambda I - H)x\|$ goes to 0. With $\lambda I - H = P_{\cdot,\alpha}(\lambda I - \Lambda_{\alpha,\alpha}) P_{\cdot,\alpha}{}^T + P_{\cdot,\beta}(\lambda I - \Lambda_{\beta,\beta}) P_{\cdot,\beta}{}^T$, and observing that $\lambda I - \Lambda_{\alpha,\alpha}$ goes to 0, it also follows that $\|Y^T P_{\cdot,\beta} D^2 P_{\cdot,\beta}{}^T x\|$ goes to 0. If no principal eigenvector of H is orthogonal to $\mathcal{X}$, $Y^T P_{\cdot,\beta}$ is of full rank, which implies that $P_{\cdot,\beta}{}^T x \rightarrow 0$. We cannot exclude completely however that the (vanishing) isospectral components of the terms with s steer an eigenvector of H to $\mathcal{X}^\perp$, (however, if this is possible at all, it would be a hard problem to design such an input s). As in the proof of theorem 10.11, we can again introduce the SVD of $X^T P_{\cdot,\beta} = \tilde{U}\tilde{\Sigma}\tilde{V}$ to prove that the principal eigenspace of H converges to $\mathcal{X}$, if no component of x 'converges to a subspace' of $\mathcal{X}$.
□

Remark 10.19
If in (iv) s is replaced by x, resulting in

$$\dot{H} = [H, [H, xx^T]] + v_1(H)xx^T v_2(H) + v_2(H)xx^T v_1(H),$$

the terms $v'(H)xx^T + xx^T v'(H)$ still cause all eigenvalues to ascend $v(\lambda)$, but the local maximum need not be reached, for eigenvectors converging to a direction orthogonal to $\mathcal{X}$. If we can assume that the eigenvectors with the largest eigenvalues

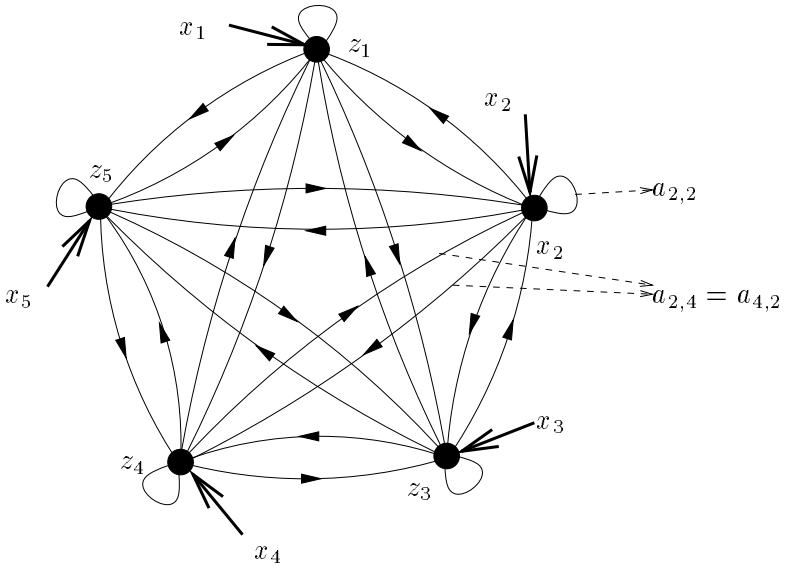


Figure 10.2: Fully connected recurrent network with state z, weights $a_{i,j}$, and external teaching signal x.

converge to directions in $\mathcal{X}$, the largest eigenvalues will converge to a local maximum of $v(\lambda)$. If the initial state H_0 has m eigenvalues in the attraction interval $(\lambda_m, +\infty)$, where $\lambda_m = \max\{\lambda \mid v \text{ is locally minimal at } \lambda\}$, they will converge to $\lambda_M = \max\{\lambda \mid v \text{ is locally maximal at } \lambda\}$ and the corresponding eigenspace will be $\mathcal{X}$, by theorem 10.11. However, when the eigenvalues are far enough from the local maxima of $v(\lambda)$, the isospectral components of $v_1(H)xx^Tv_2(H) + v_2(H)xx^Tv_1(H)$ may dominate over $[H,[H,xx^T]]$. The exact attraction region of the stable equilibria $H(\infty)$ for which $\mathcal{X}$ is an eigenspace with eigenvalue λ_M depends on the choice of v_1 and v_2. Also see simulation example 3 (figure 10.5).

10.5 Adaptation laws for memory

We are now ready to discuss adaptation laws for storing patterns in BSB-type (or related) associative memory. Patterns are shown to the network by a *teaching signal* $x(t)$. The weights of the network adapt so as to store the shown patterns. At the end of this section we will also show how the teaching signal can be realized by using the external input b of the network models, described in section 10.1.

The basic idea is contained in lemma 10.4(ii). A pattern p can be made an asymptotically stable equilibrium by adapting A so as to make p an eigenvector

with eigenvalue $\rho > 1$. In addition theorem 10.8 shows that the number of unwanted stable states is kept low by providing strongly negative eigenvalues in orthogonal directions. The algorithms discussed below exploit the theory of the previous sections to attract eigenvectors with high eigenvalues towards a pattern, shown by an input signal. When showing patterns in the subspace $\mathcal{X} = \text{span}(\mathcal{P})$ of pattern space, the systems converge to the same weight matrix as obtained by design algorithm 10.6.

We start with a straightforward application of theorem 10.11:

Algorithm 10.20

$$\dot{A} = A^2 x x^T - 2 A x x^T A + x x^T A^2.$$

Corollary 10.21 (of theorem 10.11)
Let $\mathcal{P} = \{p_i\}_{i=1}^m \subset \{-1, 1\}^n$ be a set of binary patterns, generating an m-dimensional subspace $\mathcal{X} = \text{span}(\mathcal{P}) \subset \mathbb{R}^n$. Let $x(t)$ be a signal evolving in $\mathcal{X}$, and not converging to a smaller subspace of $\mathcal{X}$ in the sense of interpretation 7.34. Let the initial matrix $A(0)$ have an eigenvalue $\rho > 1$ with multiplicity m and an eigenvalue $-\tau < 0$ with multiplicity $n - m$, and let no eigenvector of $A(0)$ be orthogonal to $\mathcal{X}$.

Then the weight matrix A, following adaptation algorithm 10.20, converges to the matrix $A_{\rho,\tau}$, specified by design algorithm 10.6.

The proof follows immediately from theorem 10.11.

To supply an appropriate teaching signal $x(t)$, there are numerous possibilities. An easy way consists in supplying a continuous signal $x(t)$, which keeps randomly supplying one of the patterns p_i for a short time period and changing to another pattern by some interpolation. See simulation example 1 in section 10.7.

To supply an initial state without eigenvectors orthogonal to $\mathcal{X}$, we can start at a time $t^- < 0$ with a state $A(t^-) = \text{diag}\{\rho, \ldots, \rho, -\tau, \ldots, -\tau\}$, with m times ρ on the diagonal and $n - m$ times $-\tau$, and supply a random signal $x(t) \in \mathbb{R}^n$ (not confined to $\mathcal{X}$) for $t^- < t < 0$. The probability of ending up at $t = 0$ in an inappropriate initial state $A(0)$ is 0. Another problem is that the dimension of $\text{span}(\mathcal{X})$ is needed to determine the initial value of A. If it is too large, A can stabilize more unwanted patterns, if it is too small, A will not converge during adaptation. Choosing a value of m can be regarded as giving the memory a fixed capacity. A possible way to avoid this lies in restarting with a larger dimension when there is no convergence. Other solutions consist in heuristic combinations with the simple adaptation laws presented in section 10.6, and would lead us away from the theory presented in the previous sections.

Because A has only two distinct eigenvalues ρ and $-\tau$, $(A-\rho I)(A+\tau I) = 0$. This

allows us to avoid the need of A^2x in adaptation algorithm 10.20 by recasting it as

Algorithm 10.22

$$\dot{A} = -(A - \rho I)xx^T(A + \tau I) - (A + \tau I)xx^T(A - \rho I).$$

Or, we can implement an algorithm on a matrix H with eigenvalues 1 and -1. Since in that case $H^2 = I$, we obtain $\dot{H} = xx^T - 2(Hx)(Hx)^T + xx^T$, or by absorbing a factor 2 in the unit of time,

Algorithm 10.23

$$\dot{H} = xx^T - (Hx)(Hx)^T.$$

This is the simplest attainable form.

However, $\rho = 1$ and $-\tau = -1$ are not appropriate as eigenvalues for design algorithm 10.6, since the principal eigenvalue ρ should be greater than 1. But, when A is defined as $A = H + \delta I$ with some small $\delta > 0$, it has eigenvalues $\rho = 1 + \delta$ and $-\tau = -(1 - \delta)$, yielding $\tau' = (\tau + 1)/(\rho - 1) = 2/\delta - 1$, which can be made arbitrarily large. In the following we will work with general eigenvalues ρ and $-\tau$ again, but it should be noted that the discussed technique can always be applied to obtain a realization with a flexible value of τ'.

A problem with the simplified version (10.23) is that it only preserves specific eigenvalues (-1 and 1). Applying theorem 10.18(ii), once for $v_1(\lambda) = v_2(\lambda) = I$ and once for $v_1(\lambda) = v_2(\lambda) = \lambda$, and with $s(t) = x(t)$ and $\Omega = 0$, we find that the eigenvalues follow

$$\dot{\lambda} = (1 - \lambda^2)(p^T x)^2.$$

Consequently, possible deviations of the eigenvalue 1 are restored but deviations of the eigenvalue -1 are amplified. Eigenvalues less than -1 are only prevented from diverging to $-\infty$ by $(p^T x)^2$ converging to 0 (fast enough). If we want to avoid the use of $H^k x$ for $k \geq 2$, some results can still be obtained by allowing first order polynomial terms $\alpha H + \beta I$. However, we will not discuss this further, as it is more difficult to study this case rigorously, and it would need some extension of the theory, presented in the previous sections.

The spectrum can be stabilized, by applying the theory, developed in sections 10.3 and 10.4. By choosing $v'(A) = -\kappa(A - \rho I)(A + \tau I)(A - \alpha I)$ in theorem 10.18 where $-\tau < \alpha < \rho$ and κ is some positive constant determining the relative contribution of the new term, we can stabilize the eigenvalues ρ and $-\tau$. If we choose, for instance, $v_1(A) = -\kappa(A - \rho I)(A + \tau I)$ and $v_2(A) = (A - \alpha I)$, we obtain

Algorithm 10.24

$$\begin{aligned}\dot{A} &= A^2xx^T - 2Axx^TA + xx^TA^2 \\ &- \kappa(A-\rho I)(A+\tau I)ss^T(A-\alpha I) \\ &- \kappa(A-\alpha I)ss^T(A-\rho I)(A+\tau I).\end{aligned}$$

Corollary 10.25 (of theorem 10.18)
Let $\mathcal{P} = \{p_i\}_{i=1}^m \subset \{-1,1\}^n$ be a set of binary patterns, generating an m-dimensional subspace of $\mathbb{R}^n$, $\mathcal{X} = \text{span}(\mathcal{P})$.
Let $x(t)$ be a signal evolving in $\mathcal{X}$, without converging to a lower-dimensional subspace and let $s(t)$ be a signal evolving in $\mathbb{R}^n$, not converging to a subspace of $\mathbb{R}^n$.
Let the initial matrix $A(0)$ have m (not necessarily distinct) eigenvalues greater than α and $n-m$ (not necessarily distinct) eigenvalues less than α.

Then the weight matrix A, following adaptation algorithm 10.24, converges to the matrix $A_{\rho,\tau}$, specified by design algorithm 10.6.

Proof:
The proof follows directly from theorem 10.18(iii,iv) for the case $v'(\lambda) = -\kappa(\lambda-\rho)(\lambda+\tau)(\lambda-\alpha)$. By this choice of $v'(\lambda)$, $v(\lambda)$ is upper-bounded and has local maxima at $\lambda = \rho$ and $\lambda = -\tau$ and a local minimum in between at $\lambda = \alpha$. From theorem 10.18(iv), we know that the eigenvalues of A, less than the local minimum α of v, converge to the local maximum $-\tau$ and the eigenvalues greater than α converge to the other local maximum ρ, while the eigen*vectors* converge as for adaptation algorithm 10.20.
□

As discussed in remark 10.19, when x is used instead of s in adaptation algorithm 10.24, the behavior depends on the isospectral components of the term $v_1(A)xx^Tv_2(A) + v_2(A)xx^Tv_1(A)$. Like in (10.23), we have studied the case for a matrix H with eigenvalues 1 and -1. When $\rho = \tau = \kappa = 1$ and $\alpha = 0$, the algorithm can be rewritten as

Algorithm 10.26

$$\dot{H} = (H+H^2)xx^T(I-H) + (I-H)xx^T(H+H^2).$$

We have not proved theoretical results for an arbitrary initial value of H. However, if H has a principal eigenvalue 1 with multiplicity m, it can be shown, (using similar ideas as in the proof of theorem 10.11), that the isospectral components of (10.26) still make the principal eigenspace converge towards $\mathcal{X}$. From theorem 10.18(iii) this implies that the eigenvalue 1 is stable, while the other eigenvalues need not converge to a local maximum of $v(\lambda)$. Simulation 3 gives

an example (Fig. 10.5). From the proof of theorem 10.8 one can easily see that it is not necessary that all negative eigenvalues are equal, as long as they are small enough, to obtain the behavior intended by design algorithm 10.6. Even when initial states with deviations of the eigenvalue 1 are allowed, all our simulations still show the intended effect, but a proof, if possible, seems to be necessarily more involved. Also observe that the eigenvalues of the initial state can be brought closer to 1 and -1 by first supplying a random signal $x(t)$, evolving in $\mathbb{R}^n$.

Finally, we want to say more about the relation between the operation phase (recall) of the associative memory and the adaptation phase (storage). As presented above, the adaptation law describes a separate adaptation phase, during which a teaching signal is presented and a weight matrix A is obtained. We can also think of the adaptation phase as taking place *simultaneously* with the operation phase, but on a slower time scale. The function of $x(t)$ can then be executed by the state z of the neural network.

Algorithm 10.27

$$\begin{aligned} \dot{z} &= -z + f(Az + b), \\ \dot{A} &= \tau[A, [A, zz^T]], \end{aligned}$$

where $\tau > 0$ is a small positive scalar.

To present a new pattern p, z can be forced to p by supplying a strong external input b, proportional to p. The adaptation law then makes the weight matrix A adapt to make b unnecessary for the stabilization of the pattern. If during the recall the system remains long enough at or near an equilibrium, the memory contents are not destroyed by the constant presence of the adaptation mechanism.

In this way we have obtained a system whose adaptation mechanism always *adapts to confirm the present state as a desirable state*. The system is taught new things by forcing its state with an external input.

10.6 Comparison with other adaptation laws

In this section we compare the adaptation laws described above, with two other adaptation laws. These, as well as the new adaptation laws presented above, can be regarded as exploiting the same basic qualitative idea that, according to lemma 10.4(ii), a pattern p can be stored by changing A gradually to make the pattern an eigenvector of the weight matrix with eigenvalue $\rho > 1$. Note that it

is not necessary for p to be an eigenvector, and that when A is adapted gradually, the stability conditions (10.6) may be satisfied long before p is an eigenvector.

The simplest adaptation law is the well known Hebbian law

Algorithm 10.28

$$\dot{A} = xx^T.$$

This law, also called autocorrelation learning, is better known in the discrete form $A^{[k+1]} = A^{[k]} + x^{[k]}x^{[k]T}$. If the teaching signal x is kept constant, $x(t) = p$, for $t > t_0$ for some $t_0 > 0$, $A(t) = A(t_0) + (t - t_0)pp^T$ and the matrix A is changing to make p an eigenvector in the sense that $A(t)p = A(t_0)p + ((t - t_0)p^Tp)p$ in which the relative contribution of the first term is steadily decreasing. It is easily shown, that p becomes a stable equilibrium pattern by letting $x(t)$ equal p for a long enough time interval. However, as the eigenvalues of A are steadily non-decreasing, the adaptation law tends to stabilize many more patterns than intended, and old patterns can nevertheless be forgotten when learning new ones, in a way which is difficult to control by means of the teaching signal.

Another adaptation law is related to linear perceptron learning. The linear perceptron adaptation law trains a matrix A to map a pattern x onto a given pattern $Ax = y$, by changing A according to

$$\dot{A} = -(Ax - y)x^T.$$

This gives A a velocity in the direction opposite to the gradient of $\|Ax - y\|^2$. If we want x to be an eigenvector with eigenvalue $\rho > 1$, we can train A to map x on ρx. When the gradient is calculated in $\mathcal{S}_{symm}(n)$ we obtain

Algorithm 10.29

$$\dot{A} = -(A - \rho I)xx^T - xx^T(A - \rho I).$$

This is a special case of the system (10.13) described in theorem 10.18, with $s(t) = x(t)$, $v(\lambda) = -\frac{1}{2}(\lambda - \rho)^2$, $v_1(\lambda) = -(\lambda - \rho)$ and $v_2(\lambda) = 1$. If $x(t) = p$ is kept constant, A converges to a matrix for which p is an eigenvector with eigenvalue ρ. If $x(t)$ evolves in some subspace $\mathcal{X}$ without converging to a lower-dimensional subspace, it follows from theorem 10.18(iii) that $\mathcal{X}$ becomes an eigenspace with eigenvalue ρ. In the local basis of lemma 3.15, one can also interpret the evolution of the eigenvectors as in interpretation 10.12. For the rotation in the plane of two eigenvectors one finds that the one with its eigenvalue closest to ρ rotates towards x.

Adaptation algorithm 10.29 has as an advantage over adaptation algorithm 10.28 that it converges, and that in the limit we can be sure that all patterns in a subspace described by $p(t)$ are stable. However, since the eigenvalues follow (10.14), all eigenvalues evolve in the direction of ρ. Therefore, many more unwanted patterns can be stabilized than by design algorithm 10.6. If one thinks of $\mathcal{P}$ as a set which can vary slowly in time (for the case where the adaptation phase and the operation phase are organized simultaneously as discussed at the end of section 10.5), all eigenvalues could end up at ρ, making A stabilize all patterns.

Compared with adaptation algorithms 10.28 and 10.29, the new algorithms, presented in this chapter, offer the advantage of preserving the negative eigenvalues. This avoids the problem of stabilizing many more patterns than intended, in an uncontrolled way. Moreover the set of stable patterns is characterized mathematically through theorem 10.8. When thinking of a slowly time-varying set $\mathcal{P}$, the effect of the preserved negative eigenvalues can be regarded as an ability to forget old patterns, that are not in $\mathcal{P}$ anymore.

Finally, as pointed out already the presented algorithms are also related with the neural adaptation algorithms of [85, 91], discussed in chapter 7, and normally not presented in the context of recurrent networks for associative memory applications. We could also think of realizing A as WDW^T. For any system, that can be written in the form

$$\dot{A} = KA + AK^T, \tag{10.16}$$

where $K \in \mathbb{R}^{n \times n}$, can be implemented as

$$\begin{array}{rcl} \dot{W} & = & KW, \\ A & = & WDW^T, \end{array} \tag{10.17}$$

as can be verified from $\dot{A} = \dot{W}DW^T + WD\dot{W}^T$. The calculation of $Az = WDW^Tz$ is done by forward and backward propagation over a one layer network, the multiplication with D is done in the second layer. In this case, isospectral terms $[A, \Omega]$ have the form (10.16) with $K = \Omega^T$, and we only need to stabilize the orthogonality of W, which is done automatically by systems like the cross product subspace trackers discussed in chapter 7. However the need to calculate WDW^Tz makes this type of system more complex than the algorithms of chapter 7.

10.7 Simulation examples

Finally, we discuss some simulation examples.

Simulation example 1
Fig. 10.3 shows a simulation of adaptation algorithm 10.20, for $n = 5$ neurons

and $\mathcal{P} = \{[1\,1\,1\,1\,-1]^T, [1\,1\,1\,-1\,1]^T, [1\,1\,-1\,1\,1]^T\}$, giving $m = \dim \mathcal{X} = 3$. The teaching signal $x(t)$ is made to present the patterns of $\mathcal{P}$ in a cyclic way, and going from pattern p_{old} to pattern p_{new}, in a time interval $[iu, (i+1)u]$, where i is integer, by

$$p(t) = p_{old} \cos^2((\pi/2)(t/u - i)) + p_{new} \sin^2((\pi/2)(t/u - i)). \tag{10.18}$$

This way, $x(t)$ moves in straight lines from one pattern to another and is differentiable in time. (Alternatively one could generalize the given theorems as to allow piecewise continuous input). The choice of $u = 0.1$ is not crucial. It is best to choose u small enough in order to show new information as soon as the principal eigenspace $\mathcal{H}$ is attracted close enough to $\mathcal{X}$. However, if one wants to minimize the number of times a pattern is to be presented, u should be chosen large enough. The initial state $A_0 = \text{diag}\{2, 2, 2, -2, -2\}$ is chosen to have an eigenvector orthogonal to $\mathcal{X}$. Between $t = -1$ and $t = 0$, a random teaching signal is supplied, to avoid the inappropriate initial state. Fig. 10.3 shows the evolution of the entries of A, and of the maximal angle ψ between the principal eigenspace $\mathcal{H}$ and $\mathcal{X}$. From $t = 1.36$ on, A stabilizes the intended binary patterns and no others.

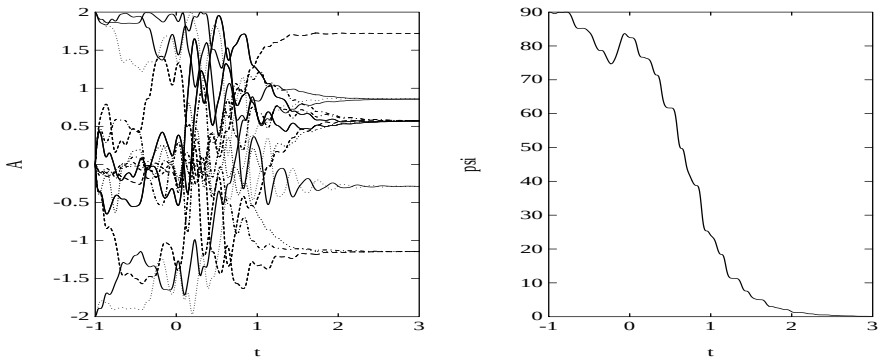


Figure 10.3: Simulation of adaptation algorithm 10.20 for 5 neurons and 3 patterns, with $\rho = \tau = 2$, supplied with a random signal between $t = -1$ and $t = 0$

Simulation example 2

Fig. 10.4 shows a simulation of a simple discretized version (introducing deviations to be corrected by the stabilization mechanism) of adaptation algorithm 10.24 for $n = 10$ and $m = 5$, $\rho = \tau = 1$, and $\alpha = 0$.

$$\begin{aligned} H(t+1) &= H(t) \\ &+ 0.01\{H(t)^2 x(t) x(t)^T - 2H(t)x(t)x(t)^T H(t) + x(t)x(t)^T H(t)^2 \\ &+ (I - H(t)^2) s(t) s(t)^T H(t) + H(t) s(t) s(t)^T (I - H(t)^2)\}. \end{aligned}$$

The patterns have been chosen randomly, and the initial state for the matrix H is equal to $\mathrm{diag}\{-0.5, -0.75, -1, -1.25, -1.5, 0.5, 0.75, 1, 1.25, 1.5\}$. The patterns are presented cyclically, and each pattern is presented during 2 consecutive steps (without interpolation between the patterns). The components of the noise $s(t)$ are independent random variables, uniformly distributed between -1 and 1. Fig. 10.4 shows the evolution of the eigenvalues of H and of the maximal angle ψ between $\mathcal{H}$ and $\mathcal{X}$. From $t = 120$, $A = H + 0.4I$ stabilizes all intended binary patterns and no others.

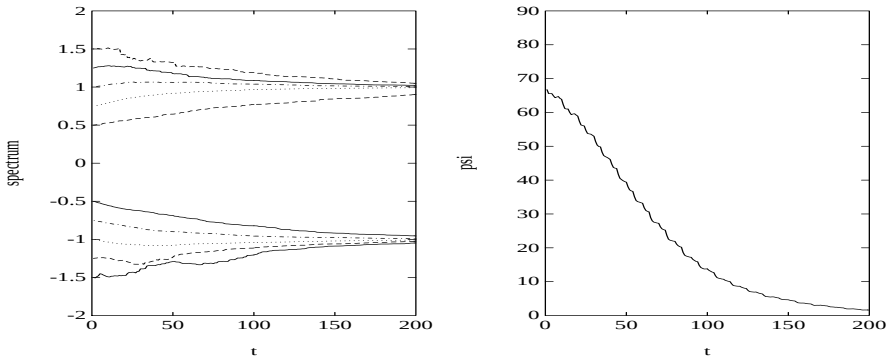


Figure 10.4: Simulation of a discretized version of adaptation algorithm 10.24 for 10 neurons and 5 patterns, with $\rho = 1$ and $\tau = 1$ and $\alpha = 0$

Simulation example 3
Fig. 10.5 shows a simulation of a discretized version of adaptation algorithm 10.26, for $n = 50$, and $m = 20$.

$$\begin{aligned} H(t+1) &= H(t) + 0.01\{(H(t) + H(t)^2)x(t)x(t)^T(I - H(t)) \\ &+ (I - H(t))x(t)x(t)^T(H(t) + H(t)^2)\}. \end{aligned}$$

The stabilizing terms correct for possible discretization errors. The patterns have been chosen randomly, and the initial state has exact eigenvalues, 1 and -1. The initial maximal angle ψ is close to 90 degrees as is normal for a random pattern set of the given size.

10.8 Conclusion

In this chapter we have studied a nonautonomous variant of the double bracket flow and different strategies for stabilizing given eigenvalues, which fit in the class of uniform parallel algorithms of definition 5.1.

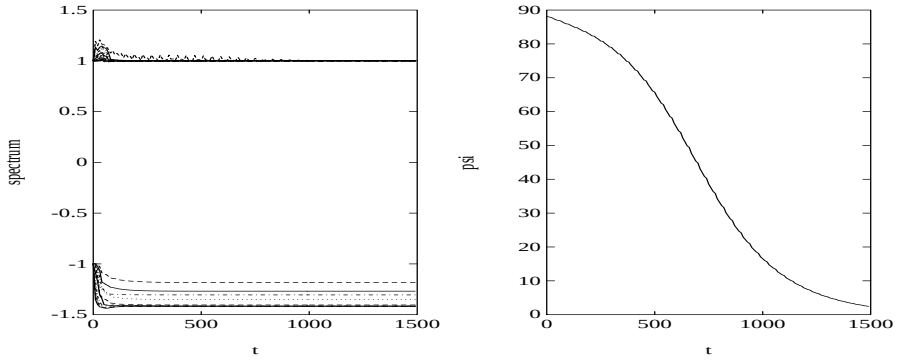


Figure 10.5: Simulation of a discretized version of adaptation algorithm 10.26 for 50 neurons and 20 patterns

Using these ideas we have derived adaptation laws for recurrent neural networks with memory applications. Exploiting the isospectrality, these laws can store patterns shown by a teaching signal, without tending to store many unwanted patterns, as is the case for more classical adaptation laws. If the operation and adaptation phase are implemented together, the adaptation can be thought of as confirming the desirability of the present state, and new things can be stored by forcing the state with an external input.

A side theme of this chapter was also the approximation of autonomous continuous-time algorithms with a complex signal flow graph (e.g. for which the number of connections between cells grows with the size of the problem) by nonautonomous lower complexity algorithms that may converge more slowly. Both the nonautonomous double bracket algorithm (algorithm 10.9) and the algorithms discussed in theorem 10.18 illustrate this idea.

Appendix

10.A Comparison of three related neural network models

In this appendix we compare the three models defined above (definitions 10.1, 10.2 and 10.3). We will indicate the states of the BSB, Hopfield and LSSM models as $z^{(B)}$, $z^{(H)}$ and $z^{(L)}$ and the output of the Hopfield model as $y^{(H)}$.

We first show the relation between the output $y^{(H)}$ of the Hopfield model and the state $z^{(L)}$ of the LSSM model.

If $|z^{(H)}{}_i| \neq 1$ for all $i = 1, \ldots, n$, f is differentiable in $z^{(H)}$ and the derivative of the output $y^{(H)}$ is described by

$$\dot{y}^{(H)} = J(z^{(H)})(-z^{(H)} + L(y^{(H)})), \tag{10.19}$$

where $J(z^{(H)})$ is the Jacobian of $f(z^{(H)})$ ($J_{i,j} = \frac{\partial f_i}{\partial z^{(H)}{}_j}$). This is a diagonal matrix with

$$\begin{aligned} J_{i,i}(z^{(H)}) &= 1 \text{ if } -1 < z^{(H)}{}_i < 1 \\ J_{i,i}(z^{(H)}) &= 0 \text{ if } z^{(H)}{}_i < -1 \text{ or } z^{(H)}{}_i > 1 \\ i &= 1, \ldots, n. \end{aligned}$$

If some $|z^{(H)}{}_i| = 1$, there is a difference between the left and right time derivative of $y^{(H)}$. We are interested in the right time derivative and will preserve the notation $\dot{y}^{(H)}$. Three cases are to be distinguished for each component of $y^{(H)}$.

If $z^{(H)}{}_i \dot{z}^{(H)}_i < 0$, $z^{(H)}{}_i$ evolves into $(-1, 1)$, and $y^{(H)}{}_i(t + \Delta t) = z^{(H)}{}_i(t + \Delta t)$ for $0 < \Delta t < \epsilon$ for some $\epsilon > 0$. Therefore

$$\dot{y}^{(H)}_i = \dot{z}^{(H)}_i. \tag{10.20}$$

If $z^{(H)}{}_i \dot{z}^{(H)}_i > 0$, $z^{(H)}{}_i$ evolves out of $[-1, 1]$, and $y^{(H)}{}_i(t + \Delta t) = y^{(H)}{}_i(t)$ for $0 < \Delta t < \epsilon$ for some $\epsilon > 0$. Therefore

$$\dot{y}^{(H)}_i = 0. \tag{10.21}$$

If $z^{(H)}{}_i \dot{z}^{(H)}_i = 0$, both (10.20) and (10.21) hold. With this information (10.19) can be generalized for all $z^{(H)}$ by defining $J(z^{(H)})$ for the case where some $|z^{(H)}{}_i| = 1$ as

$$\begin{aligned} J_{i,i}(z^{(H)}) &= 1 \text{ if } |z^{(H)}{}_i| = 1 \text{ and } z^{(H)}{}_i \dot{z}^{(H)}_i < 0, \\ J_{i,i}(z^{(H)}) &= 0 \text{ if } |z^{(H)}{}_i| = 1 \text{ and } z^{(H)}{}_i \dot{z}^{(H)}_i \geq 0, \end{aligned}$$

$$\begin{aligned} J_{i,i}(z^{(H)}) &= \frac{\partial f_i}{\partial {z^{(H)}}_i} \text{ if } |{z^{(H)}}_i| \neq 1, \\ J_{i,j}(z^{(H)}) &= 0 \text{ if } j \neq i, \\ i,j &= 1,\ldots,n. \end{aligned} \tag{10.22}$$

Fig. 10.6 shows the values J takes in different regions of the state space for $n = 2$. Equation (10.22) states that for a point $z^{(H)}$ on the boundary of those regions, J takes the value of the region $z^{(H)}$ is entering, according to the system equation. We say that neuron i is saturated iff $J_{i,i}(z) = 0$, where $z = z^{(H)}$, $z^{(B)}$, or $z^{(L)}$.

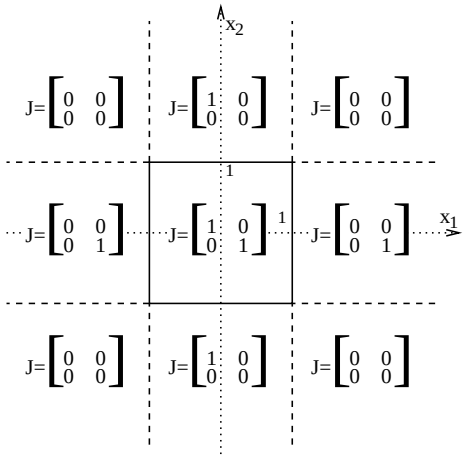


Figure 10.6: value of J in different regions

It is easy to check that $J(z^{(H)})z^{(H)} = J(z^{(H)})y^{(H)}$ (differing components of $z^{(H)}$ and $y^{(H)}$ are canceled by $J(z^{(H)})$). Therefore equation (10.19) can be recast as

$$\dot{y}^{(H)} = J(z^{(H)})(-y^{(H)} + L(y^{(H)})). \tag{10.23}$$

In the LSSM model, the restriction $-1 \leq {z^{(L)}}_i \leq 1$ implies that ${z^{(L)}}_i$ remains equal to 1 or -1 if the linear system prescribes an evolution out of the hypercube $C = [-1, 1]^n$. That is, $\dot{z}^{(L)}$ is the vector which results from $-z^{(L)} + L(z^{(L)})$ if all components that would bring $z^{(L)}$ out of C are zeroed. Therefore the LSSM model is equivalent to

$$\dot{z}^{(L)} = J(z^{(L)})(-z^{(L)} + L(z^{(L)})), \tag{10.24}$$

where the initial value of $z^{(L)}$ has to be chosen in C.

Comparing (10.23) and (10.24) one can see that the evolution of the output $y^{(H)}$ of the Hopfield model is related to the evolution of $z^{(L)}$ in the LSSM model.

The models differ where $J(z^{(H)}) \neq J(y^{(H)})$, that is when $y^{(H)} = z^{(L)}$ is on the boundary of the cube, $z^{(H)} \neq y^{(H)}$ and ${z^{(L)}}_i \dot{z}_i^{(L)} < 0$ for some $|{z^{(L)}}_i| = 1$. Fig 10.7 shows how this can also lead to a different evolution, starting at a point in $C = [-1, 1]^n$: The orbit starts at point 1, $z^{(H)} = y^{(H)} = z^{(L)}$, on the boundary of C. $\dot{z}^{(H)}$ is pointing upwards, $z^{(H)}$ evolves away from C. As long as $L(z^{(L)}) - z^{(L)}$ keeps pointing upwards, $J(z^{(H)}) = J(y^{(H)}) = J(z^{(L)})$ and $y^{(H)}$ and $z^{(L)}$ have the same evolution. However, when $\dot{z}^{(H)}$ points downwards (beyond point 2), $J(z^{(L)}) = 1$, whereas $J(z^{(H)})$ remains 0 until $z^{(H)}$ has reached the boundary of C. Therefore $z^{(L)}$ evolves inside C, while $y^{(H)}$ stays longer on the boundary. This can have an influence on the attraction basins as is shown by the following

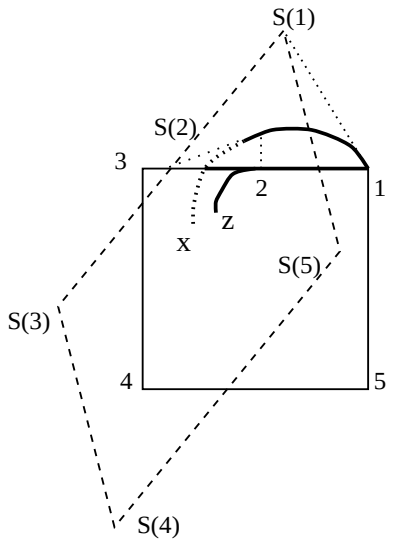


Figure 10.7: different trajectories for Hopfield and LSSM model

example.

Let

$$[A \mid b] = \begin{bmatrix} 5 & -4 & -0.1 \\ 2 & 7 & -7 \end{bmatrix},$$

then for $z^{(H)}(0) = [1\ 1]^T$, $z^{(H)}$ converges to $[-1\ 1]$, while for $z^{(L)}(0) = [1\ 1]^T$, $z^{(L)}$ converges to $[1\ -1]^T$. However, it took some effort to construct this two-dimensional example, and the case doesn't seem to occur frequently.

The BSB model is related to these models in another way. Consider the evolution of $w^{(B)} = Az^{(B)} + b$. One finds

$$\dot{w}^{(B)} = A\dot{z}^{(B)} = -Az^{(B)} + Af(w^{(B)}) = -w^{(B)} + Af(w^{(B)}) + b = -w^{(B)} + L(f(w^{(B)})).$$

That is, $w^{(B)}$ evolves like $z^{(H)}$ in the Hopfield model. However, while the output behavior of the BSB model is described in terms of $z^{(B)}$, the output behavior of

the Hopfield model is described in terms of $y^{(H)}$, the projection of $z^{(H)}$ (evolving like $w^{(B)}$) on C. This clearly leads to different trajectories and attraction basins. Nevertheless the output behavior of both models turns out to have basic similarities, as will be explained at the end of this appendix, using the following geometric interpretation.

Any system

$$\dot{z} = -z + S(z),$$

can be interpreted geometrically in state space as "z follows $S(z)$": The velocity $\dot{z}$ of z is the vector which points to $S(z)$ when based at z. Therefore z evolves towards $S(z)$ with a velocity proportional to the distance. Of course $S(z)$ changes accordingly, such that z evolves like someone following his *shadow* $S(z)$, for ever or until z and $S(z)$ coincide.

This point of view is especially fruitful for the Hopfield model, where the 'shadow' $S(z^{(H)}) = L(y^{(H)}) = L(f(z^{(H)}))$ has a simple geometrical interpretation, leading to intuitive insight in the dynamical behavior. $y^{(H)}$ is the projection of $z^{(H)}$ onto C and $S(z^{(H)})$ is the corresponding point (under the affine transformation L) of the parallellepiped $L(C)$. We have used this point of view in [28, 30, 29, 31] to derive conditions for attraction basins and for the analysis of cellular neural networks.

The shadow interpretation also allows us to see the three considered models as three ways to "confine a linear system to the hypercube $C = [-1, 1]^n$". That is, to make sure that the output of the (modified) system is in C (or approaches C asymptotically).

For a linear system

$$\dot{z} = -z + L(z),$$

$S(z) = L(z)$ is an affine function.

In the BSB model $z = z^{(B)}$ is confined to C by tackling the shadow. $L(z)$ is replaced by its projection on C, $f(L(z))$. Therefore a point z always evolves towards a shadow in C. If only initial states in C are allowed, z will never leave C.

In the LSSM model z itself is tackled. When z attempts to leave C, it is kept on the border, by canceling the responsible components of $\dot{z}$. In addition, the initial value of z needs to be in C.

The output $y^{(H)}$ of the Hopfield model is confined to C by simply projecting the state onto C.

Finally, the geometric interpretation leads to an explanation of the similarity between the output behavior of the Hopfield model and the BSB model. It follows from the argument above that $f(w^{(B)})$ evolves like the output of the Hopfield

model (with state $w^{(B)}$). Since $\dot{z}^{(B)} = -z^{(B)} + f(w^{(B)})$, *the state of the BSB model is following a shadow which evolves like the output of the Hopfield model.* This also illustrates a drawback of the BSB model. The output of the Hopfield can stabilize, for instance at a corner of the cube, after finite time. In that case the state of the BSB model will approach the corner only asymptotically.

Chapter 11

Optimization algorithms for computing a lower-bound for the structured singular value

In this chapter we consider gradient based optimization algorithms on a Lie group of structured unitary matrices, calculating a lower-bound for the structured singular value, indicated below by μ.

The existing power iteration algorithm (PIA) for this problem [87] can run into limit cycles in which case only a trivial lower-bound for μ is given, our algorithms are ascent methods, which always provide nontrivial lower-bounds for μ. However, as they are less efficient than the PIA, we suggest that both approaches are combined, for example first try the power iteration algorithm, and if it does not converge within a preset time limit, use one of our algorithms.

The link with the remainder of this work is as follows.

First, the derivation of the algorithm is an application of the ideas of section 3.3 about calculating gradients of functions over a manifold.

Secondly, as only a limited set of continuous-time algorithms are suited for analog realization, the relevance of the continuous-time approach (or in this context 'differential geometric approach' is more appropriate) also depends on the possibility of discretization. In this chapter we derive discrete-time gradient algorithms that 'live' in regular manifolds, like the Lie groups and homogeneous spaces discussed

in section 2.3. The algorithms perform line searches along geodesics on a Lie group, following the ideas of [94]. We will not go into details about differential geometric concepts as geodesics and parallel translation, but only use them.

Thirdly, we also consider continuous-time variants of the algorithm. These algorithms have played a role for early experiments. Also, as in previous chapters, simple formulas are obtained with simple parallel signal flow graphs. However, in this case, they do not lead to parallel discrete-time algorithms.

This chapter is organized as follows. In section 11.1 we describe the structured singular value problem as an optimization problem, and situate the optimization strategy. Section 11.2 discusses the Lie group of structured unitary matrices. In sections 11.3 and 11.4 the steepest ascent algorithm and the conjugate gradient algorithm for the structured singular value problem are derived. In section 11.5 we consider continuous-time algorithms and section 11.6 gives numerical experiments.

The main results of this chapter were published in [111, 37].

11.1 Optimization strategies for the structured singular value problem

11.1.1 Problem definition

The structured singular value, generally referred to as μ, is an important tool for the robustness analysis of systems with a structured uncertainty [42]. In this section we give a definition, another formulation as an equivalent optimization problem, and situate our optimization strategy and the existing power iteration algorithm [87], in a more general scheme.

Definition 11.1
The structured singular value is defined by

$$\mu_{\mathbf{\Delta}}(M) = \frac{1}{\min_{\Delta \in \mathbf{\Delta}} \{\bar{\sigma}(\Delta) | \det(I + M\Delta) = 0\}}, \tag{11.1}$$

where $\bar{\sigma}$ denotes the maximal singular value,
and $\mathbf{\Delta}$ is a set of block diagonal matrices

$$\mathbf{\Delta} = \{ \mathrm{diag}[\delta_1 I_{r_1}, \ldots, \delta_s I_{r_s}, \Delta_1, \ldots, \Delta_f] \mid \delta_k \in \mathbb{C}, \Delta_l \in \mathbb{C}^{m_l \times m_l} \},$$

defining the *structure*.
The blocks $\delta_k I_{r_k} \in \mathbb{C}^{r_k \times r_k}$ are *repeated scalar blocks*.
Blocks of the form $\Delta_l \in \mathbb{C}^{m_l \times m_l}$ are *full blocks*.

We will refer to the submatrices of a matrix A with the same size and position as $\delta_k I_{r_k}$ and Δ_l, as $A_{r_k,r_k}, k = 1, \ldots, s$ and $A_{f_l,f_l}, l = 1, \ldots, f$, respectively.

An equivalent definition is [87]

Definition 11.2

$$\mu_{\mathbf{\Delta}}(M) = \max_{Q \in \mathcal{Q}} \rho(QM),$$

where $\rho(QM)$ denotes the spectral radius, that is, the maximal absolute value of an eigenvalue of QM
and $\mathcal{Q} = \{Q \in \mathbf{\Delta} | Q^H Q = I_n\}$ is the set of *structured unitary matrices*.

Our algorithms are based on this last definition.

Remark 11.3
All presented results also hold for nonsquare matrices M, by extending them with zero rows or columns to make all blocks square.

11.1.2 Optimization strategy

No algorithm is known that finds the global optimum of the optimization problem, instead algorithms used today find lower-bounds for μ. Other algorithms exist that give upper-bounds, such that both together provide the user with an interval containing μ.

The power iteration algorithm (PIA) of Packard, Fan and Doyle [87] computes a lower-bound for μ and is computationally efficient if it converges. However, up to this date, there is no proved convergence result. As a matter of fact, current implementations do not necessarily converge as was already mentioned in [87] (see [110, 109] for a counter example). If the algorithm does not converge, no lower-bounds result, except trivial ones. The continuous-time counterpart of the PIA by Yi and De Moor [110, 109], seems to have better convergence properties but also lacks a convergence proof.

The approach developed in this work is less efficient, but has a less heuristic theoretical foundation. Both approaches fit into the following general discussion.

To solve an optimization problem (in the sense of finding a local maximum), one can either apply *general optimization algorithms* using function evaluations and possibly gradient information, to find a locally optimal solution, or one can concentrate directly on the *optimality conditions* for the given problem (taking different forms depending on the type of constraints and the smoothness of the problem).

If the *optimality conditions* cannot be solved analytically, an iterative scheme is needed to solve them. Often, it is not difficult to produce (many) iterative schemes which converge to solutions that satisfy the optimality conditions, if they converge at all. If computer simulations show that the scheme converges in a vast number of cases, such algorithms may be good engineering tools. This is where the PIA fits into the scheme. It converges in a lot of cases but not always, as mentioned above.

In this work we choose the approach of *general optimization algorithms*, more specifically the steepest ascent algorithm and the conjugate gradient algorithm. Actually, such an approach may be regarded as just another iterative algorithm to satisfy the optimality conditions, but now one with interpretable intermediate results and possibly provable convergence results (also see below). The price for these results may be the loss of computational efficiency, but for the algorithms presented below the computational efficiency is still acceptable if one takes into account that the structured singular value problem is mainly a problem of system analysis for which efficiency is not the main issue. We suggest that both approaches are combined.

We will not rigorously prove convergence of the discrete-time algorithms. This would imply a detailed consideration (and possibly adaptation) of the line search algorithms to avoid jumping between different points Q with the same (or almost the same) value for the objective function ρ. However, since only the obtained value of ρ, and not the point Q where it is reached, is important it is not worth putting too much energy in such considerations. We come back to the convergence issue at the end of section 11.3.

In the above discussion, we did not yet mention the constraint. To take into account the highly regular constraint $Q \in \mathcal{Q}$ of the optimization problem (definition 11.2), we directly define an algorithm that "lives" in $\mathcal{Q}$ instead of in a Euclidean space. The steepest ascent and conjugate gradient algorithms on this manifold will be derived in sections 11.3 and 11.4. First, the next section introduces some facts about the manifold $\mathcal{Q}$.

11.2 The manifold $\mathcal{Q}$ of structured unitary matrices

The optimization methods discussed in [94] generalize optimization methods over Euclidean spaces to optimization methods over Riemannian manifolds by making use of intrinsic properties of the manifold. To this end, *straight lines are replaced by geodesics*. Tangent vectors at one point are adapted to other points by parallel translations along geodesics. In this section we establish some properties of the manifold $\mathcal{Q}$ of structured unitary matrices, which will be needed

further on.

Proposition 11.4
(i) The manifold $\mathcal{Q} = \{Q \in \mathbf{\Delta} | Q^H Q = I_n\}$ is a Lie group.
(ii) The corresponding Lie algebra $\mathfrak{q}$ is given by

$$\mathfrak{q} = \{S \in \mathbf{\Delta} | S^H + S = 0\}. \tag{11.2}$$

The tangent space at $Q \in \mathcal{Q}$ equals

$$T_Q \mathcal{Q} = \{T \in \mathbb{C}^{n \times n} | T = SQ, S \in \mathbf{\Delta}, S^H + S = 0_n\}. \tag{11.3}$$

(iii) The inner product of tangent vectors $RQ \in T_Q \mathcal{Q}$ and $SQ \in T_Q \mathcal{Q}$ with $R, S \in \mathfrak{q}$, induced by the standard inner product in $\mathbb{C}^{n \times n}$ (definition 3.27) can be written as

$$\langle RQ, SQ \rangle = \langle R, S \rangle = \mathrm{Tr}(R^H S) \tag{11.4}$$

and defines a bi-invariant Riemannian metric on $\mathcal{Q}$.

(iv) Given the inner product (11.4) and the corresponding Riemannian connection, the geodesic emanating from $Q = Q(0)$ in a direction SQ, parameterized by the real parameter t, is given by

$$Q(t) = \exp(St)Q. \tag{11.5}$$

(v) The parallel translation of a tangent vector $RQ = R(0)Q(0) \in T_Q \mathcal{Q}$ along the geodesic (11.5) is given by

$$R(t)Q(t) = \exp(\frac{1}{2}St)R(0)\exp(-\frac{1}{2}St)Q(t). \tag{11.6}$$

Proof:
(i) One can easily prove that $Q \in \mathcal{Q}$ iff $Q_{f_l, f_l} \in \mathrm{U}(m_l)$, $l = 1, \ldots, f$ (that is full blocks are unitary matrices) and $Q_{r_k, r_k} = q_k I_{r_k}$ for some $q_k \in S_1 = \{c \in \mathbb{C} | c^* c = 1\}$. Therefore $\mathcal{Q}$ can be identified with the product of Lie groups $U(m_1) \times \ldots \times U(m_f) \times S_1^s$, which is itself a Lie group.
(ii) Consider a path Q(t) on $\mathcal{Q}$ through $Q = Q(0)$. Q preserves its structure iff $\dot{Q} \in \mathbf{\Delta}$ and Q stays unitary iff $\frac{d}{dt}(QQ^H) = 0$, which can be worked out as $\dot{Q}Q^H + Q\dot{Q}^H = \dot{Q}Q^H + (\dot{Q}Q^H)^H = 0$. This proves (11.3) and (11.2), where $S = \dot{Q}Q^H$.
(iii) Clearly, $\langle RQ, SQ \rangle = \mathrm{Re\ Tr}(RQQ^H S^H) = \mathrm{Re\ Tr}(RS^H) = \langle R, S \rangle$. And $\mathrm{Tr}(R^H S)$ is real because R and S are skew Hermitian: $\mathrm{Tr}(R^H S) = \mathrm{Tr}(RS^H) = \mathrm{Tr}(S^H R) = (\mathrm{Tr}(R^H S))^*$.
To prove the bi-invariance one can verify (by similar manipulations) that $\langle UA, UB \rangle = \langle A, B \rangle = \langle AU, BU \rangle$ for any $U \in \mathcal{Q}$.
(iv,v) These formulas follow from the application of the general formula given by Smith [94, p.19], and are almost identical to his formulas for the example of the orthogonal group [94, p.22].
□

11.3 The Steepest Ascent Algorithm

In this section we derive the steepest ascent algorithm (SAA) on the manifold $\mathcal{Q}$, for the optimization problem of definition 11.2. First we introduce the algorithm, then we consider the computational cost. Finally we discuss the convergence issue.

11.3.1 The algorithm

The SAA is an iterative algorithm, calculating in each step $k = 1, 2, \ldots$ a new point $Q^{[k]}$ on the manifold $\mathcal{Q}$, given a point $Q^{[k-1]}$. To this end, a (one dimensional) optimization problem, often referred to as *line search*, is solved, maximizing $\rho(QM)$ along a geodesic on $\mathcal{Q}$, emanating from Q_{k-1} in the direction of the gradient. To simplify some of the derivations, $\Phi(Q) = (\rho(QM))^2$ will be used as an objective function, instead of $\rho(QM)$.

The objective function $\Phi(Q)$ is not differentiable at all points $Q \in \mathcal{Q}$. Theoretically, three cases are to be distinguished.

case a: the spectral radius $\rho(QM)$ corresponds to one eigenvalue of QM with algebraic multiplicity 1.
case b: the spectral radius $\rho(QM)$ corresponds to $k > 1$ eigenvalues of QM with algebraic multiplicity 1.
case c: the spectral radius $\rho(QM)$ corresponds to an eigenvalue with algebraic multiplicity greater than 1.

Case a is generic. Case b is likely to occur at isolated points of some geodesics (when the absolute value of one eigenvalue crosses the absolute value of another eigenvalue). However, since $\rho(QM)$ is defined as $\max_i |\lambda_i|$ where λ_i are the eigenvalues of QM, and since these absolute values are smooth functions of Q in a neighborhood of points of case b, the maximum of $\rho(QM)$ cannot occur at a point of case b unless $\rho(QM)$ and $\Phi(QM)$ are differentiable at that point. Case c does in general not occur, and we will take a heuristic approach (see remark 11.8). The following theorem gives a formula for the gradient in differentiable points, and shows how the same formula gives an ascent direction (or 0 in a local maximum) for points of case b.

Theorem 11.5
(i) Let the spectral radius of $X = QM$ correspond to one eigenvalue λ of X with algebraic multiplicity 1 (case a above), and let p be a corresponding right eigenvector and v a corresponding left eigenvector, satisfying $v^H p = 1$. The gradient of $\Phi(Q) = \rho^2(QM)$, at the point $Q \in \mathcal{Q}$, with respect to the standard induced inner product (11.4) is given by

$$\text{grad}_Q \Phi = \Phi(Q) GQ \text{ where } G = \pi(vp^H - pv^H),$$

and π denotes the orthogonal projection on $\mathbf{\Delta}$ and is given by

$$\pi(A) = \mathsf{diag}[\alpha_1 I_{r_1}, \ldots, \alpha_s I_{r_s}, A_{f_1,f_1}, \ldots, A_{f_f,f_f}] \text{ for any } A \in \mathbb{C}^{n\times n},$$

where $\alpha_k = \frac{1}{r_k}\mathsf{Tr}(A_{r_k,r_k}), k = 1, \ldots, s$. That is, π zeros elements that have to be zero in $\mathbf{\Delta}$, and averages elements that have to be equal in $\mathbf{\Delta}$.

(ii) If the spectral radius $\rho(QM)$ corresponds to $m > 1$ eigenvalues of QM with algebraic multiplicity 1, then $\rho(QM)$ is the maximum of m differentiable functions, corresponding to the different eigenvalues with absolute value $\rho(QM)$. For any choice λ among these eigenvalues, and the corresponding right and left eigenvectors p and v, satisfying $v^H p = 1$, $\Phi(Q)GQ$ is the gradient of one of these m functions, and gives an ascent direction of $\rho(QM)$ or gives 0 in a critical point.

Proof:
(i) Consider a path $Q(t) \in \mathcal{Q}$ through $Q = Q(0)$, with velocity $\dot{Q}(0) = SQ \in T_Q\mathcal{Q}$. From definition 3.21 of the gradient, we have to prove that $\Phi(Q)GQ \in T_Q\mathcal{Q}$ (which is clear from proposition 11.4(ii)), and that

$$\dot{\Phi}(0) = \langle \Phi(Q)GQ, SQ\rangle, \tag{11.7}$$

where $\Phi(t) = \Phi(Q(t))$.

From $X = QM$ one finds $\dot{X} = \dot{Q}M = SQM = SX$. And since, by assumption the spectral radius of X corresponds to only one eigenvalue λ, $\lambda(t)$ is differentiable at $t = 0$ and from lemma 3.32 (also see [60]) $\dot{\lambda}(0) = v^H\dot{X}p = v^HSXp = \lambda v^HSp$. Finally,

$$\begin{aligned}
\dot{\Phi}(0) &= \tfrac{d}{dt}(\lambda^*\lambda) \\
&= \dot{\lambda}^*\lambda + \lambda^*\dot{\lambda} \\
&= \lambda^*\lambda v^HSp + \lambda^*\lambda p^HS^Hv \\
&= \Phi(Q)(v^HSp - p^HSv) \\
&= \langle S, \Phi(Q)(vp^H - pv^H)\rangle \\
&= \langle \pi(S), \Phi(Q)(vp^H - pv^H)\rangle \\
&= \langle S, \Phi(Q)\pi(vp^H - pv^H)\rangle.
\end{aligned}$$

With proposition 11.4(iii) this proves (11.7).

(ii) In a neighborhood of Q, $\Phi(Q)$ can now be regarded as the maximum of m functions, that are equal at Q. The gradient of each of these functions separately is given by $\Phi(Q)GQ$, with one possible choice for λ, and the corresponding p and v. These gradients give steepest ascent directions (or 0 in critical points) for the individual functions, and therefore certainly give ascent directions for the maximum $\Phi(Q)$.
□

Application of the steepest ascent method to the present problem now yields the following algorithm.

Algorithm 11.6 (Steepest Ascent Algorithm)

1. initialize Q_0
2. $k = 0$
3. **repeat**
4. compute $\rho = \rho(Q_k M)$
   and corresponding right and left eigenvectors p and v,
   such that $v^H p = 1$
5. $G = \pi(vp^H - pv^H)$
6. $t = \arg\max_{t>0} \rho(\exp(Gt)Q_k M)$
7. $Q_{k+1} = \exp(Gt)Q_k$
8. $k = k + 1$
9. **until** convergence

Remark 11.7
The exponentiation of $\exp(Gt) = \exp(\pi(vp^H - pv^H)t)$ can be calculated from the block components separately,

$$\exp(Gt) = \text{diag}[\exp(G_{r_1,r_1}t), \ldots, \exp(G_{r_s,r_s}t), \exp(G_{f_1,f_1}t), \ldots, \exp(G_{f_f,f_f}t)]. \tag{11.8}$$

For the calculation of $\exp(G_\square t)$ for different t, where the index $\square$ denotes one of the block components in (11.8), the eigenvalue decomposition of $G_\square$ needs to be calculated only once. If the eigenvalue decomposition of $G_\square$ is given by $G_\square = EiDE^H$, where $E^H E = I$ and D is real and diagonal, $\exp(G_\square t)$ is found as:

$$\exp(G_\square t) = E \exp(iDt) E^H.$$

Note that for repeated scalar blocks one finds $E = I$. For full blocks, more efficient formulas can be obtained also, by exploiting the fact that the block components of G have rank 2 (if not 0), but the computational benefit is rather small compared to the cost of the repeated computation of ρ, p and v.

Remark 11.8
For points of case c, calculation of an ascent direction is much more difficult than for other points. In addition the effort is not worthwhile, as this case does not normally occur and if it occurs, it can generally be avoided. In most cases, when a point of case c is met during the line search, one can simply take another point. In many cases points of case c can be avoided by multiplying Q with $U \in \mathcal{Q}$, for which $Up = p$ and $U^H v = v$. In that case $\Phi(UQ) = \Phi(Q)$ but in general UQ will no longer be of case c. If none of this works the algorithm is stopped (and possibly restarted from another initial value). After all, the point of case c, can be a local optimum in contrived examples. Also see the discussion of the convergence issue at the end of this section.

11.3.2 Computational cost

The computationally most expensive steps of the algorithm are the evaluations of the spectral radius of X and the corresponding right and left eigenvector. These do not only occur in line 4 of the algorithm but mainly in line 6, during the line search. (Actually, except for the first step in the iteration, nothing needs to be calculated in line 4, since ρ, p and v were calculated already during the last execution of line 6). In order to minimize this cost, the following things can be done.

1. The number of eigenvalue/eigenvector calculations is kept low, using an efficient line search algorithm by Fletcher [44, pp.33]. This algorithm finds a point, satisfying the (modified) Wolfe-Powell conditions (normally given for minima instead of maxima)

$$\begin{array}{rcl} \Phi(t) & \geq & \Phi(0) + t\,\rho_F\,\frac{d\Phi}{dt}(0), \\ |\frac{d\Phi}{dt}(t)| & \leq & \sigma_F\,\frac{d\Phi}{dt}(0), \end{array} \tag{11.9}$$

where ρ_F and $\sigma_F \geq \rho_F$ are fixed parameters. Typically $\sigma_F = 0.5$ and $\rho_F = 0.01$, for a moderately accurate line search [44, p.30]. Note that Fletcher's algorithms are designed for smooth optimization problems. However because the local maximum does not occur at a nondifferentiable point, normally no problems arise. (However, as we have not proved that no nondifferentiable maxima of case c can exist, to be really sure one should use a line search algorithm that can also take into account this case).

2. For the calculation of ρ, p and v, some profit can be gained from the fact that only the principal eigenvalue and the corresponding eigenvectors are needed, and the fact that previous calculations for nearby X provide good initial estimations. Both the well known power algorithm (not to be confused with the PIA for the structured singular value, called power algorithm for its resemblance with the classical power algorithm) and the inverse iteration algorithm can save a lot of computations. However, the power algorithm is not efficient if the maximal eigenvalue of X is not well separated (in absolute value) from the other eigenvalues, and the inverse iteration algorithm can converge to the wrong eigenvector with a bad initial guess for the eigenvalue. Therefore, the price of the computational improvement, is a number of heuristic rules to handle the distinction between different cases, or the loss of the ascent property. However, our experience shows that problems are rare and efficiency can be increased, when the line search is made more robust against inaccurate gradient information. In most numerical experiments however we have used the MATLAB function "eig" [97], to calculate a full eigenvalue decomposition.

11.3.3 Remarks on the convergence issue

Finally, we come back to the convergence issue, already mentioned in section 11.1. For discrete-time ascent algorithms, convergence is a subtle issue. The ascent property, together with the boundedness of Q, only proves convergence of ρ (or Φ) not of Q. Whether Q converges can depend on properties of the line search (although the fact that $\mathcal{Q}$ is compact already avoids a number of problems). However, since we are only interested in ρ (to estimate μ), this is no problem. For this reason we do not put more energy in proving convergence results for Q. Nevertheless, it is important that not too many points are attractive, especially if they have low ρ. Saddle points can be attractive in theory, but only from special initial conditions. Moreover random perturbations are often sufficient to escape. Because the function is nondifferentiable, in theory also nondifferentiable points could be attractive, without being locally maximal. However, in trying to construct such examples, we always found the following facts on our way. *First*, around points of case b, the objective function is the maximum of differentiable functions, making it hard to imagine attraction to a point where they are equal. *Secondly*, the line search always evaluates points at two sides of the line maximum, making it hard to avoid evaluating points with a value higher than the candidate attractor. *Thirdly*, for points of case c with geometric multiplicity 1, non-differentiability seems to correspond with an eigenvalue splitting into two eigenvalues with derivative $+\infty$ for one eigenvalue and $-\infty$ for the other one (also see remark 3.33), making the point highly repulsive (again because we are maximizing the maximal one). *Finally*, this leaves points of case c with geometric multiplicity greater than 1. These are more difficult to study, but also much more unlikely to occur. We have not been able to construct attractive nondifferentiable examples.

In summary, *the algorithm keeps increasing ρ, and not in a trivial way, like taking directions that converge to a direction of no change, or taking step sizes converging to 0, where it could easily do better.*

11.4 The Conjugate Gradient Algorithm

In this section we derive the conjugate gradient algorithm (CGA) on the manifold $\mathcal{Q}$, solving the optimization problem as given by definition 11.2. For the CGA the search direction at a point Q_k is determined in most steps by the gradient at Q_k and the previous search direction at Q_{k-1}. In most versions, the gradient direction itself is taken every n-th step (where n is the problem dimension). Smith proposes the following formula for the search direction H_kQ, at a point Q_k using the gradients G_kQ and $G_{k-1}Q$ and the previous search direction $H_{k-1}Q$. It is very similar to the Polak-Ribière formula for the CGA in Euclidean spaces, but tangent vectors at the point Q_{k-1} from the previous iteration step, are

adapted to Q_k by parallel translation along the geodesic through Q_{k-1} and Q_k.

$$H_k = G_k + \gamma_k \tau H_{k-1},$$

where

$$\gamma_k = \frac{\langle G_k - \tau G_{k-1}, G_k \rangle}{\langle G_{k-1}, H_{k-1} \rangle},$$

where τ denotes parallel translation from Q_{k-1} to Q_k, applied to matrices $S \in \mathfrak{q}$ representing a tangent vector SQ at Q. It follows from theorem 1 that $\tau H_{k-1} = H_{k-1}$. To calculate τG_{k-1}, formula (11.6) is needed. This leads to the following algorithm.

Algorithm 11.9 (Conjugate Gradient Algorithm)

1. initialize Q_0
2. $k = 0$
3. **repeat**
4. compute $\rho = \rho(Q_k M)$
   and corresponding right and left eigenvectors p and v,
   such that $v^H p = 1$
5. $G_k = \Phi(Q)\pi(vp^H - pv^H)$
6. if $(k \bmod n) = 0$ then $H_k = G_k$
7. else $\tau G_{k-1} = \exp(\frac{1}{2}H_{k-1}t)G_{k-1}\exp(-\frac{1}{2}H_{k-1}t)$
8. $\gamma = \frac{\langle G_k - \tau G_{k-1}, G_k \rangle}{\langle G_{k-1}, H_{k-1} \rangle}$
9. $H_k = G_k + \gamma H_{k-1}$
10. $t = \arg\max_{t>0} \rho(\exp(H_k t)Q_k M)$
11. $Q_{k+1} = \exp(H_k t)Q_k$
12. $k = k + 1$
13. **until** convergence

Remark 11.10
For the calculation of $\exp(Ht)$ for different values of t, the same remark holds as for the SAA (remark 11.7).

Remark 11.11
For the line search of line 10, the same strategy as for the SAA is used. However, since the performance of the CGA is more sensitive to the quality of this line search, the conditions (11.9) are made stronger by taking $\sigma_F = 0.1$ ($\rho_F = 0.01$) [44, p.85].

The CGA is known to have better asymptotic convergence properties than the SAA (assuming convergence to a differentiable point). The additional cost of computing the conjugate gradient search direction is small compared to the cost of the function evaluations and gradient calculations, especially when the structured matrices consist of small blocks, since these computations can be performed on the block components separately. However, due to the stronger

requirements for the line search (remark 11.11), individual iterations may require more computations. In section 11.6 we show that for the present problem, both algorithms are more or less equivalent.

11.5 Continuous-time algorithms

In this section, we consider again continuous-time algorithms. *First*, we briefly consider the continuous-time gradient ascent. The algorithms, presented in this section played some role in the development of the algorithms presented above, mainly because of the simplicity of formulation, and the ease of implementation using standard integration routines. *Secondly*, we derive an algorithm, also calculating (or approximating) the required eigenvectors with the continuous-time power method.

11.5.1 A continuous-time gradient ascent algorithm

The gradient ascent algorithm

$$\dot{Q} = \text{grad}_Q \Phi \tag{11.10}$$

is only well defined for trajectories avoiding the points of case c, defined above. In points of case b, $\dot{Q}$ should be interpreted as a left or right derivative, depending on the choice of p and v. We have been able to use it for early tests, because points of case c are hardly ever encountered, and points of case b give no problems for most numerical integrators. If no points of case c are met and the algorithm is not attracted to a saddle point or a nondifferentiable point, (three unlikely cases), the algorithm converges to a local maximum of $\Phi(Q)$.

11.5.2 Combination with the power method

It is also possible to formulate a continuous-time algorithm, using the continuous-time power algorithm to approximate the eigenvalue/eigenvector calculations.

Since the continuous-time power algorithm, finds the eigenvalue with maximal real part and corresponding eigenvectors, it is necessary to recast the optimization problem as

$$\mu_{\mathbf{\Delta}}(M) = \max_{Q \in \mathcal{Q}} \text{Re}\, \bar{\lambda}(QM), \tag{11.11}$$

where $\bar{\lambda}$ denotes the eigenvalue with maximum real part. This is equivalent to the problem of definition 11.2, because an appropriate multiplication of Q with a complex scalar q, with $|q| = 1$, leaves $\rho(QM)$ invariant, while making

the corresponding eigenvalue real. A derivation, similar to the derivation of theorems 11.5 shows that

$$\nabla_Q \mathrm{Re}\,\bar{\lambda} = \pi(\bar{\lambda}^* \bar{v}\bar{p}^H - \bar{\lambda}\bar{p}\bar{v}^H)Q,$$

where $\bar{p}$ and $\bar{v}$ denote the right and left eigenvectors, corresponding to $\bar{\lambda}$.

The eigenvectors and eigenvalue can be computed by the following power algorithm, slightly adapted (compare with [16]), to obtain both the left and right eigenvectors of QM while preserving $\tilde{p}^H\tilde{p} = 1$ and $\tilde{v}^H\tilde{p} = 1$.

Algorithm 11.12

$$\begin{array}{rcl} \dot{\tilde{p}} & = & QM\tilde{p} - \tilde{\lambda}\tilde{p} \\ \dot{\tilde{v}}^H & = & \tilde{v}^H QM - \hat{\lambda}\tilde{v}^H \end{array} \quad \text{where } \tilde{\lambda} = \tilde{p}^H QM\tilde{p} \text{ and } \hat{\lambda} = 2\tilde{v}^H QM\tilde{p} - \tilde{\lambda}.$$

This is a generalization of the continuous-time power algorithm 4.3 for real symmetric matrices. It only differs from the algorithm given by Chu [16] (for the right eigenvector only), in the choice of $\tilde{\lambda}$ and $\hat{\lambda}$. The choice of these parameters does not influence the direction of $\tilde{p}$ and $\tilde{v}$ and were chosen here to preserve $\tilde{p}^H\tilde{p} = 1$ and $\tilde{v}^H\tilde{p} = 1$.

The following algorithm combines the gradient approach with the power method for the calculation of the eigenvectors.

Algorithm 11.13

$$\begin{array}{rcl} \dot{\tilde{p}} & = & QM\tilde{p} - \tilde{\lambda}\tilde{p}, \\ \dot{\tilde{v}}^H & = & \tilde{v}^H QM - \hat{\lambda}\tilde{v}^H, \\ \dot{Q} & = & \tau\pi(\tilde{\lambda}^*\tilde{v}\tilde{p}^H - \tilde{\lambda}\tilde{p}\tilde{v}^H)Q, \end{array}$$

where $\tilde{\lambda} = \tilde{p}^H QM\tilde{p}$ and $\hat{\lambda} = 2\tilde{v}^H QM\tilde{p} - \tilde{\lambda}$ and $\tau > 0$ is a scalar.

The power equations (for $\tilde{p}$ and $\tilde{v}$) assume of course that Q is constant. In practice this can be approximated by letting τ go to zero. As in the discrete-time algorithm, the price for better approximation of the eigenvectors, is the slower (relative) rate of the ascent. Some heuristics can be devised to make τ depend on an estimate of the convergence error of the power algorithm. In simulations we have set

$$\tau = \frac{1}{0.1 + \|\dot{\tilde{p}}\|\,\|\dot{\tilde{v}}\|/\|\tilde{v}\|}. \tag{11.12}$$

Recall that $\|\tilde{p}\| = 1$. If $\tilde{p}$ and $\tilde{v}$ are close to convergence, τ is close to 10. If they are far from convergence τ drops to 0. By combining the power algorithm

and the ascent algorithm in this way, the ascent of the objective function is no longer guaranteed. In practice, in all our experiments convergence could be forced, by the given formula for τ. However, finding a formula for τ that guarantees convergence is still an open problem.

Remark 11.14
Note that the right hand side of algorithm 11.13 consists of matrix vector products and vector outer products. Although this is not strictly a system of type I (definition 5.1), it is closely related and parallel signal flow graphs are easily obtained. For a neural-network-like interpretation, one could consider the power equations as operation level and the equation for Q as adaptation level. We make this remark to stress that the existence of simple parallel signal flow graphs is not an isolated characteristic of only a few algorithms, discussed in previous chapters. However, coming up with discrete-time, *pipelined* versions (showing the same amount of parallelism) as in chapter 8 is much more difficult, if possible at all.

11.6 Numerical experiments

In this section we support our results with numerical experiments. We give an idea of the efficiency of our algorithms and compare them with the PIA as implemented in the MATLAB μ toolbox [2]. The MATLAB code for our algorithms is available via anonymous ftp from ftp@esat.kuleuven.ac.be in the directory /pub/SISTA/dehaene/prog/ccmu.

All experiments were performed with the same convergence test. The algorithm stops if the relative increase of Φ drops below $1E-4$ for 2 consecutive steps. (A maximal number of 1000 iterations was set, but did not occur during the experiments.)

Fig. 11.1 compares the computational cost of SAA, CGA and PIA, for complex matrices with different sizes and a similar block structure, consisting of two repeated scalar blocks of size z and two full blocks of size z, where $z = 1, \ldots, 10$. The matrices were generated at random and for each size the computational cost was averaged over 100 experiments with different matrices. Experiments with different block structures gave no significant difference.

The SAA and CGA require aproximately the same amount of computations. (With weaker convergence criteria the SAA tends to take less time than the CGA). As mentioned above, the PIA takes considerably less time. However, it should be noted that we are comparing an experimental version of the SAA and CGA with a fine tuned version of PIA. Similar results were obtained with different convergence criteria. The experiments of figure 11.1 used random initial values. As for the amount of computations, the effect of different initial values is not big.

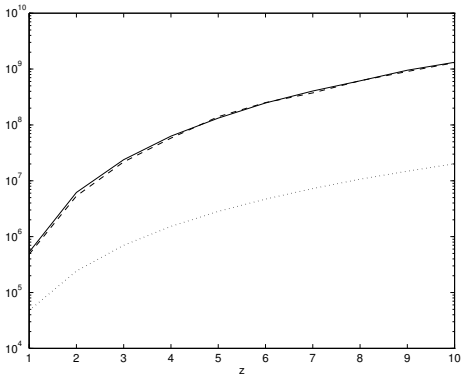


Figure 11.1: Number of floating point operations for CGA (full line), SAA (dashed line) and PIA (dotted line), for matrices with different sizes and a block structure, consisting of two repeated scalar blocks of size z and two full blocks of size z, where $z = 1, \ldots, 10$.

Figure 11.2 shows the computational cost for the CGA (similar results are obtained with the SAA), when the eigenvectors and eigenvalues are computed with power iterations or inverse iterations (see section 11.3). For both power iterations and inverse iterations classical convergence tests [48] were used with a tolerance of $1E-4$. For power iterations an upper limit of 50 iterations was set. For inverse iterations this limit was set to 10 iterations. If the limit is reached the eigenvectors and eigenvalues are calculated with the MATLAB function "eig". Fig. 11.2 shows the result for matrices of different sizes, generated in the same way as for the comparison of the SAA, CGA and PIA, described above. Note that power iterations and inverse iterations require approximately the same number of operations.

Another aspect is the quality of the obtained lower bound. To this end we have compared the PIA and CGA (similar results are obtained for the SAA) with random or SVD based initial value, for 1000 random matrices, generated in the same way as for the comparison of the computational cost, described avove. The SVD based initial value is programmed in the MATLAB μ-toolbox, and was heuristically adapted for use with the CGA. For each matrix it was noted which ones of the 4 algorithms obtained the greatest lower bound or a lower bound within 0.1% from the greatest one obtained. It turns out that for random initial values, the CGA performs as well as the PIA (according to table 11.1 the CGA performs slightly but not significantly better), while the initial value gives a better improvement for the PIA than for the CGA. We leave it as an open problem to find better initial values for the CGA and SAA.

Fig. 11.3a shows an example of the time evolution of $\mathrm{Re}\,\tilde{\lambda}$ (dashed line $- - -$),

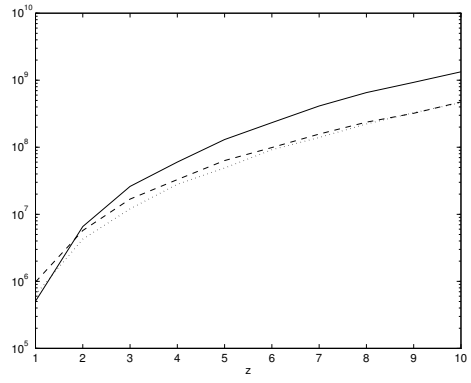


Figure 11.2: Number of floating point operations for SAA, with eigenvector/eigenvalue computation with the MATLAB function "eig" (full line), with power iterations (dashed line), and with inverse power iterations (dotted line).

	PIA	CGA
Random initial value:	54.5%	55.7%
SVD based initial value:	84.7%	68.1%

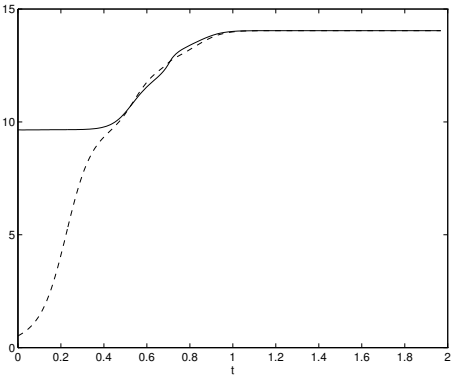
Table 11.1: Comparison between the PIA and CGA with random or SVD based initial values. Number of times the greatest of the four lower-bounds was reached by each method, within 0.1% accuracy, on 2000 experiments.

obtained with the continuous-time algorithm 11.13 where τ was determined by (11.12). The initial values $X(0) \in \mathbb{C}^{7\times 7}$, and $p(0), v(0) \in \mathbb{C}^7$ were chosen at random. The block structure consists of a full block of size 3 and two repeated scalar blocks of size 2. On the same figure also $\mathrm{Re}\,\bar{\lambda}$ is plotted (full line). One sees that because of the random initial values, it takes some time before $\tilde{\lambda}$ finds $\mathrm{Re}\,\bar{\lambda}$, but afterwards it manages to follow. Fig. 11.3b also shows the value of τ (11.12). On more than 200 experiments, no divergence of the algorithm was observed.

11.7 Conclusion

In this chapter we have derived continuous-time and discrete-time algorithms for computing the structured singular value. The algorithms implement ascent methods on a manifold of structured unitary matrices. Unlike the existing PIA algorithm, the new discrete-time algorithms always find a nontrivial lower-bound. However, they are less efficient than the PIA algorithm and we propose to combine both algorithms. We have also considered continuous-time algo-

a)



b)

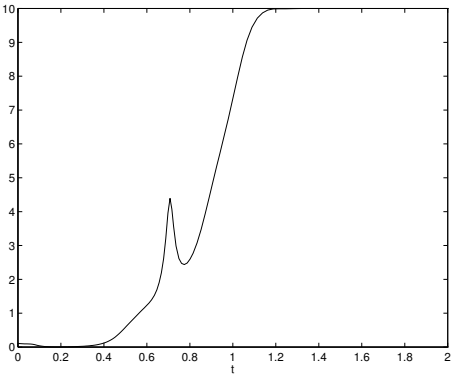


Figure 11.3: example of the behavior of continuous-time algorithm 11.13 with randomly chosen initial values $Q(0), \tilde{p}(0)$ and $\tilde{v}(0)$ and a block structure consisting of a full block of size 3 and two repeated scalar blocks of size 2. The value of τ was chosen as in (11.12). a) evolution of $\mathrm{Re}\,\bar{\lambda}$ (full line —) and $\mathrm{Re}\,\tilde{\lambda}$ (dashed line), b) evolution of τ.

rithms. These algorithms proved useful for early experiments, and give rise to simple parallel signal flow graphs. However, in this case, they do not lead to parallel discrete-time algorithms.

Chapter 12

Conclusion

By presenting our results in the general context of continuous-time matrix algorithms, we may have raised questions like 'What makes the continuous-time viewpoint successful' and more important 'To what extent can the presented ideas be generalized for application to other problems'. In this last chapter, we try to find answers in our experience with the presented material, combining issues from different chapters, and discussing both difficulties and possibilities. Meanwhile, we also formulate some open problems. Of course, many aspects of the continuous-time viewpoint to matrix algorithms have been left untouched in this work, and the last word about it has not yet been written.

Section 12.1 discusses the issue of parallelism in continuous-time algorithms. Section 12.2 considers *analog* realization and section 12.3 considers the design of parallel *discrete-time* algorithms. In section 12.4 we look at the discretization issue focusing on optimization algorithms. Section 12.5 comments on the analysis of algorithms with continuous-time tools and in section 12.6 we say a last word about neural network aspects. Finally section 12.7 summarizes the main contributions.

12.1 Parallelism

First we discuss the issue of parallelism in continuous-time algorithms, which is linked to three topics of this work, analog realization, design of parallel discrete-time algorithms and neural network interpretations.

Parallelism is a natural, often inevitable, feature of the realization of a continuous-time algorithm. Because the algorithm is described by giving the evolution (the

'velocity') of the current state, as a function of this state and possibly the current input, the whole right hand side (the velocity) of the algorithm is to be evaluated instantaneously in order to proceed. For most algorithms in this work this has turned out to yield simple and regular signal flow graphs. We attribute this fact to the relatively simple form of the formulas for tangent spaces and derivatives of matrix decompositions in chapters 2 and 3. However some cases were more complicated. Algorithm 11.10, for example, assumes the instantaneous parallel calculation (or at least tracking) of eigenvectors. The algorithm was brought into a simpler form (algorithm 11.13), using the continuous-time power algorithm. For $\tau \to 0$, this algorithm can be thought of as operating on two time levels. The equations of the power algorithm should actually converge (on the fast time scale) before the 'velocity' of the matrix Q (on the slow time scale) is known. Nevertheless the algorithm makes sense for bigger values of τ also, if the algorithm on the slow time scale converges. For in that case, the algorithm on the fast time scale can come closer to convergence, and in the end, the combined algorithm can converge to the same equilibrium point. New theoretical arguments may be necessary to analyze the behavior when the algorithm on the slow time scale is far from convergence.

This idea of different time scales is also often present in recurrent neural network models. The time scales then correspond to the operation level and the adaptation level (section 9.1). An example is the simultaneous implementation of the adaptation laws and operation laws discussed in section 10.5. Also the realization of the division in section 6.6 with a differential amplifier fits in this context. The algebraic loop containing the amplifier is in reality a dynamical system converging on a faster time scale. By exploiting these different time scales one could extend the uniform parallel systems of definition 5.1 to systems that can calculate vectors like $A^{-T}x$ also if A is not triangular but for instance positive (or negative) definite. An example of such a system is the neural independent component analyzer of [63, 18].

As a concluding remark concerning the natural parallelism of continuous-time algorithms we recall ideas of chapter 10. A problem of continuous-time algorithms, is that because the right hand side must be evaluated instantaneously, the complexity of the algorithm appears as circuit complexity. In chapter 10, we have shown how this spatial complexity can sometimes be exchanged with time complexity. The core of the idea could be formulated as 'substituting a matrix by a simpler time-varying one, still descending a considered Lyapunov function'. To further generalize this idea, a statistic framework might also be appropriate. In such a framework, a more general idea could be formulated as 'substituting a matrix by a simpler stochastic matrix, whose expectation equals the substituted matrix'. If that matrix is simply an identity matrix one can work with white noise signals. However this formulation is too broad and does not guarantee success. We leave further investigation as an open problem.

12.2 Analog realizations

The topic of *analog realization* has been addressed in section 6.6 for two simple algorithms, the continuous-time Gentleman-Kung algorithm, and its LU based variants. We have only considered high-level aspects. Analog realizations are not only desirable because they can be cheap and fast but another advantage is that for some problems, analog algorithms and their signal flow graphs can be derived rather easily, while discrete-time counterparts may not be easy to find, or may not show the same degree of parallelism. As examples, we cite continuous-time algorithms for the structured singular value (section 11.5), some of the neural network adaptation laws of chapter 10, the square-root QR flow (algorithm 4.10), the original QR flow and the double bracket flow. However the main problems are limited accuracy, and the need for application specific design. Analog realizations of the double bracket flow are reported in [89] (for a special case) and [93]. However they are only feasible for small matrices and have low accuracy. Nevertheless there are good reasons to believe in the feasibility of the realization of algorithms of section 6.6 as algorithm 6.20 consists of essentially the same building blocks as LMS algorithms, which have already been realized [67]. The successfully realized, neural independent component analyzer of [63, 18] (for 5 input signals) is even much more complex.

As the main problem of analog realizations is accuracy, an appropriate domain of applications may be preprocessing for pattern recognition. For these operations, the output of the preprocessing often need not be very accurate, as long as it contains sufficient information to distinguish between the categories to be recognized.

12.3 Systolic algorithms

In this work we have paid special attention to the *link between discrete-time systolic algorithms* and continuous-time algorithms with uniform parallel signal flow graphs. In chapter 6 we have derived continuous-time algorithms, that turned out to be continuous-time versions of existing systolic algorithms. In chapter 8 we have exploited the natural parallelism of continuous-time algorithms for the design of *new* systolic arrays.

The design of such an array can be split up in three steps. In a *first* step a signal flow graph for the continuous-time algorithm is derived. The *second* step converts it into a signal flow graph for the discrete-time algorithm. In the *third* step the algorithm is pipelined to obtain a systolic algorithm. The first step has been discussed above (section 12.1). We discuss the second and third step.

The *second* step is not trivial. In all given examples, the correspondence with

systolic algorithms is obtained by exact integration of the continuous-time algorithm with piecewise constant input. However, it is not straightforward to obtain this integration. For the new systolic algorithms of chapter 8 they were obtained by analogy with the results of chapter 6. Moreover, the integration was not done locally by integration of the equations of continuous-time cells to obtain discrete-time cells, but globally on the matrix equations. Therefore, it was not a priori clear what a signal flow graph for the discrete-time version would look like, although it was reasonable to assume the existence of a signal flow graph which becomes equivalent to the continuous-time signal flow graph in the limit for $T_s \rightarrow 0$. However, there exist systolic algorithms that cannot easily be translated into continuous-time algorithms. In the Jacobi-type SVD updating array of [76, 78] for example, matrix entries are not fixed to a cell but travel around as the columns and rows of the matrix are permuted. This way operations involving non-neighboring entries can be implemented as local operations involving neighboring cells. In the limit for $T_s \rightarrow 0$, the effect of the permutation scheme does not vanish, so that 'infinitesimal steps should have finite effects'.

Integrating the cells of the continuous-time signal flow graph to directly obtain the blocks of the discrete-time signal flow graph is possible for the presented cases but not trivial. The main problem is that the assumption of piecewise constant input is not valid for the cells separately, because even when the external input is constant, separate cells can get nonconstant input from other cells. To find a discrete-time version, one has to capture nonconstant signals between cells (over an interval $[t^{[k]}, t^{[k+1]}]$ of constant external input) in one single scalar to be communicated between discrete-time cells. Further investigation of this issue might shed light on the question when the structure of the continuous-time signal flow graphs can be exploited to find discrete-time algorithms. We leave this as an open problem.

However, note that the exact integration with piecewise constant input is certainly not the only way to derive discrete-time algorithms from continuous-time versions. In section 12.4 we come back to this issue.

The *third* step in the conversion of a continuous-time algorithm into a systolic algorithm consists of pipelining the signal flow graph. This is entirely a discrete-time problem. However, the strategy of storing older versions of the matrix in lower rows and introducing a correction scheme (8.2), has a more transparent counterpart, in the 'continuous-place-coordinate' picture, that is by thinking in terms of a partial differential equation instead of a matrix equation (this view is related to neural field equations, or maybe 'synaptic field' would be more appropriate. There may also be a relation with wave digital filtering [43]). The 'pipelined' version is obtained by a substitution of the time and place coordinates. However this approach introduces a new integration problem to find the 'discrete-place-coordinate' version (or a new realization problem, to realize the partial differential equation). We leave further investigation as an open problem.

As a final remark, with hindsight one could also give discrete-time derivations of the new discrete-time algorithms without referring to the continuous-time viewpoint but that should be no surprise, as there is nothing continuous in the final result. The constructive proof of theorem 8.1 can be regarded as such a derivation by analogy with the (simpler) continuous-time derivation. Nevertheless, the fact that the continuous-time approach often provides surprisingly different viewpoints and simpler derivations can only foster inspiration.

The relevance of the continuous-time viewpoint for the *design* of *parallel* algorithms, can then be summarized as follows. The simple parallelization and the link between continuous-time algorithms and known systolic algorithms has provided inspiration for the derivation of new systolic algorithms. For the LU based algorithm 8.4, the continuous-time algorithm has also allowed to quickly derive and compare signal flow graphs for different versions, storing uf(A) or iuf(A) for instance. The link between the discrete-time algorithm and its continuous-time version also makes it possible to derive discrete-time algorithms which directly inherit the results of the continuous-time analysis, further discussed in section 12.5.

For both the analog realization and the conversion into parallel discrete-time algorithms, exploitation of the natural parallelism of continuous-time algorithms is nontrivial. Nevertheless, this work has shown that both may be possible. As for analog realizations, algorithms need to be simple, and applications should not be too dependent on a high degree of accuracy. For generalizations of the derivation of parallel discrete-time algorithms, some problems in signal processing with higher-order statistics may be realistic candidates. Many problems in this area take similar forms as the signal processing problems considered in this work. A typical problem for instance consists in diagonalizing a symmetric tensor with higher-order cumulants (generalizing $E\{xx^T\}$) by multiplication from different sides with orthogonal matrices. Investigation of this type of problem may also shed new light on the convergence behavior of the neural independent component analyzer, already mentioned above.

12.4 Discrete-time optimization algorithms

In this section we consider the discretization aspect. We concentrate on optimization algorithms. Some issues are also relevant for the discretization of parallel algorithms, discussed above.

Integration of continuous-time algorithms by a standard integration packages is a useful tool for experiments during the design stage, for example for comparing the values of different local minima obtained. However in general this does not lead to efficient algorithms.

In chapter 11 we have applied the steepest ascent and conjugate gradient method, by line search along geodesics. This approach was discussed for general Riemannian manifolds in [94]. However in practice the methods are mainly useful for regular manifolds like Lie groups and homogeneous spaces, because cheap formulas are needed for the computation of geodesics. Also a generalization of the Newton method is possible, but this assumes the knowledge of Hessians and often leads to complicated formulas. A problem with this method, as with many methods, is that the convergence results for the discrete-time algorithms are more involved than for their continuous-time counterparts (gradient-flows), and they can depend on subtle characteristics of the line search. However, this remark is not specific for the case of optimization on manifolds. Moreover, for the structured singular value this problem turned out not to be very important since we are interested in the value of the optimum and not the point where it is reached.

We believe that the idea of optimization algorithms 'living' in regular manifolds, can be easily extended to algorithms that are *attracted* toward these manifolds. To this end, the viewpoint of matrix decompositions as nonlinear coordinates is fruitful. As an example, we think of an important problem in signal processing and system identification, of approximating a Hankel matrix by a rank deficient Hankel matrix. One can for instance design an algorithm that lives in the subspace of Hankel matrices, evolves towards the matrix to be approximated while decreasing its minimal singular value to realize attraction to the set of rank deficient matrices. This leads to algorithms with a relatively simple form, but not efficient. To compute directions with decreasing minimal singular value one has to calculate a left and right singular vector (much in the way a left and right eigenvector were needed for the structured singular value problem). However approximations as in section 11.5 may be possible.

Other discretization approaches interpret the issue more freely, preserving the main ideas of descending the objective function and staying on the manifold, exploiting as much as possible the insight gained from the geometric approach. We mention an approach, choosing search directions for which the line search is computationally cheap. On the orthogonal group, one can for instance choose directions $Q\Omega$ with Ω of rank 2. Updates then amount to plane rotations. In the special case $\Omega = \omega(E^{(i,j)} - E^{(j,i)})$, these are Givens rotations (or Jacobi rotations on the isospectral manifolds). In [54] a globally convergent Jacobi-like algorithm is described, optimizing the objective function of the double bracket flow and also sorting the eigenvalues. We also mention [74] and [62].

It can be proved that the integration by assuming piecewise constant input in chapter 8 can be interpreted (in the specific case of algorithm C) as putting steps along geodesics in the direction of the gradient of $\mathrm{Tr}A^T x^{[k]} x^{[k]^T} A$ (the 'stochastic gradient directions'). A main difference with the above approach is that we do not optimize along this geodesic. This would amount to rotating the

signal subspace estimate $\mathrm{col}(A)$ until $x^{[k]}$ lies in it. In the presence of noise this is not an optimal choice. This point of view can lead to a generalization of our approach in cases where exact integration for constant input is not possible.

12.5 Analysis of continuous-time algorithms

Concepts from continuous-time system theory and differential geometry can be used in many ways for the analysis of algorithms. Convergence proofs using Lyapunov functions, or stability analyses of equilibria for systems on manifolds, often take simpler forms in continuous-time formulations. In this work, we have put emphasis on a new analysis, obtained by tracking matrix decompositions. Given the evolution of a matrix as a velocity, to know the evolution of a matrix factor one has only to apply a linear transformation (the derivative of that matrix factor) to the given velocity. We have applied this idea to prove attraction to orthogonality and to track canonical angles between subspaces in chapter 7. Also in chapter 9 it proved a useful tool to analyze the evolution of both eigenvalues and eigenvectors for a class of neural network adaptation laws.

For the subspace tracking problem of chapter 7, we have presented our analysis as an alternative to the approach using [73] and [70]. Our deterministic approach has the advantage that the results are not only valid asymptotically and for slowly vanishing step sizes (conditions 7.11). However, the continuous-time analysis can also be complementary to the method of [73] and [70]. Tracking canonical angles, for example, can also be applied to the autonomous equation with $E\{xx^T\}$ instead of xx^T. In this case, the stochastic approach even avoids details related to multiple eigenvalues and the example of interpretation 7.34. Finally, we refer again to the open problem, formulated in section 7.5, concerning the extension of our continuous-time analysis to a stochastic framework.

12.6 Neural network adaptation laws

The field of neural networks has provided new impulses to the idea of analog computation and more generally to the idea of studying computation from a dynamical systems point of view. Although it is not clear whether the principal working mechanisms of biological neural networks will eventually be explained in continuous-time or discrete-time terms, the transition between the physical continuous-time description and the discrete-time nature of some of the problems to be solved is certainly organized in another way than with classical computers.

An important characteristic of neural information processing is that it is realized by parallel operation of very simple building blocks. In this work we have

shown how the natural parallelism (see 12.2) of continuous-time algorithms easily gives rise to such neural-network-like structures. We have shown how some classical signal processing operations can be mapped onto neural networks with generalized Hebbian adaptation laws.

We have already mentioned problems in higher-order signal processing, as realistic candidates for generalization. Another obvious generalization, but not trivial to realize, is to try and use similar strategies for nonlinear estimation problems.

For associative memory applications, we have discussed adaptation laws, capable of storing patterns in memory, shown by a teaching signal. Although our adaptation laws perform as well as off-the-net design algorithms and better than known on-the-net adaptation algorithms, the problem of organizing neural network memories for real applications is still a huge problem. The idea of letting nonlinear systems adapt so as to reproduce shown behavior is not easy to realize in a robust and reliable way, and we prefer not to speculate about the eventual solution of this problem.

12.7 Main contributions

Finally, we summarize the main contributions of this work. More elaborate summaries can also be found in the conclusions to the different chapters.

We have given an introduction to continuous-time matrix algorithms, putting special emphasis on formulas for tracking matrix decompositions, and the corresponding interpretations of matrix decompositions as nonlinear coordinate transformations of matrix space.

We have paid much attention to *parallel* realizations of algorithms. We have described a class of continuous-time matrix algorithms (definition 5.1), with uniform parallel signal flow graphs. These signal flow graphs consist of simple identical cells (or a limited number of different types) and are easily derived from the formal description of the algorithms. They can provide a basis for the design of *analog realizations* or for the design of *parallel discrete-time* algorithms. They also have a *neural network* interpretation. Moreover the continuous-time algorithms of the class can be studied using the presented tools from continuous-time matrix algorithms. We have considered applications in adaptive signal processing.

We have derived three continuous-time algorithms (and some variants) for recursive least-squares estimation, using the results for tracking matrix decompositions. These algorithms fit in the class of definition 5.1, and have simple signal flow graphs. They are continuous-time versions of discrete-time systolic algorithms. Because of their simple form, two algorithms are candidates for analog

realization. We have considered high level aspects of this topic.

For subspace tracking applications, we have shown how the continuous-time viewpoint provides a framework for unifying seemingly different algorithms. We have concentrated on algorithm C (algorithm 7.10) and given a continuous-time analysis for this algorithm, showing attraction of the matrix state to orthogonality and describing the evolution of the signal subspace estimate in terms of canonical angles. The uniform parallel signal flow graph of algorithm C, has been exploited to derive a new discrete-time systolic algorithm for subspace tracking, algorithm D (algorithm 8.3 and some variants), with a similar signal flow graph and inheriting the main properties of algorithm C. The obtained signal flow graph can be pipelined horizontally and vertically to obtain problem size independent throughput.

From the viewpoint of *neural networks*, the algorithms of definition 5.1, form a class of *adaptation laws*, that can be realized on the network, and at the same time lend themselves for mathematical analysis of the global behavior. We have given neural network interpretations to the continuous-time algorithms for recursive least-squares estimation and subspace tracking. We have also derived new 'on-the-net' adaptation laws for associative memory applications, storing patterns shown by a teaching signal, and performing as well as 'off-the-net' design algorithms.

Finally we have considered an optimization problem from robust control theory, the computation of the structured singular value. We have again derived continuous-time algorithms with simple signal flow graphs, but we have concentrated on the derivation of discrete-time algorithms without considering parallelization. We have derived a steepest ascent algorithm and a conjugate gradient algorithm, on a manifold of structured unitary matrices. The algorithms are less efficient than the existing power iteration algorithm, but unlike the latter always find a nontrivial lower-bound for the structured singular value.

Continuous-time matrix algorithms offer to those who appreciate it a geometric and inspiring view on the design, analysis and parallel realization of matrix algorithms, often with neural network interpretations. Although analog realizations seem only realistic in simple cases and the conversion into discrete-time algorithms is not a priori straightforward, different approaches exist and successful examples were given. We have shown how the continuous-time approach catches some classical adaptive signal processing operations, and associative memory learning procedures in a class of local adaptation laws for neural networks. As more concrete results this work offers a continuous-time algorithm for RLS related applications, candidate for analog realization, a new efficient discrete-time systolic algorithm for subspace tracking and a new discrete-time algorithm for computing the structured singular value.

Bibliography

[1] J.A. Anderson, J.W. Silverstein, S.R. Ritz, and R.S. Jones, "Distinctive features, categorical perception, and probability learning: some applications of a neural model," *Psychological Review*, vol. 84, pp. 413–451, 1977.

[2] G.J. Balas, J.C. Doyle, K. Glover, A. Packard, and R. Smith, *μ-Analysis and Synthesis Toolbox*. The MathWorks Inc., Natick, Mass. 01760 USA, 1993.

[3] P. Baldi, "Linear learning: Landscapes and algorithms," in *Advances in neural information processing systems I. Proceedings of the 1988 NIPS Conference, Denver, CO* (D.S. Touretzky, ed.), Morgan Kaufmann, 1988.

[4] D.A. Bayer and J.C. Lagarias, "The nonlinear geometry of linear programming. I. Affine and projective scaling trajectories," *Transactions of the American Mathematical Society*, vol. 314, no. 2, pp. 499–526, August 1989.

[5] D.A. Bayer and J.C. Lagarias, "The nonlinear geometry of linear programming. III. Projective Legendre transform coordinates and Hilbert geometry," *Transactions of the American Mathematical Society*, vol. 314, no. 2, pp. 527–581, August 1989.

[6] R. Bhatia, "Matrix factorizations and their perturbations," *Linear Algebra and its Applications*, vol. 197,198, pp. 245–276, 1994.

[7] A.M. Bloch, R.W. Brockett, and T. Ratiu, "A new formulation of the generalized Toda lattice equations and their fixed point analysis via the moment map," *Bulletin of the American Mathematical Society*, vol. 23, pp. 447–456, 1990.

[8] R.W. Brockett, "Dynamical systems that sort lists and solve linear programming problems," in *Proceedings of the 27th IEEE Conference on Decision and Control Austin, Texas*, pp. 779–803, 1988.

[9] R.W. Brockett, "Least squares matching problems," *Linear Algebra and its applications*, vol. 122/123/124, pp. 761–777, 1989.

[10] R.W. Brockett, "Dynamical systems that learn subspaces," in *Mathematical system theory: The influence of R.E. Kalman*, pp. 410–420, Springer Verlag, 1990.

[11] R.W. Brockett, "Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems," *Linear Algebra and its Applications*, vol. 146, pp. 79–91, 1991.

[12] R.W. Brockett, "Differential geometry and the design of gradient algorithms," in *Proceedings of Symposia in Pure Mathematics*, vol. 54, pp. 69–91, 1993.

[13] A. Bunse-Gerstner, R. Byers, V. Mehrman, and N.K. Nichols, "Numerical computation of an analytic singular value decomposition of a matrix valued function," *Numerische Mathematik*, vol. 60, pp. 1–40, 1991.

[14] D.R.J. Chillingworth, *Differential topology with a view to applications*, vol. 9 of *Research notes in mathematics*. Pitman Publishing, 1977.

[15] M.T. Chu, "The generalized Toda flow, the QR algorithm and the center manifold theory," *Siam J. Alg. Disc. Methods*, vol. 5, pp. 187–201, 1984.

[16] M.T. Chu, "On the continuous realization of iterative processes," *SIAM Review*, vol. 30, no. 3, pp. 375–387, September 1988.

[17] M.T. Chu and K.R. Driessel, "The projected gradient method for least squares matrix approximations with spectral constraints," *SIAM Journal of Numerical Analysis*, vol. 27, no. 3, pp. 1050–1060, August 1990.

[18] M.H. Cohen and A.G. Andreou, "Analog CMOS integration and experimentation with an autoadaptive independent component analyzer," *IEEE Transactions on Circuits and Systems-II*, vol. 42, no. 2, pp. 65–77, February 1995.

[19] P. Comon, C. Jutten, and J. Hérault, "Blind separation of sources, Part II: Problems statement," *Signal Processing*, vol. 24, no. 11-20, 1991.

[20] L. Debo and M. Pattyn, "Adaptieve verwerking van antennesignalen aan de hand van continue parallelle algoritmen," Master's thesis, K.U.Leuven, May 1994.

[21] R.D. DeGroat, "Noniterative subspace updating," *IEEE Transaction on Signal Processing*, vol. 40, no. 3, pp. 571–577, 1992.

[22] R.D. DeGroat and E.M. Dowling, "Sphericalized subspace tracking: convergence and detection schemes," in *Proceedings of the 26th annual Asilomar conference*, pp. 561–565, 1992.

[23] J. Dehaene, "Performance analysis of a subspace tracking algorithm. Request for a stochastic interpretation." IUAP 17 Study Day, Louvain-la-Neuve, October 5 1994. abstract only.

[24] J. Dehaene, M. Moonen, and J. Vandewalle, "Uniform local implementation on networks and arrays of continuous time matrix algorithms for adaptive signal processing," Tech. Rep. 1993-34I, ESAT-SISTA K.U.Leuven, 1993.

[25] J. Dehaene, M. Moonen, and J. Vandewalle, "A continuous time approach to the analysis and design of parallel algorithms for subspace tracking," in *SVD and Signal Processing, III, Algorithms, Applications and Architectures* (M. Moonen and B. De Moor, eds.), pp. 259–266, Elsevier, 1995.

[26] J. Dehaene, M. Moonen, and J. Vandewalle, "Continuous-time recursive least-squares estimation, adaptive neural networks and systolic arrays," *IEEE Transactions on Circuits and Systems-I*, vol. 42, no. 2, pp. 116–119, February 1995.

[27] J. Dehaene and J. Vandewalle, "Adaptive signal processing with unidirectional Hebbian adaptation laws,"

[28] J. Dehaene and J. Vandewalle, "Attraction basins in Hopfield type neural networks, a geometrical interpretation," Tech. Rep. 1992-08I, ESAT-SISTA K.U.Leuven, 1992. available via anonymous ftp from ftp@esat.kuleuven.ac.be as /pub/SISTA/dehaene/reports/92-08I.ps.Z.

[29] J. Dehaene and J. Vandewalle, "A geometrical approach to the analysis of the dynamical behaviour of neural networks with simple piecewise linear limiters," in *Proceedings of the IEEE international workshop on Cellular Neural Networks and Applications, CNNA-92, München*, pp. 234–239, 1992.

[30] J. Dehaene and J. Vandewalle, "A new approach to the design of associative memories for three related neural models," Tech. Rep. 1992-15I, ESAT-SISTA K.U.Leuven, 1992. available via anonymous ftp from ftp@esat.kuleuven.ac.be as /pub/SISTA/dehaene/reports/92-15I.ps.Z.

[31] J. Dehaene and J. Vandewalle, "A new geometric approach to the design of neural associative memory," in *Proceedings of the second International Conference on Artificial Neural Networks ICANN-92, 4-7 September 1992, Brighton*, pp. 207–211, 1992.

[32] J. Dehaene and J. Vandewalle, "Learning with matrix flows, implementable on a network structure," in *Proceedings of the European Conference on Circuit Theory and Design, ECCTD 93, Davos, Switzerland* (H. Dedieu, ed.), pp. 1471–1476, Elsevier, 1993.

[33] J. Dehaene and J. Vandewalle, "Locally implementable learning with isospectral matrix flows," in *Proceedings of the European Symposium on Artificial Neural Networks, Brussels, April 7-8-9, 1993* (M. Verleysen, ed.), pp. 189–194, D facto conference services, 1993.

[34] J. Dehaene and J. Vandewalle, "Adaptive signal processing with unidirectional Hebbian adaptation laws," in *Proceedings of the Belgian Neural Network Contact Group, Louvain-la-Neuve, November 30, 1994* (E. De Bodt and M. Verleysen, eds.), pp. 81–88, D facto publications, 1994.

[35] J. Dehaene and J. Vandewalle, "Locally implementable matrix flows and their application to neural network learning," *Journal of Circuits, Systems and Computers*, vol. 4, no. 1, pp. 23–51, 1994.

[36] J. Dehaene and J. Vandewalle, "Uniform local implementation on networks and arrays of continuous time matrix algorithms for adaptive signal processing," in *Systems and Networks: Mathematical Theory and Applications, Proceedings of the International Symposium MTNS '93, Regensburg, Germany, August 2-6, Volume II, Invited and Contributed Papers* (U. Helmke, R. Mennicken, and J. Saurer, eds.), pp. 651–654, Akademie Verlag, 1994. A more extensive account is available via anonymous ftp from ftp@esat.kuleuven.ac.be as /pub/SISTA/dehaene/reports/92-34I2.ps.Z.

[37] J. Dehaene, C. Yi, and B. De Moor, "Calculation of the structured singular value with gradient based optimization algorithms on a Lie group of structured unitary matrices," Tech. Rep. 1994-46I, ESAT-SISTA K.U.Leuven, 1994. submitted for publication.

[38] P. Deift, T. Nanda, and C. Tomei, "Ordinary differential equations and the symmetric eigenvalue problem," *SIAM Journal of Numerical Analysis*, vol. 20, no. 1, pp. 1–22, February 1983.

[39] E.F. Deprettere, ed., *SVD and Signal Processing, Algorithms, Applications and Architectures*. Norht-Holland, 1988.

[40] E.M. Dowling and R.D. DeGroat, "Adaptation dynamics of the spherical subspace tracker," *IEEE Transactions on Signal Processing*, vol. 40, no. 10, pp. 2599–2602, October 1992.

[41] E.M. Dowling, R.D. Degroat, D.A. Linebarger, and H. Ye, "Sphericalized SVD updating for subspace tracking," in *SVD and Signal Processing, III, Algorithms, Applications and Architectures,* (M. Moonen and B. De Moor, eds.), pp. 227–234, Elsevier, 1995.

[42] J.C. Doyle, "Analysis of feedback systems with structured uncertainties," *IEE Proceedings*, vol. 129, pp. 242–250, 1982.

[43] A. Fettweis, "Wave digital filters: Theory and practice," *Proceedings of the IEEE*, vol. 74, no. 2, pp. 270–327, February 1986.

[44] R. Fletcher, *Practical Methods of Optimization.* Chichester: Wiley & Sons, 1993.

[45] C.B. Garcia and W.I. Zangwill, "An approach to homotopy and degree theory," *Mathematics of Operations Research*, vol. 4, no. 4, pp. 390–405, November 1979.

[46] W.M. Gentleman, "Least squares computations by givens transformations without square roots," *J. Inst. Maths Applics*, vol. 12, pp. 329–336, 1973.

[47] W.M. Gentleman and H.T. Kung, "Matrix triangularization by systolic arrays," *Real-Time Signal Processing IV, Proc. SPIE*, vol. 298, pp. 19–26, 1981.

[48] G. Golub and C.F. Van Loan, *Matrix computations.* Johns Hopkins University Press, 1991.

[49] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Networks*, vol. 2, pp. 17–61, 1988.

[50] S. Haykin, *Adaptive filter theory.* Englewood Cliffs: Prentice Hall, 1991.

[51] D.O. Hebb, *The Organization of Behavior.* New York: John Wiley & suns, 1949.

[52] S. Helgason, *Differential Geometry, Lie Groups and Symmetric Spaces.* New York: Academic Press, 1978.

[53] U. Helmke, "Isospectral flows on symmetric matrices and the Riccati equation," *Systems & Control Letters*, vol. 16, pp. 159–165, 1991.

[54] U. Helmke, "Isospectral matrix flows for numerical analysis," in *SVD and Signal Processing, III, Algorithms, Applications and Architectures,* (M. Moonen and B. De Moor, eds.), pp. 33–59, Elsevier, 1995.

[55] U. Helmke and J.B. Moore, *Optimization and Dynamical Systems.* New York: Springer Verlag, 1993.

[56] N.J. Higham, "Computing the polar decomposition - with applications," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 4, pp. 1160–1174, October 1986.

[57] M.W. Hirsch and S. Smale, *Differential Equations, Dynamical Systems, and Linear Algebra.* New York: Academic Press, 1974.

[58] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences USA*, vol. 79, pp. 2254–2258, 1982.

[59] J.J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences USA*, vol. 81, pp. 3088–3092, May 1984.

[60] R.A. Horn and C.R. Johnson, *Matrix Analysis.* Cambridge University Press, 1985.

[61] K. Hornik and C.M. Kuan, "Convergence analysis of local feature extraction algorithms," *Neural Networks*, vol. 5, pp. 229–240, 1992.

[62] K. Hüper, J. Götze, and S. Paul, "Subspace separation by discretizations of double bracket flows," in *SVD and Signal Processing, III, Algorithms, Applications and Architectures,* (M. Moonen and B. De Moor, eds.), pp. 251–258, Elsevier, 1995.

[63] C. Jutten and J. Hérault, "Blind separation of sources, Part I: An adaptive algorithm based on neuromimetic architecture," *Signal Processing*, vol. 24, pp. 1–10, 1991.

[64] I. Karasalo, "Estimating the covariance matrix by signal subspace averaging," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. AASP-34, no. 1, pp. 8–12, 1986.

[65] T. Kato, *A short introduction to Perturbation Theory for Linear Operators.* Springer-Verlag, 1982.

[66] T. Kohonen, *Self Organization and Associative Memory.* Springer-Verlag, 1988.

[67] F.J. Kub and E.W. Justh, "Analog CMOS implementation of high frequency least-mean-square error learning circuit," in *Proceedings of the IEEE international Solid-State Circuits Conference ISSCC, 15-17 February 1995*, pp. 74–75, 1995.

[68] S.Y. Kung, K.I. Diamantras, and J.S. Taur, "Neural networks for extracting pure/constrained/oriented principal components," in *SVD and Signal Processing, II, Algorithms, Applications, and Architectures* (Richard J. Vaccaro, ed.), pp. 57–81, Elsevier, 1991.

[69] P. Kunkel and V. Mehrmann, "Smooth factorization of matrix valued functions and their derivatives," *Numerische Mathematik*, vol. 60, pp. 115–132, 1991.

[70] H.J. Kushner and D.S. Clark, *Stochastic Approximation Methods for Constrained and Unconstrained Systems.* New York: Springer-Verlag, 1978.

[71] J.C. Lagarias, "The nonlinear geometry of linear programming. II. Legendre transform coordinates and central trajectories," *Transactions of the American Mathematical Society*, vol. 320, no. 1, pp. 193–225, July 1990.

[72] J.H. Li, A.N.Michel, and W. Porod, "Analysis and synthesis of a class of neural networks: Linear systems operating on a closed hypercube," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 11, pp. 1405–1422, November 1989.

[73] L. Ljung, "Analysis of recursive stochastic algorithms," *IEEE Transactions on automatic control*, vol. AC-22, no. 4, pp. 551–575, August 1977.

[74] R. Mahony, *Optimization Algorithms on Homogeneous Spaces.* PhD thesis, Australian National University, March 1994.

[75] J.G. McWhirter, "Recursive least-squares minimisation using a systolic array," in *Proceedings of SPIE, Real time signal processing VI, 1983*, p. 431, 1983.

[76] M. Moonen, *Jacoby-type updating algorithms for signal processing, systems identification and control.* PhD thesis, ESAT-SISTA K.U.Leuven, November 1990.

[77] M. Moonen and B. De Moor, eds., *SVD and Signal Processing III, Algorithms, Applications and Architectures.* Elsevier, 1995.

[78] M. Moonen, E. Deprettere, I.K. Proudler, and J.G. McWhirter, "Pipelined filter structures for adaptive eigenvalue and singular value decompositions," in *Proceedings of the International Conference on Acoustics Speech and Signal Processing, ICASSP 95*, 1995.

[79] M. Moonen, P. Van Dooren, and J. Vandewalle, "A singular value decomposition updating algorithm for subspace tracking," *SIAM Journal of Matrix Analysis and Applications*, vol. 14, no. 2, pp. 353–371, 1993.

[80] M. Moonen and J.G. McWhirter, "A systolic array for recursive least squares by inverse updating," *Electronics Letters*, vol. 29, no. 13, pp. 1217–1218, 1993.

[81] F. Moons and D. Leman, "Studie van de analoge IC-implementatie van een adaptief signaalverwerkingssysteem," Master's thesis, K.U.Leuven, May 1995.

[82] T. Nanda, "Differential equations and the QR algorithm," *SIAM Journal of Numerical Analysis*, vol. 22, no. 2, pp. 310–321, April 1985.

[83] E. Oja, "A simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, pp. 267–273, 1982.

[84] E. Oja, "Neural networks, principal components, and subspaces," *International Journal of neural Systems*, vol. 1, pp. 61–68, 1989.

[85] E. Oja, "Principal components, minor components, and linear neural networks," *Neural Networks*, vol. 5, pp. 927–935, 1992.

[86] E. Oja and J. Karhunen, "On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix," *Jounal of Mathematical Analysis and Applications*, vol. 106, pp. 69–84, 1985.

[87] A. Packard, M.K.H. Fan, and J. Doyle, "A power method for the structured singular value," in *Proceedings of the 27th Conference on Decision an Control, Austin, Texas, December 1988*, pp. 2132–2137, 1988.

[88] C.T. Pan and R.J. Plemmons, "Least squares modifications with inverse factorization: parallel implications," *Journal of Computational and Applied Mathematics*, vol. 27, no. 1-2, pp. 109–127, 1989.

[89] S. Paul and K. Hüper, "Analog rank filtering," Tech. Rep. TUMS-LNS-TR-91-21, Technical University Munich, November 1991.

[90] I.K. Proudler, J.G. McWhirter, M. Moonen, and G. Hekstra, "The formal derivation of a systolic array for recursive least squares estimation," *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Application*, 1995. to appear.

[91] T.D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Networks*, vol. 2, pp. 459–473, 1989.

[92] D.H. Sattinger and O.L. Weaver, *Lie Groups and Algebras with Applications to Physics, Geometry and Mechanics*, vol. 61 of *Applied Mathematical Sciences*. Springer-Verlag, 1993.

[93] N. Saxena and J.J. Clark, "Analogue system for eigenvalue computation and sorting based on an isospectral matrix flow," *Electronics Letters*, vol. 31, January 1995.

[94] S.T. Smith, *Geometric Optimization Methods for Adaptive Filtering*. PhD thesis, Harvard University, Cambridge, Massachusetts, May 1993.

[95] E. Sorouchyari, "Blind separation of sources, Part III: Stability analysis," *Signal Processing*, vol. 24, pp. 21–29, 1991.

[96] W.W. Symes, "The QR algorithm and scattering for the finite nonperiodic Toda lattice," *Physica 4D*, pp. 275–280, 1982.

[97] The MathWorks Inc., Natick, Mass. 01760 USA, *MATLAB Reference Guide*, 1992.

[98] Richard J. Vaccaro, ed., *SVD and Signal Processing, II, Algorithms, Applications, and Architectures*. Elsevier, 1991.

[99] F. Vanpoucke, *Algorithms and architectures for adaptive array signal processing*. PhD thesis, K.U.Leuven, February 1995.

[100] B. Van Veen and K.M. Buckley, "Beamforming: a versatile approach to spatial filtering," *IEEE ASSP Magazine*, pp. 4–23, April 1988.

[101] D.S. Watkins, "Understanding the QR algorithm," *SIAM Review*, vol. 24, no. 4, pp. 427–440, October 1982.

[102] D.S. Watkins, "Isospectral flows," *SIAM Review*, vol. 24, pp. 379–391, 1984.

[103] D.S. Watkins and L. Elsner, "Self-similar flows," *Linear Algebra and its Applications*, vol. 110, pp. 213–242, 1988.

[104] B. Widrow and M.A. Lehr, "30 years of adaptive neural networks: Perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, September 1990.

[105] R. Williams, "Feature discovery through error-correction learning," Tech. Rep. 8501, Institute of Cognitive Science, San Diego, CA, 1985.

[106] W.Y. Yan, U. Helmke, and J.B. Moore, "Global analysis of Oja's flow for neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 674–683, September 1994.

[107] B. Yang, "Gradient based subspace tracking algorithms and systolic implementation," *International Jounal of High Speed Electronics and Systems*, vol. 4, no. 2, 1993.

[108] B. Yang, "Projection approximation subspace tracking," *IEEE Transactions on Signal Processing*, vol. 43, no. 1, pp. 95–107, January 1995.

[109] C. Yi, *Robustness analysis and controller design for systems with structured uncertainties*. PhD thesis, K.U.Leuven, May 1995.

[110] C. Yi and B. De Moor, "Continuous time power iteration for computing a lower bound of μ," in *Proceedings of the 1994 American Control Conference, Baltimore, Maryland, June 29 - July 1, 1994*, pp. 1416–1417, 1994.

[111] C. Yi, J. Dehaene, and B. De Moor, "Computation of the structured singular value with gradient based optimization algorithms," in *Proceedings of the 1995 American Control Conference*, vol. 5, pp. 3034–3038, IEEE Service Center, Piscataway NJ, 1995.

[112] J.M. Zurada, *Introduction to Artificial Neural Systems*. West Publishing Company, 1992.