

Nonlinear Eigenvalue Problems

by

Ross Adams Lippert

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy


at the


MASSACHUSETTS INSTITUTE OF TECHNOLOGY

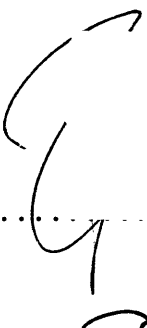
June 1998

©Ross A. Lippert, 1998.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author 
Department of Mathematics
May 1, 1998

Certified by .. 
Alan Edelman
Assistant Professor
Thesis Supervisor

Accepted by 
Hung Cheng
Chairman, Applied Mathematics Committee

Accepted by
Richard Melrose
Chairman, Departmental Committee on Graduate Students

JUN 01 1998 Science

Nonlinear Eigenvalue Problems

by

Ross Adams Lippert

Submitted to the Department of Mathematics
on May 1, 1998, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis begins by exploring the problem left open by Wilkinson of computing the distance of a given diagonalizable matrix to the nearest matrix with degenerate structure. Algorithms for finding the nearest nondiagonalizable matrix to a given matrix are presented and analyzed, and an algorithm based on the work of Alexander Malyshev is presented for finding the nearest nondiagonalizable matrix with a specific nontrivial canonical form.

This thesis will then recast the Wilkinson problem in terms on a more general nonlinear eigenvalue problem. Software written by the author to solve general nonlinear eigenvalue problems is presented, as well as a variety of examples of problems which the software can solve, including Local Density Approximation (LDA) problems arising in solid state physics.

Lastly, this thesis presents several algorithms for computing operators commonly arising in LDA problems (Laplacian, pointwise multiplication, etc.) using a truncated basis of interpolating scaling functions (the Deslauriers-Dubuc functions). The algorithms suggest a means of improving the performance of common linear operations in any truncated compactly supported wavelet basis.

Thesis Supervisor: Alan Edelman

Title: Assistant Professor

Acknowledgments

I would like to thank both Alan Edelman and Tomás Arias. Both of whom argued against me at times when I thought I should stop because I could not derive some result or other. Needless to say, they usually ended up being right. I think that this is one of the most valuable lessons about research that I have learned.

I would also like to thank Alan for discussing all aspects of this thesis with me. While pushing and prodding me to stay focused, he nonetheless gave me room to indulge my own curiosities and form my own style. I would also like to thank my academic siblings, my fellow advisees, Yanyuan Ma, Parry Husbands, and Peter McCorquodale, for all their help in my graduate existence.

There are many people who have stimulated my thoughts and provided me with a great deal of material with which to work. I'd like to thank Gil Strang for his wavelets class, Steve Smith for his valuable work on Stiefel-Grassmann optimization, Alexander Malyshev for his inspirations on general multiple eigenvalue problems, and Erik Elmroth for his insights into the double eigenvalue problem.

I would like to thank the many friends I have made at MIT whose company provided a welcome distraction from academic work, notably Elizabeth Shapere and Nancy Nangeroni, as well as sundry others who have shared drinks with me at the Phoenix Landing.

I finally would like to thank my parents for their encouragement and for providing me with the means to pursue my own interests wherever those interests took me.

Contents

1	Foreword	17
2	The Computation and Sensitivity of Double Eigenvalues	19
2.1	Introduction	19
2.2	The geometry of eigenvalues	22
2.2.1	The level sets of eig	22
2.2.2	The double eigenvalue variety	25
2.2.3	An illustration by characteristic polynomials	26
2.2.4	A pseudospectra “movie”	29
2.3	Finding the nearest double eigenvalue	33
2.3.1	A perturbative look at double eigenvalue averaging	33
2.3.2	An iterative fixed-point algorithm	36
2.3.3	The singularities of the spectral portrait	37
2.4	Nearest \hat{A} 's are basically J_2 's	41
2.5	Conditioning and stability	46
2.5.1	Sensitivity analysis of $A = \hat{A} + \sigma uv^H$	47
2.5.2	Ill-posed problems and principal curvatures	53
2.5.3	Stability of the fixed point algorithm	59
2.5.4	Sensitivity of the critical points of the spectral portrait	60
2.6	Nearest \hat{A} 's come from simple σ_{\min}	62
2.7	Triple eigenvalues	66
3	The Computation of Nearest k-fold Eigenvalues	69

3.1	Introduction	69
3.2	Malyshev's formula for $\text{dist}(A, J_2)$	70
3.3	Formula for $\text{dist}(A, J_k)$	70
3.4	Finding Δ	73
3.4.1	Generic success: $\text{tr}(U^H V) \neq 0$	76
3.4.2	Nongeneric success: $\text{tr}(U^H V) = 0$	77
3.5	The 2-norm "normals"	78
3.6	A possible rationale for the 2-norm normals	82
3.7	Concluding remarks	84
4	Solving General Nonlinear Eigenvalue Problems	85
4.1	Introduction	85
4.2	Calling <code>sg_min</code>	86
4.3	Sample problems and their differentials	90
4.3.1	The Procrustes problem	91
4.3.2	Nearest Jordan structure	92
4.3.3	Trace minimization	93
4.3.4	Trace minimization with a nonlinear term	93
4.3.5	Simultaneous Schur decomposition problem	94
4.3.6	INDSCAL	95
4.4	Running our examples	96
4.4.1	A sample Procrustes problem	97
4.4.2	A sample Jordan block problem	98
4.4.3	A sample trace minimization problem	99
4.4.4	A sample LDA toy problem (trace minimization with nonlinear term)	101
4.4.5	A sample simultaneous Schur decomposition problem	102
4.4.6	A sample INDSCAL problem (simultaneous diagonalization)	103
4.5	Making modifications (the structure of the code)	104
4.5.1	Subroutine dependencies	105
4.5.2	Under the hood	107

4.5.3	What to modify	109
4.6	Geometric technicalities	111
4.6.1	Manifolds	111
4.6.2	The difference between a tangent and a differential	113
4.6.3	Inner products, gradients, and differentials	114
4.6.4	Getting around $\text{Stief}(n, k)$	115
4.6.5	Covariant differentiation	116
4.6.6	Inverting the covariant Hessian (technical considerations)	118
5	Multiscale Computation with Interpolating Wavelets	121
5.1	Introduction	121
5.2	Introduction to interpolets	126
5.2.1	Interpolet Multiresolution Analysis	126
5.2.2	Interpolet transforms	131
5.3	Accuracy of interpolet approximation	133
5.4	Truncated bases	134
5.5	Multilevel algorithms for ∇^2 and other operators	142
5.5.1	∇^2 in 1-level decomposition	144
5.5.2	Computing other operators	149
5.5.3	∇^2 in 2-level decomposition	151
5.6	Efficient implementation	154
5.6.1	Data structures	155
5.6.2	Implementation overview	155
5.6.3	Block-to-block subroutines	156
5.7	Results and conclusions	156
A	Matlab source for Stiefel-Grassmann optimization	167

List of Figures

2-1	$\ e^{tA}\ _2$ (solid) and $\ e^{t\hat{A}}\ _2$ (dashed) vs. t for A defined by (2.1) and its nearest defective matrix	21
2-2	(Left) M_λ defined by the constraint $\mathbf{eig}(A) - \lambda = 0$ for fixed λ . The gradient of the constraint function points in the direction of increasing λ and has magnitude inversely proportional to the distance to $M_{\lambda+\delta\lambda}$. (Right) the M_λ defined by the constraint $\sigma_{\min}(A - \lambda I) = 0$ for fixed λ , giving a gradient of unit length in the direction of increasing λ , the distance to next M_λ being proportional to $s = v^H u$	24
2-3	The envelope can be identified by the critical points of the distance from A to M_λ	26
2-4	A graph in the a - b plane of the lines of polynomials of the form $p(x) = x^3 + ax + b$ satisfying $p(\lambda) = 0$	27
2-5	A graph in a - b - c space of the double root variety, or swallowtail.	28
2-6	At $\sigma = 0$ we have a single eigenvalue at -0.3 and a parabolic valley for the double eigenvalue at 0.15 . The conservation law indicates that there is also a saddle in the portrait (it sits between the valleys).	30
2-7	As we begin to move off \mathcal{D} , the double eigenvalue splits into two complex conjugates which move off from their origin very quickly. The valley of the double eigenvalue (charge $+1$) has split into two valleys ($+1$ each) with a saddle (-1) between them.	30

2-8	We then see a critical change to the left of $\lambda = 0.15$ as the saddle which initially sat between the two valleys at $\sigma = 0$ decays into two saddles and a maximum.	31
2-9	The maximum which was spun off from the decay collides with the saddle at $\lambda = 0.15$. At the focal distance $\sigma = 0.45$ the maximum and saddle have annihilated each other, creating a degenerate shoulder.	31
2-10	After the collision, the critical points have exchanged charges. The $\lambda = 0.15$ critical point is now a maximum while a saddle heads from it to the right.	32
2-11	The fixed point algorithm converging.	38
2-12	The fixed point algorithm approaching a limit-cycle.	39
2-13	A possible problem finding a normal from \hat{A} to A illustrated in characteristic polynomial space.	42
2-14	The normals of the cubic double root variety forming their parabola-like envelope. The concave down arc in the plot is another part of the envelope which occurs over cubics with complex coefficients.	47
2-15	The normals of the swallowtail. A faint hood-like shape begins to emerge.	48
2-16	The swallowtail and its hood-like evolute of ill-conditioned problems.	48
2-17	In the cubic polynomial space, a particle moves along the double root variety, with velocity v and centripetal acceleration $a(v, v)$	54
2-18	The centripetal acceleration of the particle moving along the double root variety can be associated with a circle centered at the focal point of the double root variety.	55
2-19	Below the focal point of a hemispherical surface, the distance minimizing p on the hemisphere to a given q is at the bottom of the surface.	64
2-20	Above the focal point of the hemispherical surface, the bottom point of the hemisphere is now a maximum of distance (the nearest p on the hemisphere to q has become either of the end points of the hemisphere).	64
4-1	Procrustes Problem	98
4-2	Jordan Problem	99

4-3	Trace Minimization Problem	101
4-4	LDA Toy Problem	102
4-5	Simultaneous Schur Problem	103
4-6	(INDSCAL) Simultaneous Diagonalization Problem	105
4-7	Our code tree: dependencies of various modules	106
4-8	The unconstrained differential of $F(Y)$ can be projected to the tangent space to obtain the covariant gradient, G , of F	115
4-9	In a flat space, comparing vectors at nearby points is not problematic.	117
4-10	In a curved manifold, comparing vectors at nearby points can result in vectors which do not lie in the tangent space.	117
5-1	(a) shows $\phi(x/2) = \frac{-1}{16}\phi(x-3) + \frac{9}{16}\phi(x-1) + \phi(x) + \frac{9}{16}\phi(x+1) + \frac{-1}{16}\phi(x+3)$. (b) shows the functions $\frac{-1}{16}\phi(x-3), \frac{9}{16}\phi(x-1), \phi(x), \frac{9}{16}\phi(x+1), \frac{-1}{16}\phi(x+3)$	127
5-2	(a) the smooth function $f(x) = (\frac{x}{2} + 3)^3 e^{-(\frac{x}{4})^4}$. (b) the approximation to $f(x)$ in $\mathcal{I}_0(Z)$. (c) component of the approximation in $\mathcal{I}_1(C_1)$. (d) component of the approximation in $\mathcal{I}_1(D_1)$	135
5-3	A visual summary of the 1-level touching condition. Solid plots represent functions centered at points in S . Dotted plots represent functions centered at points in $Z^n - S$. Tick marks delimit the overlap of the two functions.	137
5-4	Our example of the 1-level touching good basis in one dimension. Note that the two functions plotted do not touch. Tick marks denote the set $S \subset Z$	138
5-5	A generic example of a truncation which meets our definitions of good and 1-level touching. In black are the points of $S \subset Z^n$	139
5-6	An example of a 2-level touching good basis which can be used for a diatomic molecule (two atomic cores). The points in $S \subset Z^n$ are where residual values could be significant.	140
5-7	The previously used implementation is on the left, and the implementation employing a 1-level touching algorithm is on the right. (Note the difference in scale on the vertical axes.)	158

5-8	The previously used implementation is on the left, and the implementation employing a 2-level touching algorithm is on the right. (Note the difference in scale on the vertical axes.)	159
5-9	The previously used implementation is on the left, and the implementation employing a multilevel algorithm is on the right. (Note the difference in scale on the vertical axes.)	160
5-10	The condition number of the Laplacian operator represented in truncated multiresolution interpolet basis as a function of the number of refinement levels k with and without simple diagonal preconditioning and compared with the condition number in an orthogonal basis with the same resolution. Lines indicate results for bases with a single atomic center of refinement, and points represent results for two atomic centers corresponding to the nitrogen molecule. . . .	162
5-11	Convergence of the solution to Poisson's equation for the nuclear potential in a nitrogen molecule in an interpolet basis with $k = 8$ levels of refinement about each nucleus.	163
5-12	Effective condition number of Poisson's equation for the nuclear potential in a nitrogen molecule with simple diagonal preconditioning as a function of k , the number of levels of refinement about each nucleus.	163

List of Tables

2.1	phenomena observed along the normal	32
4.1	A short list of the optional arguments for <code>sg_min</code>	88
5.1	Notation for multiscale computations.	125

Chapter 1

Foreword

What is a *nonlinear eigenvalue problem*? For us, it has become a catch-all phrase for a flavor of eigenvalue problem that one does not currently expect to see in an LAPACK style software library or a Matrix Computations style text. One way to “nonlinearize” the usual $Ax = \lambda x$ problem is to have the elements of A depend on x and λ . Another example arises when one wishes to regularize the eigenstructure of A according to some prescribed canonical form (i.e., given A what is the nearest \hat{A} such that \hat{A} has some prescribed eigenstructure). Characteristic of such nonlinear eigenvalue problems is a variational or optimizational procedure at the forefront of the computation. Polynomial eigenproblems are sometimes referred to as nonlinear (e.g. Demmel [23] uses this nomenclature, but we are not in favor of this use of the term).

Some nonlinear eigenvalue problems have structure from which specialized algorithms for solution can be found. There is a wealth of geometric theory which can lead one to the nearest defective \hat{A} to a given A . This geometric theory is elaborated in Chapter 2. There also appears to be some geometric structure to finding the nearest \hat{A} to A with an arbitrary canonical structure, though this is a much more difficult problem than defectiveness, and the theory is still incomplete (Chapter 3).

More general nonlinear eigenvalue problems may not have as rich a geometric structure as was found in Chapters 2 or 3 but can still be formulated as optimizations of some function $F(Y)$ over the set of k -dimensional subspaces of \mathbb{R}^n or \mathbb{C}^n , much the same way that the

eigenvalue problem can be cast as an optimization of the Rayleigh quotient:

$$\min_{x^H x=1} x^H A x.$$

This leads one to consider optimizations over the Stiefel or Grassmann manifolds (or a set of manifolds in between which we call *Stiefel-Grassmann manifolds* a.k.a. flag manifolds). Fortunately, these manifolds have enough structure that their differential geometry is computable in terms of elementary functions and matrix exponentiation. Chapter 4 provides the computational details for abstracting the optimization algorithms of flat spaces to their curved analogues on Stiefel-Grassmann manifolds.

One particular Stiefel-Grassmann optimization (actually, it is a pure Grassmann optimization) is the computation of electronic ground states in the Local Density Approximation (LDA). In an LDA problem the eigenvalue problem is

$$A(Y)Y = Y\Lambda,$$

where Y is $n \times k$ and Λ is diagonal. This is an eigenvalue problem where the operator A depends on the eigenspace Y , of the sort mentioned earlier.

Work on this problem led to development of the practical aspects of wavelet computations, as the Y were being represented in a wavelet basis of interpolating scaling functions. This led to a digression from general nonlinear eigenvalue problems to the improvement of the performance of common wavelet operators. Thus Chapter 5 may appear to be somewhat out of place in this thesis, but I hope that this explanation suffices. Chapter 5 appears to have generalizations to wavelet bases beyond those of interpolating scaling functions. Whether this is possible or not remains to be seen.

Chapter 2

The Computation and Sensitivity of Double Eigenvalues

2.1 Introduction

A fundamental problem in numerical analysis is to understand how eigenstructure affects the numerical behavior of algorithms. Classical insights come from studying the conditioning of the eigenvalue problem. From this point of view, ill-conditioning is related to nearness to a defective matrix. More recent insights are based on pseudospectra where interesting behavior is associated with non-normality. In this chapter, we revisit the classical insights with the added technology of differential geometry. Pseudospectra are windows into n^2 dimensional space. Singularities in the spectral portrait are relics of a defective matrix. “Singularities” of these singularities (to be defined later) indicate a center of curvature of the defective matrices and play a role in conditioning.

If A is an $n \times n$, diagonalizable matrix, we can find a neighborhood about A in matrix space where the eigenvalues are all analytic functions of the elements of A . Thus we have n analytic functions $\lambda_j = \mathbf{eig}_j(A)$. An eigenvalue λ_j is said to be ill-conditioned if first order variations A result in very large first order variations in λ_j . A textbook example is the

12 × 12 Frank matrix

$$F_{12} = \begin{pmatrix} 12 & 11 & 10 & \dots & 3 & 2 & 1 \\ 11 & 11 & 10 & \dots & 3 & 2 & 1 \\ 0 & 10 & 10 & \dots & 3 & 2 & 1 \\ 0 & 0 & 9 & \dots & 3 & 2 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 2 & 2 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 1 \end{pmatrix}.$$

An ϵ perturbation of the upper right element results in a change in the smallest two eigenvalues of order $10^7\epsilon$ (a relative change of order 2×10^8).

Wilkinson [67, 68] showed that if the conditioning of some of the eigenvalues of a matrix is poor, then that matrix must lie near some defective matrix. He further presented several bounds on this distance. In particular, the Frobenius norm distance from F_{12} to the nearest matrix with a double eigenvalue is only 1.822×10^{-10} .

Nearness to a defective matrix can also sometimes explain transient behaviors in the matrix exponential. Such transient behavior is explored by Trefethen [63], who exhibited the transient behavior of

$$A = \begin{pmatrix} -1 & 5 \\ 0 & -2 \end{pmatrix}, \tag{2.1}$$

a matrix with distinct negative eigenvalues. His message from this example is that defective eigenvalues are neither necessary nor sufficient for transient growth. It is curious, however, that in his example, there is a defective matrix lurking. Let U be defined by the singular value decomposition $U\Sigma V^T = A + \frac{3}{2}I$. Then

$$U^T A U = \begin{pmatrix} -1.5 & 5.0495 \\ 0.049509 & -1.5 \end{pmatrix}$$

and thus A is a distance 0.049509 ($= 9 \times 10^{-3} \|A\|_F$) from a matrix \hat{A} with a double eigenvalue, -1.5 . A plot of the norms of the exponentiations of A and \hat{A} in Figure 2-1 shows that the defective matrix does play a role.

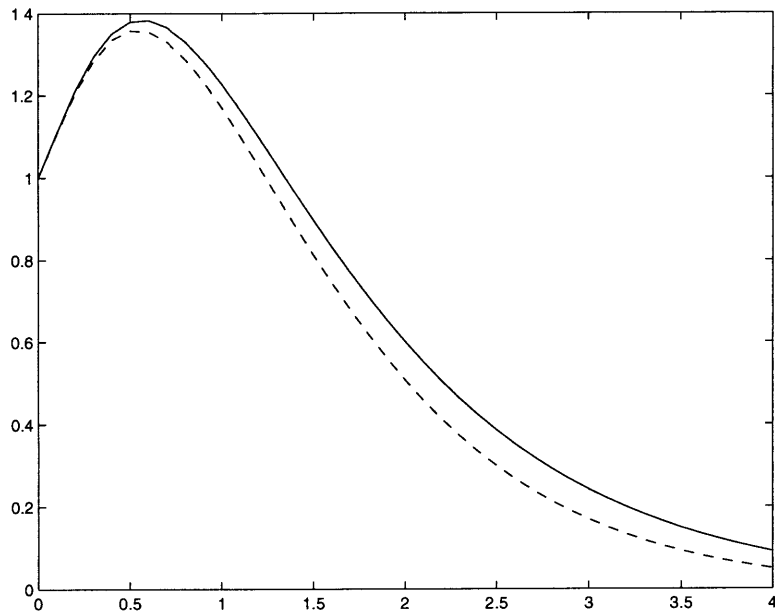


Figure 2-1: $\|e^{tA}\|_2$ (solid) and $\|e^{t\hat{A}}\|_2$ (dashed) vs. t for A defined by (2.1) and its nearest defective matrix

Given the significance of a lurking nearby defective matrix in these examples, this chapter studies the computation of the shortest (least squares) distance of any matrix, A , with distinct eigenvalues, to the nearest matrix, \hat{A} , with at least one repeated eigenvalue. Wilkinson reviewed and critiqued several bounds on this distance, although no procedure was given for this computation. Recently, Malyshev [49] has studied the 2-norm version of this problem. There has also been recent work on the computation of the distance to more general Jordan structures in [22, 33, 29, 30], and also in our `sg_min` template (see Chapter 4).

This chapter closely studies the $n^2 - 1$ dimensional surface of defective matrices and presents two methods (the `sg_min` approach is a third, though it will not be discussed in this chapter) which will identify the \hat{A} for a given A and give appropriate condition numbers on \hat{A} and its repeated eigenvalue. One method proposed will relate the nearby \hat{A} to the critical points of the spectral portrait. We shall find that the conditioning of \hat{A} is directly related to the degeneracy of the associated critical point. A second method proposed relies on successive improvements to an approximation of the repeated eigenvalue. This method will have a stability which is also dependent upon conditioning of the critical points of the spectral

portrait. Although we feel that there is more theoretical value in the first method, we present the second because of its easy implementation in terms of common Matlab operations.

Throughout this chapter, unless otherwise stated, we will use the Frobenius matrix norm and inner product. We also assume passing familiarity with concepts from differential geometry and singularity theory. (We recommend [13] as an excellent reference for singularity theory.)

2.2 The geometry of eigenvalues

In this section, we will examine the level sets of the **eig** function and their envelope, the double eigenvalue variety. We will consider a matrix A with eigenvalue λ to belong to a surface M_λ in matrix space. This is a surface of co-dimension 1 with a continuous normal vector field almost everywhere. The envelope of the M_λ , \mathcal{D} , the double eigenvalue variety is comprised of all matrices with at least one repeated eigenvalue. This surface is also of co-dimension 1 and also has a continuous normal field almost everywhere.

2.2.1 The level sets of **eig**

Consider the function **eig**(A) returning some eigenvalue of A . If A is diagonal, then **eig** may be presumed analytic in a neighborhood of A . We define $M_\lambda = \mathbf{eig}^{-1}(\lambda)$ to be the set of all matrices over either \mathbb{R} or \mathbb{C} with eigenvalue λ . M_λ is a variety which is differentiable almost everywhere. M_λ may be thought of as a level set of **eig**. All $A \in M_\lambda$ can be characterized by the constraint equation $e(\lambda, A) = 0$ for λ fixed, where $e(\lambda, A) = \mathbf{eig}(A) - \lambda$ in a neighborhood of A .

It is well known (see [39], pp. 320-324) that if $\lambda = \mathbf{eig}(A)$ is a simple eigenvalue with right eigenvector x ($Ax = \lambda x$) and left eigenvector y ($y^H A = \lambda y^H$), then variations in λ relate to the variations in A by the formula

$$\dot{\lambda} = \frac{y^H \dot{A} x}{y^H x}. \quad (2.2)$$

This formula captures the differential behavior of $\lambda(t) = \mathbf{eig}(A(t))$ at diagonalizable matrices.

More generally, if $\lambda(t)$ is independent of $A(t)$ the derivative of the constraint function, e , is

$$\frac{d}{dt}e(\lambda(t), A(t)) = \frac{y^H \dot{A} x}{y^H x} - \dot{\lambda}. \quad (2.3)$$

(Note, the condition $\lambda(t) = \mathbf{eig}(A(t))$ implies $e = \frac{d}{dt}e = 0$ reducing to Equation (2.2).)

The differential behavior of any scalar (analytic) function $z = g(w)$ can be captured by a gradient and a Riemannian (Hermitian) inner product.

$$\dot{z} = \langle \nabla g, \dot{w} \rangle.$$

Therefore, using the Frobenius inner product on matrices $\langle A, B \rangle = \text{tr}(A^H B)$, we may rewrite the perturbation formula (2.2) as

$$\dot{\lambda} = \left\langle \frac{y x^H}{x^H y}, \dot{A} \right\rangle,$$

from which we see that the gradient of \mathbf{eig} is $\nabla \mathbf{eig} = \frac{y x^H}{x^H y}$, and the differential behavior of e is

$$\frac{d}{dt}e(\lambda(t), A(t)) = \langle \nabla \mathbf{eig}, \dot{A} \rangle - \dot{\lambda}.$$

Geometrically, the gradient of any function is perpendicular to that function's level sets. Thus, $\frac{y x^H}{x^H y}$ is normal to the surface M_λ . If we select y and x such that $\|y\| = \|x\| = 1$ then $N = y x^H$ is the unit normal (unique up to sign [or phase]). We may identify $s = x^H y$ as the inverse of the length of the gradient (also up to sign [or phase]).

There is an unfortunate characteristic of the constraint function e . At points of M_λ where the eigenvalue, λ , is ill-conditioned, the right and left eigenvectors of λ are close to being perpendicular, thus $s = x^H y$ is very small and the magnitude of $\nabla \mathbf{eig}$ approaches infinity. If A were a matrix with a nontrivial Jordan block for λ then $x^H y$ would vanish and the gradient of \mathbf{eig} (as well as the differential of e) would be infinite.

To obtain a better differential behavior at A with ill-conditioned λ , we must dispense with e and use a constraint equation with a better differential behavior to characterize the M_λ surfaces. If we let $f(\lambda, A) = \sigma_{\min}(A - \lambda I)$, we may define M_λ by the constraint equation

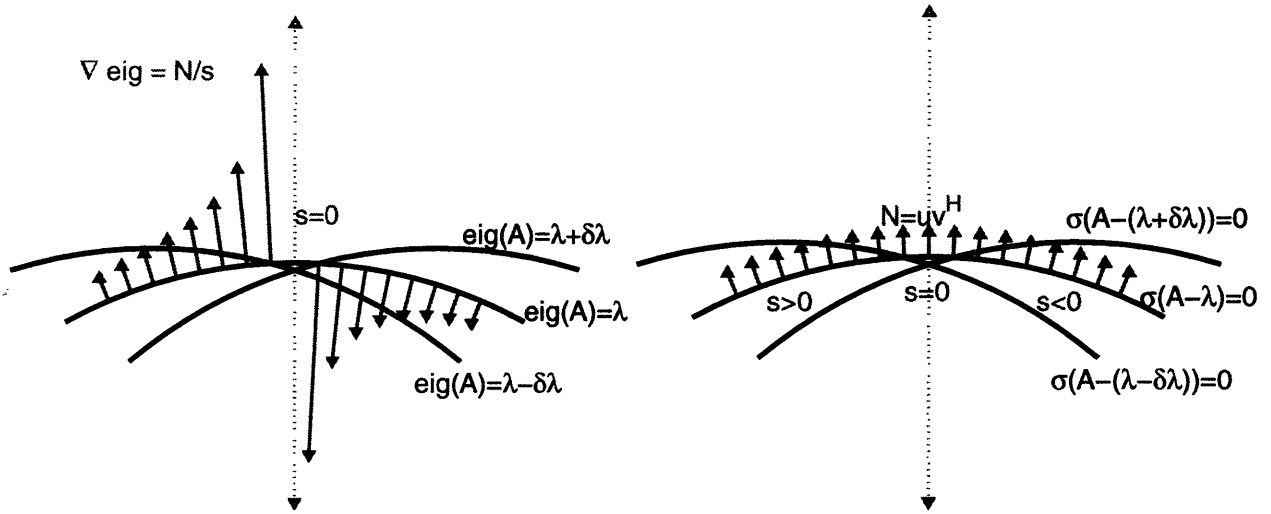


Figure 2-2: (Left) M_λ defined by the constraint $\text{eig}(A) - \lambda = 0$ for fixed λ . The gradient of the constraint function points in the direction of increasing λ and has magnitude inversely proportional to the distance to $M_{\lambda+\delta\lambda}$. (Right) the M_λ defined by the constraint $\sigma_{\min}(A - \lambda I) = 0$ for fixed λ , giving a gradient of unit length in the direction of increasing λ , the distance to next M_λ being proportional to $s = v^H u$.

$f(\lambda, A) = 0$, λ fixed. The function f has the differential behavior

$$\frac{d}{dt}f(\lambda(t), A(t)) = u^H \dot{A} v - \dot{\lambda} u^H v, \quad (2.4)$$

where u and v are the right and left singular vectors of $A - \lambda I$ (when $f(\lambda, A) = 0$ the phases of u and v are such that the right hand side of (2.4) is real and nonnegative).

Observe that u and v are unit length left and right eigenvectors (respectively) of A , and that setting $\frac{d}{dt}f = 0$ gives

$$\dot{\lambda} u^H v = \langle uv^H, \dot{A} \rangle, \quad (2.5)$$

which recovers (2.2). The gradient f with λ fixed is the unit normal to M_λ , while the cosine of the eigenvectors weights the $\dot{\lambda}$ term. Figure 2-2 summarizes the distinction between the differential behaviors of e and f .

Another way to view $f(\lambda, A)$ is as the distance from A to M_λ . Since f is the distance function to M_λ , the magnitude of its gradient (with λ fixed) must be of unit length and

perpendicular to M_λ . (One can derive 2.5 directly from this interpretation without differentiating f .)

2.2.2 The double eigenvalue variety

The double eigenvalue variety, \mathcal{D} , is the envelope of the family M_λ . Recall that the envelope of a family of surfaces S_t parameterized by t can be defined as a *tangent surface*, E , having the property that for $x \in E$, there exists a t_0 such that $x \in S_{t_0}$ and the tangent space of E at x coincides with the tangent space of S_{t_0} at x .

The characterization of the M_λ in terms of $f(\lambda, A) = \sigma_{\min}(A - \lambda I)$ allows us to analyze \mathcal{D} . It is well known ([66], pp. 9-10) that $A \in \mathcal{D}$ iff there exist λ , u , and v such that $Av = \lambda v$, $u^H A = \lambda u^H$, and $u^H v = 0$. Thus, for a differentiable curve $A(t) \in \mathcal{D}$ with repeated eigenvalue $\lambda(t)$, we have by (2.4)

$$f(\lambda(t), A(t)) = \frac{d}{dt} f(\lambda(t), A(t)) = 0 = u^H \dot{A} v = \langle uv^H, \dot{A} \rangle,$$

giving the unit normal $N = uv^H$ of \mathcal{D} at $A(t)$, confirming that \mathcal{D} is the envelope of the M_λ .

We can also recover an important property of envelopes since $f(\lambda, A)$ is the distance function from A to M_λ . It is the case for all differentiably parameterized families of surfaces, that the envelope is given by the the critical points of the distance function, i.e. $\hat{A} \in \mathcal{D}$ when $\frac{d}{dt} f(\lambda(t), A) = 0$ for some A where \hat{A} is the nearest element of M_λ to A (see Figure 2-3).

In some sense, we have arrived at a geometric reasoning for the choice of Wilkinson's "locally fastest" perturbation to a matrix A near \mathcal{D} which would move A to \mathcal{D} nearly optimally. If one suspects that two eigenvalues, λ_1, λ_2 , of A can be coalesced with a small perturbation E , Wilkinson suggested using a perturbation of the form $E = \sigma(y_1 + y_2)(x_1 + x_2)^H$ where the y_i and the x_i are the left and right eigenvectors of λ_i and σ is adjusted so that $A + E \in \mathcal{D}$ (Section 9 of [68]).

From the preceding results, we see that if $\hat{A} \in \mathcal{D}$ then the normal line $\hat{A} + \sigma uv^H$ (parameterized by σ) passes through \mathcal{D} along a perpendicular, thus the perturbation $E = \sigma uv^H$ can be thought of as the perturbation which maximizes the change in the distance to \mathcal{D} for small σ and is then the "locally fastest" perturbation to a nearby diagonalizable matrix A .

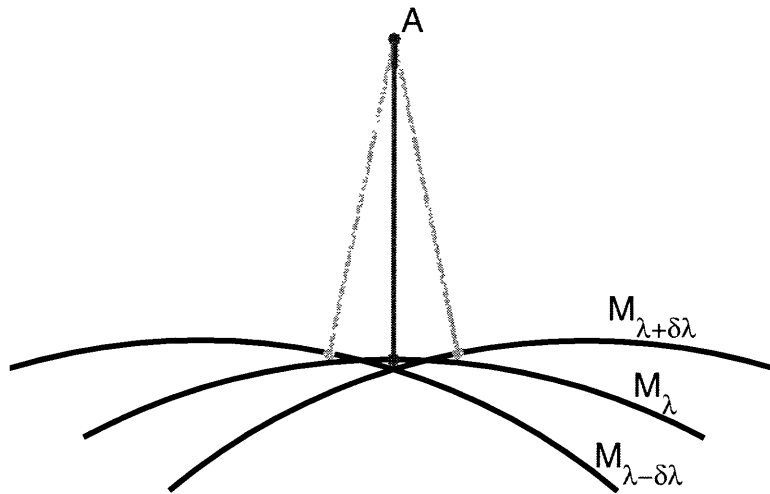


Figure 2-3: The envelope can be identified by the critical points of the distance from A to M_{λ} .

If one is close to \mathcal{D} then one expects $(x_1 + x_2)/2$ will be close to v and $(y_1 + y_2)/2$ will be close to u . The “locally fastest” perturbation is then expected to be close to the actual normal of the variety at the actual minimizing point. We will elaborate on this heuristic in Section 2.3.1.

2.2.3 An illustration by characteristic polynomials

For purposes of “picturing” \mathcal{D} and M_{λ} geometry, we consider monic n^{th} degree polynomials, which may be thought of as projections of the $n \times n$ matrices via the characteristic polynomial. Consider the cubic polynomials of the form $p(x) = x^3 + ax + b$ (corresponding to the traceless 3×3 matrices). Polynomials with a fixed root λ can be represented by a linear relation between a and b of the form

$$\lambda^3 + a\lambda + b = 0.$$

A collection of these fixed root lines is shown in Figure 2-4.

The envelope of those lines is a cusp. By varying the constraint equation

$$3\lambda^2\dot{\lambda} + a\dot{\lambda} + \dot{a}\lambda + b = 0,$$

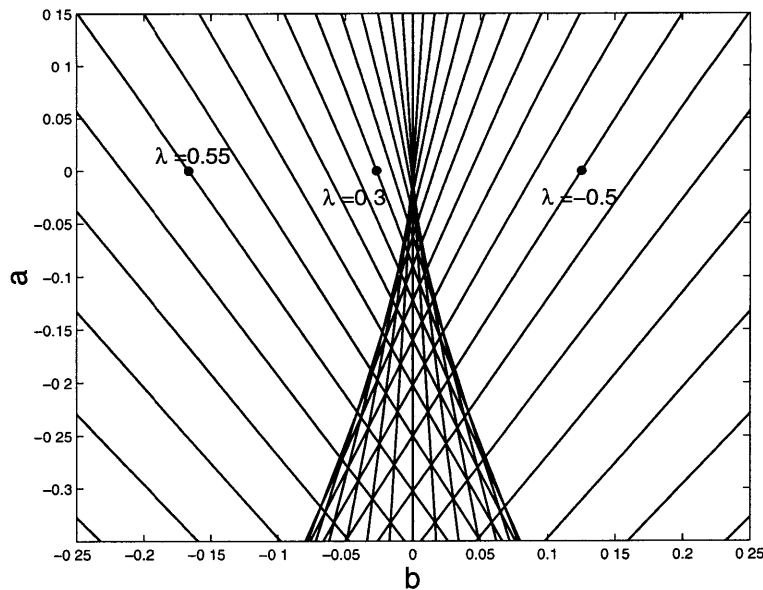


Figure 2-4: A graph in the a - b plane of the lines of polynomials of the form $p(x) = x^3 + ax + b$ satisfying $p(\lambda) = 0$.

we obtain the gradient of λ as a function of a and b ,

$$\dot{\lambda} = \frac{-(\lambda, 1) \cdot (\dot{a}, \dot{b})}{3\lambda^2 + a}.$$

We may define a unit normal $N = \frac{(\lambda, 1)}{\sqrt{\lambda^2 + 1}}$ and set the inverse magnitude of the gradient $s = \frac{3\lambda^2 + a}{\sqrt{\lambda^2 + 1}}$.

The envelope of the lines of this single root family of polynomials is given by $s = 0$, or $a = -3\lambda^2$ (and thus $b = 2\lambda^3$ by the constraint). This gives a curve of polynomials with double root λ , seen as the cusp-shaped envelope in Figure 2-4. Observe that the cusp point is the polynomial $p(x) = x^3$, the triple root polynomial.

More complicated shapes arise for quartic polynomials of the form $x^4 + ax^2 + bx + c$. In this case, the double root variety is a collection of multiple cusps, the swallowtail.

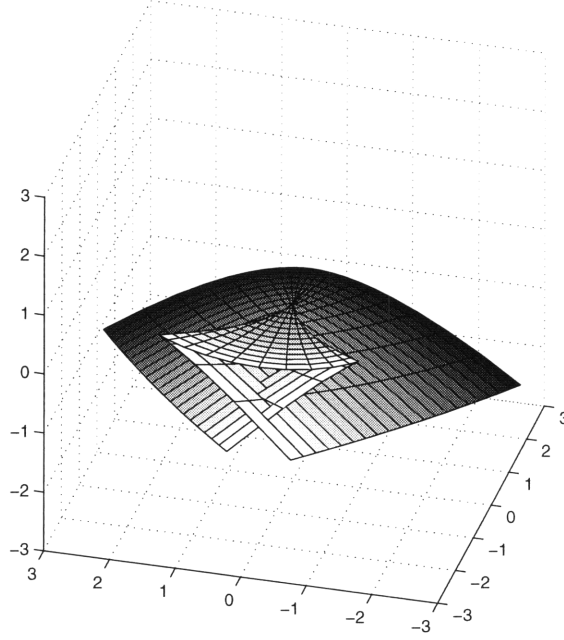


Figure 2-5: A graph in a - b - c space of the double root variety, or swallowtail.

In matrix space, the cusp may be formed from the 1-dimensional family of matrices

$$M(\lambda) = \begin{pmatrix} \lambda & \sqrt{1-2\lambda^2} & \lambda \\ 0 & -2\lambda & \sqrt{1-2\lambda^2} \\ 0 & 0 & \lambda \end{pmatrix}.$$

One then has $\det(xI - M(\lambda)) = x^3 - 3\lambda^2x + 2\lambda^3$, and it is clear that $uv^H = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$.

One can verify that $\det(xI - M(\lambda) - \sigma uv^H) - \det(xI - M(\lambda)) = \sigma(\lambda x + 1)$, parallel to the normals of the double root polynomials. For the quartic case, we may use the two parameter surface

$$M(\lambda, \mu) = \begin{pmatrix} \lambda & 1 & -\lambda^3 + \frac{\lambda}{2} & \lambda^2 \\ 0 & -\lambda - \sqrt{\mu} & 1 + \lambda^4 - \lambda^2(1 - \mu) & -\lambda^3 + \frac{\lambda}{2} \\ 0 & 0 & -\lambda + \sqrt{\mu} & 1 \\ 0 & 0 & 0 & \lambda \end{pmatrix},$$

for which $\det(xI - M(\lambda, \mu)) = (x^2 - \lambda^2)^2 - \mu(x - \lambda)^2$.

These matrix families embed the cusp and the swallowtail in matrix space as subsurfaces in (respectively) 9 and 16 dimensional spaces.

2.2.4 A pseudospectra “movie”

Since we wish to find the nearest matrix $\hat{A} \in \mathcal{D}$ to a matrix A , it is useful to study the normal line in matrix space, $\hat{A} + \sigma uv^H$ (parameterized by σ). We proceed by taking a journey along a normal to \mathcal{D} and plotting the pseudospectra as we go. The spectral portrait will be our 2 dimensional window onto n^2 dimensional matrix space.

We have seen in the previous sections that along this line, the double eigenvalue separates into two distinct eigenvalues which move apart initially with great speed. Additionally, they will leave behind a stationary point whose location and character will remain invariant as we go up the normal line for some finite length.

We start out on the cubic polynomial example matrix

$$M(\lambda) = \begin{pmatrix} \lambda & \sqrt{1 - 2\lambda^2} & \lambda \\ 0 & -2\lambda & \sqrt{1 - 2\lambda^2} \\ 0 & 0 & \lambda \end{pmatrix}.$$

with $\lambda = 0.15$. The series of spectral portraits presented (Figures 2-6 through 2-10) are those of $M - \sigma uv^H$ for various interesting σ .

The reader studying these plots may wish to recall a certain “conservation law” which comes from planar bifurcation theory (see [59, 16]). The “law” is that if all the critical points of a surface are nondegenerate, they may be assigned charges, all extrema get a +1 charge and all saddles get a -1 charge, which will be locally conserved through all continuous deformations of the surface. Additionally, spectral portraits are continuous, and their contours approach concentric circles about the origin for large $|\lambda|$. Thus any spectral portrait can be continuously deformed into a bowl and therefore has a total charge of +1.

The contour lines in Figures 2-6 through 2-10 are not uniform, but have been chosen to be dense about the current value of σ and to highlight other important qualitative features.

We can summarize the sights we have seen on this journey in Table 2.1. The reader

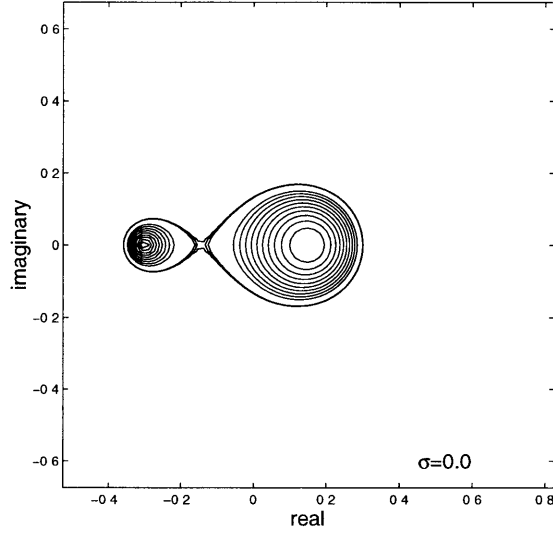


Figure 2-6: At $\sigma = 0$ we have a single eigenvalue at -0.3 and a parabolic valley for the double eigenvalue at 0.15 . The conservation law indicates that there is also a saddle in the portrait (it sits between the valleys).

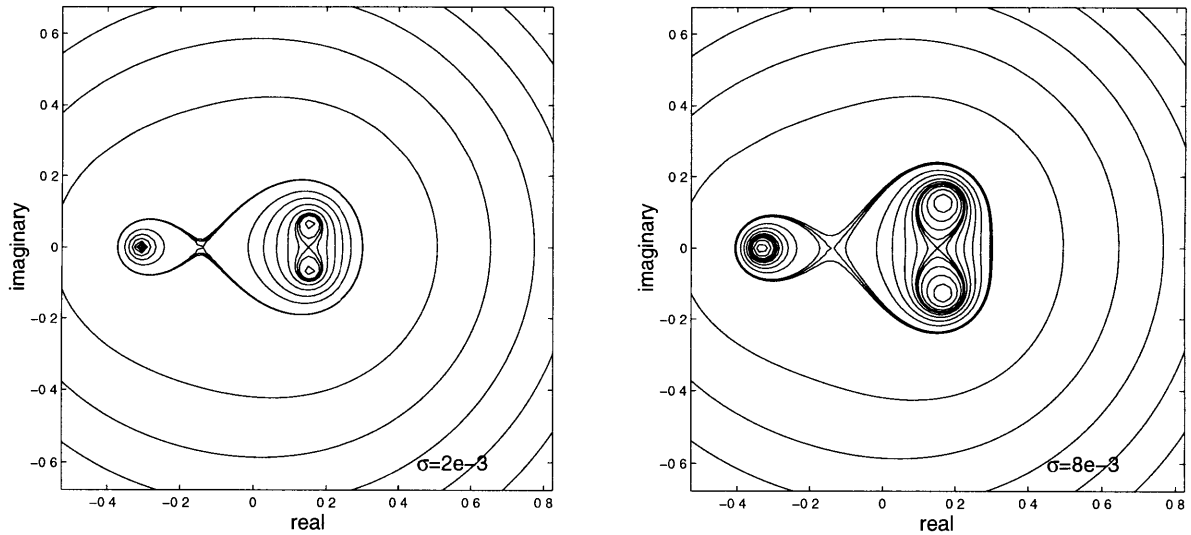


Figure 2-7: As we begin to move off \mathcal{D} , the double eigenvalue splits into two complex conjugates which move off from their origin very quickly. The valley of the double eigenvalue (charge $+1$) has split into two valleys ($+1$ each) with a saddle (-1) between them.

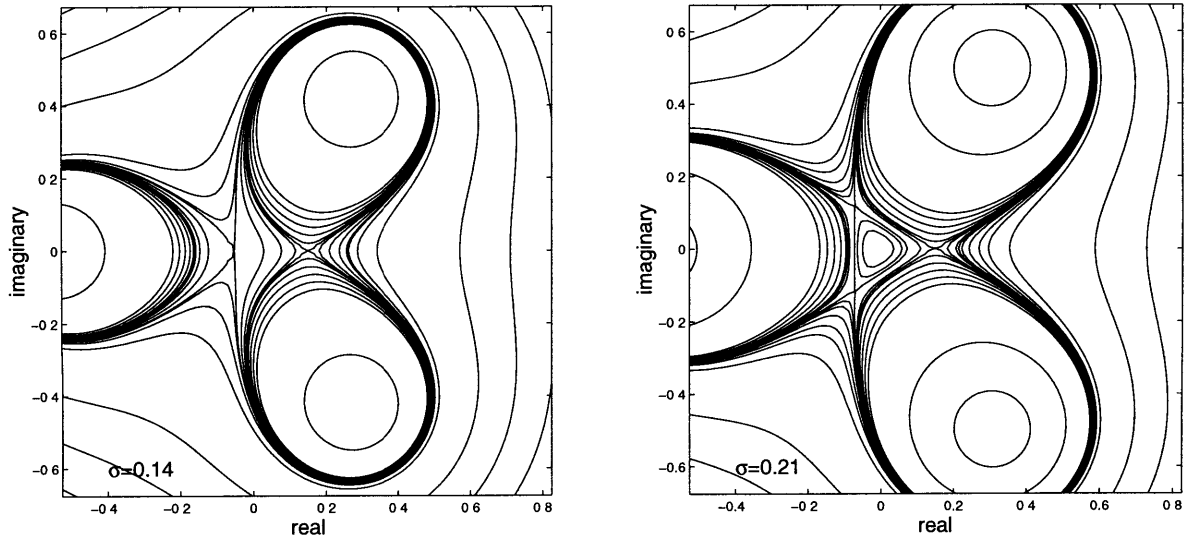


Figure 2-8: We then see a critical change to the left of $\lambda = 0.15$ as the saddle which initially sat between the two valleys at $\sigma = 0$ decays into two saddles and a maximum.

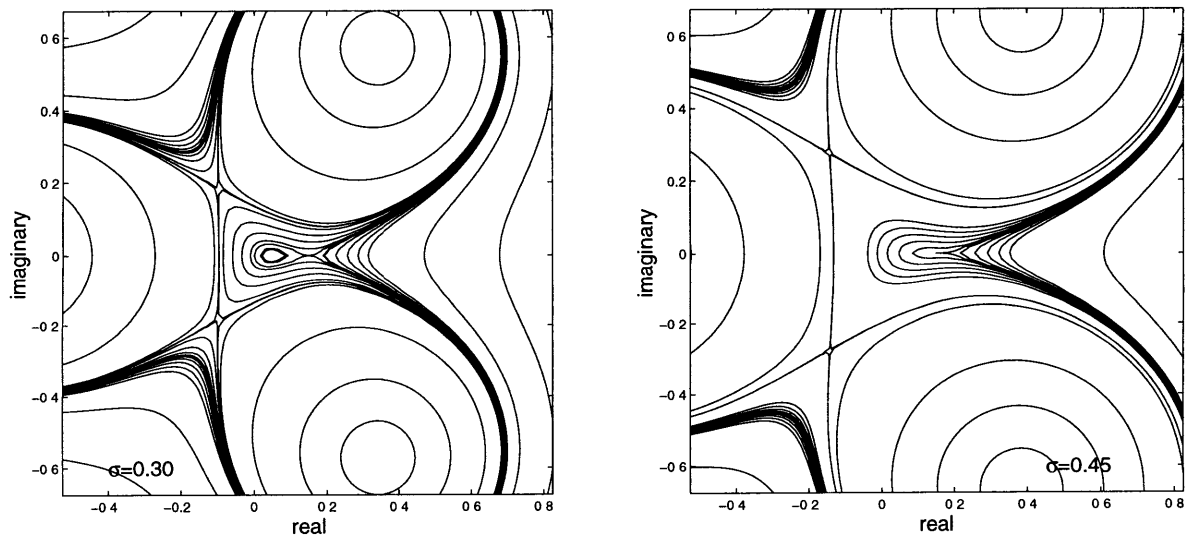


Figure 2-9: The maximum which was spun off from the decay collides with the saddle at $\lambda = 0.15$. At the focal distance $\sigma = 0.45$ the maximum and saddle have annihilated each other, creating a degenerate shoulder.

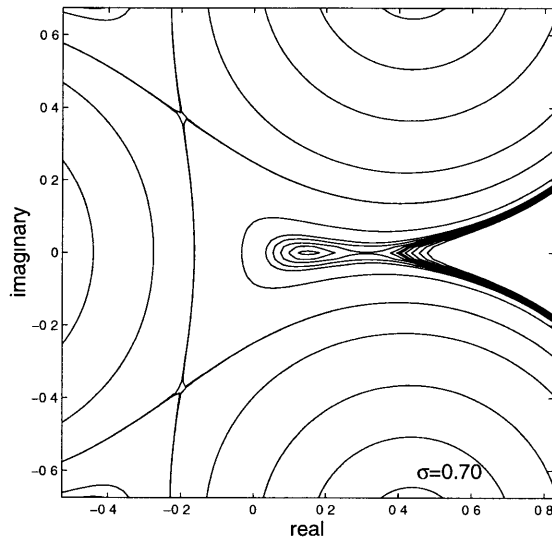


Figure 2-10: After the collision, the critical points have exchanged charges. The $\lambda = 0.15$ critical point is now a maximum while a saddle heads from it to the right.

σ	observation
$\sigma = 0.0$	double eigenvalue
$0.0 < \sigma < 0.45$	double eigenvalue splits leaving behind a saddle
$\sigma = 0.45$	the saddle becomes degenerate
$\sigma > 0.45$	the saddle has become a local maximum

Table 2.1: phenomena observed along the normal

may wish to review this section as he/she reads subsequent sections to understand the mathematical significance of the observations listed here and to illustrate the mathematical phenomena described in subsequent sections.

2.3 Finding the nearest double eigenvalue

One way to find the nearest $\hat{A} \in \mathcal{D}$ to a given A is by directly finding the repeated eigenvalue λ of \hat{A} . If the repeated eigenvalue λ were known then one can take \hat{A} to be the nearest matrix to A that has eigenvalue λ (using the singular value decomposition to find \hat{A}).

This section explores methods of calculating that λ . We will begin by exploring the common heuristic of averaging two nearby eigenvalues of A . A fixed point algorithm based on an improvement of this idea will be presented. We then show that λ can also be determined by an examination of pseudospectra of A , namely via the critical points of the spectral portrait.

2.3.1 A perturbative look at double eigenvalue averaging

A small perturbation in the normal direction of a generic point $\hat{A} \in \mathcal{D}$ results in a diagonalizable matrix $A = \hat{A} + \sigma uv^H$. The J_2 block that is part of the structure of \hat{A} bifurcates into two J_1 blocks. In this section, we will examine the heuristic of approximating the double eigenvalue of \hat{A} by the average of the two nearest eigenvalues of A by examining the Puiseux series (see [43]) in σ .

Puiseux series are nothing new in the description of the bifurcation of multiple eigenvalues, however, the additional contribution that we add is the derivation of the series along the geometric normal of \mathcal{D} . We will also derive the second order behavior of the series along the normal which can be used to measure the error of the averaging heuristic.

We quantify the non-surprising fact that the averaging heuristic tends to break sooner (i.e. for smaller σ) when \hat{A} is close to a nongeneric point of \mathcal{D} , that is, is close to being more defective than J_2 or derogatory. Later sections will show that one cannot expect to find a well-conditioned double eigenvalue if the nearby \hat{A} is close to being nongeneric, and thus the

averaging heuristic must suffer from difficulties which will also plague the more sophisticated algorithms suggested in this chapter.

We adopt the convention, in this section and throughout this chapter (with the exception of Section 2.5.2), of using v and u for the right and left eigenvectors respectively, for $\hat{A} \in \mathcal{D}$ and of writing the singular value decomposition of \hat{A} as

$$\hat{A} = \lambda I + \tilde{U} \tilde{\Sigma} \tilde{V}^H,$$

where $\tilde{U} = (u_1 \ u_2 \ \dots \ u_{n-1})$, $\tilde{V} = (v_1 \ v_2 \ \dots \ v_{n-1})$, and $\tilde{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_{n-1})$. Additionally, we set $r = \tilde{U}^H v$ and $l = \tilde{V}^H u$, noting that $v = \tilde{U} r$ and $u = \tilde{V} l$ since $u^H v = 0$ and $(\tilde{U} \ u)$ and $(\tilde{V} \ v)$ are unitary matrices.

We now consider a Puiseux series expansion of the eigenvalues and right eigenvectors of $A(\sigma) = \hat{A} + \sigma uv^H$ given by

$$\lambda + \lambda(\sigma) = \lambda + \omega \sigma^{\frac{1}{2}} \lambda_1 + \omega^2 \sigma \lambda_2 + \omega^3 \sigma^{\frac{3}{2}} \lambda_3 + \dots$$

and

$$v + v(\sigma) = v + \omega \sigma^{\frac{1}{2}} v_1 + \omega^2 \sigma v_2 + \omega^3 \sigma^{\frac{3}{2}} v_3 + \dots$$

satisfying

$$(A(\sigma) - (\lambda + \lambda(\sigma))I)(v + v(\sigma)) = 0, \tag{2.6}$$

where $\omega^2 = 1$ and $v^H v_k = 0$. Since $v^H v_k = 0$ we will write $v_k = \tilde{V} r_k$ for some $n-1 \times 1$ vector r_k and let $r(\sigma) = \tilde{V}^H v(\sigma)$. The series for the average of the two eigenvalues is given by

$$\lambda + \frac{1}{2}(\lambda(\sigma)|_{\omega=1} + \lambda(\sigma)|_{\omega=-1}) = \lambda + \sigma \lambda_2 + \sigma^2 \lambda_4 + \dots$$

Thus, averaging will be a good approximation to λ as long as the λ_2 term is relatively small.

Substituting $A(\sigma) - \lambda = \sigma uv^H + \tilde{U} \tilde{\Sigma} \tilde{V}^H$ into (2.6) we have

$$(\tilde{U} \tilde{\Sigma} \tilde{V}^H + \sigma uv^H - \lambda(\sigma))(v + v(\sigma)) = 0,$$

which can be simplified to

$$(\tilde{U}\tilde{\Sigma}\tilde{V}^H - \lambda(\sigma))v(\sigma) - \lambda(\sigma)v + \sigma u = 0. \quad (2.7)$$

We split (2.7) into two sets of equations, first by multiplying on the left by u^H ,

$$\lambda(\sigma)\sigma = u^H v(\sigma), \quad (2.8)$$

and then by multiplying on the right by \tilde{U}^H ,

$$\tilde{\Sigma}r(\sigma) - \lambda(\sigma)\tilde{U}^H\tilde{V}r(\sigma) = \lambda(\sigma)r. \quad (2.9)$$

We may invert Equation (2.9) to obtain

$$r(\sigma) = \lambda(\sigma)(\tilde{\Sigma} - \lambda(\sigma)\tilde{U}^H\tilde{V})^{-1}r.$$

To find $\lambda(\sigma)$ we substitute into (2.8) to get

$$\sigma = \lambda(\sigma)u^H v(\sigma) = \lambda(\sigma)l^H r(\sigma) = \lambda^2(\sigma)l^H(\tilde{\Sigma} - \lambda(\sigma)\tilde{U}^H\tilde{V})^{-1}r,$$

or

$$\sigma = \sum_{k \geq 1} u^H(\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^H)^k v \lambda^{k+1}(\sigma) = \sum_{k \geq 1} c_k \lambda^{k+1}(\sigma), \quad (2.10)$$

which must be inverted to obtain the series for $\lambda(\sigma)$ (the reader may note that $\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^H = (\hat{A} - \lambda I)^\dagger$, the Moore-Penrose inverse of $\hat{A} - \lambda I$, and that $c_1 = u(\hat{A} - \lambda I)^\dagger v = u^H w$ where w is the right generalized eigenvector of \hat{A}).

We seek only the first two coefficients of $\lambda(\sigma)$. They are given by

$$\lambda_1 = \frac{1}{\sqrt{u^H w}} \quad (2.11)$$

$$\lambda_2 = \frac{u^H(\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}^H)^2 v}{2(u^H w)^2}. \quad (2.12)$$

From (2.11) and (2.12) we see that the second order term in the expansion becomes significant when

$$|\sigma| \sim 4 \left| \frac{(u^H w)^3}{(u^H (\tilde{V} \tilde{\Sigma}^{-1} \tilde{U})^2 v)^2} \right|.$$

Thus we see that the averaging heuristic can be expected to work poorly if $u^H w (= l^H \tilde{\Sigma}^{-1} r)$ is small or if $\|\tilde{V} \tilde{\Sigma}^{-1} \tilde{U}^H\|_2 = \|(\hat{A} - \lambda I)^\dagger\|_2 = \frac{1}{\sigma_{n-1}}$ is large. There is an algebraic interpretation for both of these cases. If $u^H w = 0$ then it can be shown that \hat{A} must have at least $J_3(\lambda)$ defectiveness. If $\|(\hat{A} - \lambda I)^\dagger\|_2 = \infty$, then $\hat{A} - \lambda I$ must have two vanishing singular values, and thus has a derogatory Jordan structure in λ .

In summary we see that the averaging heuristic will produce close approximations to λ for nearby \hat{A} so long as the nearby \hat{A} are not close to any less generic structure than a $J_2(\lambda)$ block. For all of the algorithms presented in this chapter to calculate nearby double eigenvalues, analogous conditions will be found which govern their stability and/or accuracy.

2.3.2 An iterative fixed-point algorithm

Edelman and Elmroth [28] introduced an algorithm to find the nearest $\hat{A} \in \mathcal{D}$ to a given matrix A . The algorithm is the simultaneous solution of “dual” problems which can be stated as follows:

Problem 1:

Given a matrix A and an approximation to the perturbation bringing A to \mathcal{D} , find the repeated eigenvalue of \hat{A} .

Problem 2:

Given a matrix A and an approximation of the repeated eigenvalue of \hat{A} , find the smallest perturbation bringing A to \mathcal{D} .

For Problem 1, they use a clustering algorithm on the eigenvalues of $A - \sigma uv^H$, where σuv^H is the approximation to the normal bringing A to \mathcal{D} (σ a scalar and uv^H an approximate normal). For Problem 2, they approximate u and v with the left and right singular vectors of $\sigma_{\min}(A - \lambda I)$ for the given approximate repeated eigenvalue λ .

This resulted in an extremely compact Matlab routine, which reads roughly as

```

lam = guess;
while (1)
    [u,s,v] = svd(A-lam*I);
    Ahat = A - u(:,n)*s(n,n)*v(:,n)';
    e = eig(Ahat);
    lam = cluster_routine(e);
end

```

where the cluster routine takes the mean of two nearest elements of \mathbf{e} .

The resulting algorithm converges linearly much of the time, but it will sometimes go into cycles which are empirically not convergent. In Section 2.5.3, we present a detailed analysis of the convergence of the fixed point algorithm.

The Figures 2-11 and 2-12 show the fixed point algorithm both succeeding and failing for two different random matrices. We have plotted the new value of the variable \mathbf{e} against its value in the previous iteration.

2.3.3 The singularities of the spectral portrait

We have identified that the normal space of a matrix $\hat{A} \in \mathcal{D}$ is given by the line $\hat{A} + \sigma uv^H$, parameterized by σ , where u, v are left and right eigenvectors of \hat{A} . Thus if A lies in the normal space of its nearest matrix $\hat{A} \in \mathcal{D}$, then $A = \hat{A} + \sigma uv^H$. There is, however, no obvious way to compute u and v without knowing \hat{A} in advance. In this section, we present a method for this based on the spectral portrait of A .

Let $\sigma(x, y) = \sigma_{\min}(A - (x + iy)I)$. In the case where $\sigma(x, y)$ is smooth, we have the following theorem about the derivatives of $\sigma(x, y)$ due to Sun (see [60]).

Theorem 2.3.1 *Let $U(x, y)\Sigma(x, y)V^H(x, y) = A - (x + iy)I$, where $\sigma_n(x_0, y_0)$ is simple, and let $\tilde{U} = (u_1 \ u_2 \ \dots \ u_{n-1})$, $\tilde{V} = (v_1 \ v_2 \ \dots \ v_{n-1})$, and $\tilde{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_{n-1})$. We have then*

$$\frac{\partial \sigma}{\partial x}(x_0, y_0) = -\text{Real}\{u_n^H v_n\},$$

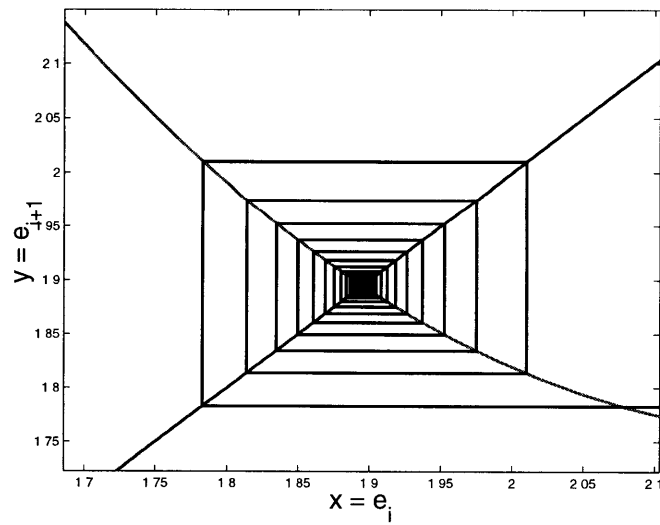
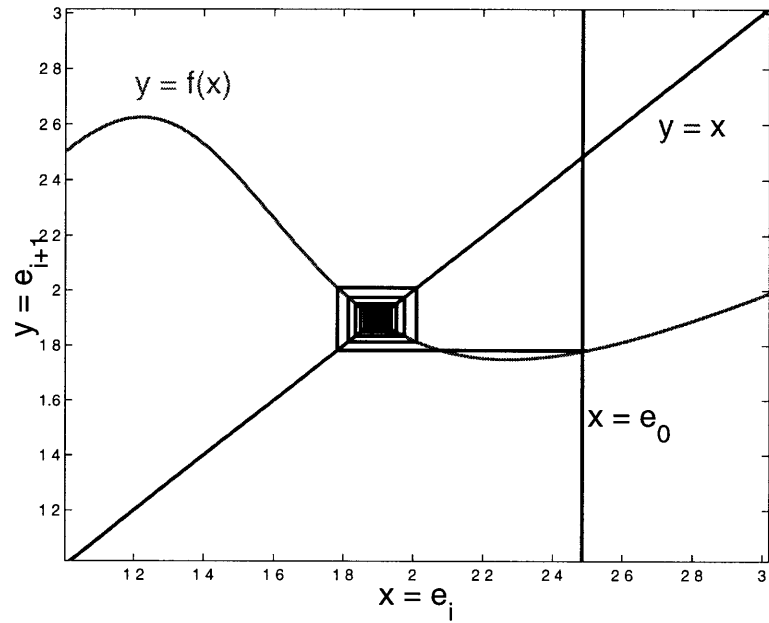


Figure 2-11: The fixed point algorithm converging.

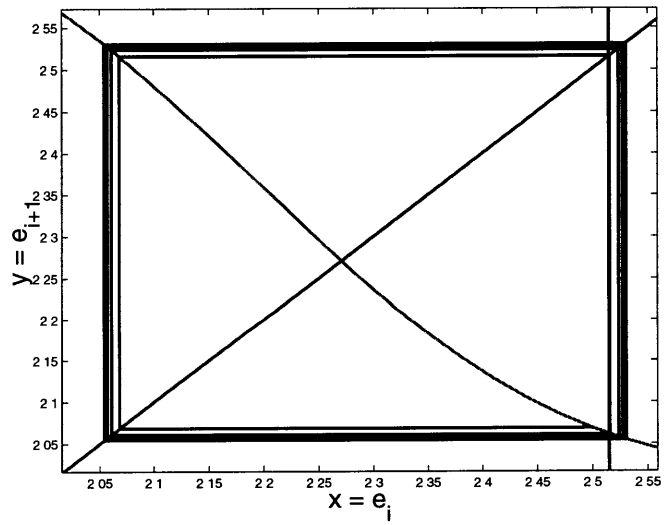
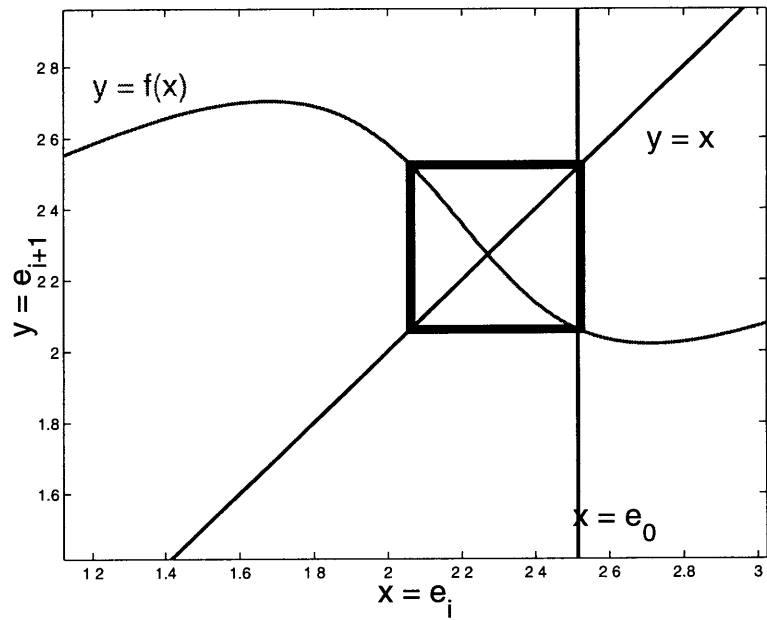


Figure 2-12: The fixed point algorithm approaching a limit-cycle.

$$\frac{\partial \sigma}{\partial y}(x_0, y_0) = \text{Imag}\{u_n^H v_n\},$$

$$\begin{aligned} \frac{\partial^2 \sigma}{\partial x^2}(x_0, y_0) &= \text{Real}\{r^H \Phi r + l^H \Phi l + 2l^H \Psi r\} + \text{Imag}\{u_n^H v_n\}^2 / \sigma_n, \\ \frac{\partial^2 \sigma}{\partial x \partial y}(x_0, y_0) &= -\text{Imag}\{2l^H \Psi r\} + \text{Imag}\{u_n^H v_n\} \text{Real}\{u_n^H v_n\} / \sigma_n, \\ \frac{\partial^2 \sigma}{\partial y^2}(x_0, y_0) &= \text{Real}\{r^H \Phi r + l^H \Phi l - 2l^H \Psi r\} + \text{Real}\{u_n^H v_n\}^2 / \sigma_n, \end{aligned}$$

where $\Phi = \sigma_n(\sigma_n^2 I - \tilde{\Sigma}^2)^{-1}$, $\Psi = \tilde{\Sigma}(\sigma_n^2 I - \tilde{\Sigma}^2)^{-1}$, $r = \tilde{U}^H v_n$, and $l = \tilde{V}^H u_n$.

Sun's theorem implies that $\sigma_x = \sigma_y = 0$ iff $u^H v = 0$. In this case, the matrix $\hat{A} = A - \sigma(x_0, y_0)uv^H \in M_{x_0+iy_0}$ is a point on the envelope of the family of M_λ , i.e. $\hat{A} \in \mathcal{D}$. Thus we have

Corollary 2.3.2 *If $\sigma(x, y)$ is stationary for some $\lambda_0 = x_0 + iy_0$ then A lies on the normal line of the matrix $\hat{A} \in \mathcal{D}$ with repeated eigenvalue λ_0 and left and right eigenvectors u and v where $\hat{A} = A - \sigma(x_0, y_0)uv^H$ and u and v are the left and right singular vectors of $A - \lambda_0 I$ with singular value $\sigma(x_0, y_0)$.*

Since the stationary points of $\sigma(x, y)$ correspond to normal lines off of \mathcal{D} . An examination of the spectral portrait of A can be used to determine the distance from A to $\hat{A} \in \mathcal{D}$.

This would lead to an algorithm which is based on finding the repeated eigenvalue of \hat{A} first, i.e.

```
(x0,y0) = find lowest critical points of the spectral portrait;
lam = x0 + iy0;
[u,s,v] = svd(A-lam*I);
Ahat = A - u(:,n)*s(n,n)*v(:,n)';
```

where one finds the critical point of lowest height by Newton's method or some other optimization routine over the complex plane.

Another result of Sun's formulae is that at stationary points

$$\det \begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix} = |r^H \Phi r + l^H \Phi l|^2 - |2l^H \Psi r|^2.$$

This formula will be useful in the later section on conditioning and stability (Section 2.5).

2.4 Nearest \hat{A} 's are basically J_2 's

We have already seen that generically speaking, the matrix $A - \hat{A}$ is a rank one matrix and \hat{A} has a J_2 structure. Of course, less generic Jordan structures may arise in special cases, but in this section we will show that even when \hat{A} has a less generic Jordan structure, $A - \hat{A}$ still has the form of a generic J_2 normal.

In Section 2.3.3, we showed that, with perhaps few generic conditions on A , the nearest matrix, $\hat{A} \in \mathcal{D}$, to A is always given by $A - \sigma uv^H$ where $\sigma = \sigma_{\min}(A - \lambda I)$ is stationary in λ , u and v are the associated singular vectors, and λ is the repeated eigenvalue of \hat{A} for which u and v are right and left eigenvectors.

If one were to be cavalier, one might assume generically that the nearest element of \mathcal{D} was always some defective matrix with a J_2 structure. If this were the case, then \mathcal{D} can be assumed to be locally differentiable about \hat{A} . Consequently, there would be a unique normal perturbation that would be of the form uv^H where $u^H v = 0$. Thus, the \hat{A} would correspond to a critical point of the spectral portrait of $A - \lambda I$ since the spectral portrait is a graph of $\sigma_{\min}(A - \lambda I)$.

However, one might consider the ways in which this generic assumptions could be violated. If \hat{A} were derogatory or had degeneracy less generic than J_2 , then there could be multiple normal directions off of \hat{A} , and $A - \hat{A}$ would not be of the assumed rank 1 form. One particular example of where one might really be concerned is in the case where \hat{A} actually has a J_3 structure and A might sit above all of the rank 1 normals of \hat{A} as shown in Figure 2-13 (since a matrix with J_3 structure is a cusp point of \mathcal{D} , there is no reason to believe that it has a continuous local normal field).

Clearly, one needs to have some lemmata to understand the conditions under which such

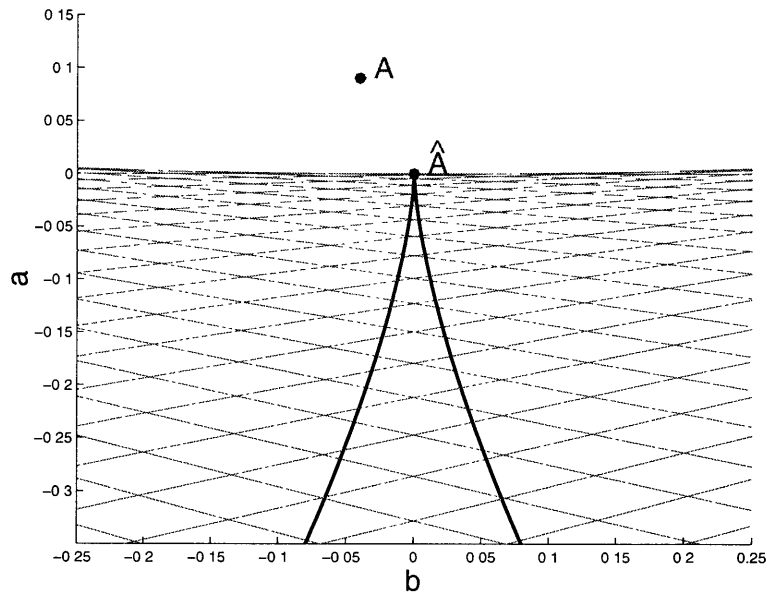


Figure 2-13: A possible problem finding a normal from \hat{A} to A illustrated in characteristic polynomial space.

an event will or will not happen in order to make generalizations which are everywhere valid. The purpose of this section is to clarify these possibilities. We present in this section a series of lemmatae which will progressively refine the possible form of $N = A - \hat{A}$ until we find that it has the form of a normal to a generic J_2 . The reader who desires to skip the technicalities of this chapter is advised to skip this section.

It will be useful to take a partial canonical decomposition of \hat{A} in the repeated eigenvalue. That is, let

$$\hat{A} = (X \quad \tilde{X}) \begin{pmatrix} J(\lambda) & 0 \\ 0 & M \end{pmatrix} (Y \quad \tilde{Y})^H,$$

where $(Y \quad \tilde{Y})^H (X \quad \tilde{X}) = I$ and $J(\lambda)$ is the canonical matrix of the Jordan structure of \hat{A} at λ .

Lemma 2.4.1 *Let \hat{A} be the nearest matrix in \mathcal{D} to A . Then*

$$N = A - \hat{A} = YW X^H,$$

for some W .

Proof: We also decompose $N = A - \hat{A}$ according to the following (slightly twisted) decomposition,

$$N = (Y \quad \tilde{X}) \begin{pmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{pmatrix} (X \quad \tilde{Y})^H,$$

one may check that $(X \quad \tilde{Y})^H (Y \quad \tilde{X}) = I$, and that the displacements $YN_{11}X^H$, $\tilde{X}N_{12}X^H$, $YN_{21}\tilde{Y}^H$, and $\tilde{X}N_{22}\tilde{Y}^H$ are mutually orthogonal.

Since $\hat{A} + \tilde{X}N_{22}\tilde{Y}^H$ has the same canonical structure in λ as \hat{A} , it is clear that $N_{22} = 0$ for \hat{A} to be the minimizer. Similarly, by observing that $(\hat{A} + YN_{21}\tilde{Y}^H)X = XJ$ and $Y^H(\hat{A} + \tilde{X}N_{12}X^H) = JY^H$, we see that N_{12} and N_{21} must vanish. \square

For the next lemma, we consider the case where J , the Jordan structure of \hat{A} at λ , is derogatory and thus of the form $J = \begin{pmatrix} J^{(1)} & 0 \\ 0 & J^{(2)} \end{pmatrix}$ and the eigenvectors are correspondingly partitioned into $Y = (Y_1 \quad Y_2)$ and $X = (X_1 \quad X_2)$. Decompose N as

$$N = (Y_1 \quad Y_2) \begin{pmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{pmatrix} (X_1 \quad X_2)^H.$$

Lemma 2.4.2 *Suppose that the Jordan structure of \hat{A} is derogatory. Then N is of the form,*

$$N = Y_1 N_{11} X_1^H + Y_2 N_{22} X_2^H.$$

Proof: Observe that $\hat{A} + X_2 W Y_1^H$ has the Jordan structure of $\begin{pmatrix} J^{(1)} & W \\ 0 & J^{(2)} \end{pmatrix}$ which, while being possibly different from J , still has a repeated eigenvalue λ .

$\hat{A} + X_2 W Y_1^H$ is closer to A than \hat{A} if $\langle X_2 W Y_1^H, N \rangle = \langle W, N_{12} \rangle$ is greater than zero for some arbitrarily small W . Thus, $N_{12} = 0$.

By a similar argument, $N_{21} = 0$. \square

Finally, we can, in a sense, eliminate the derogatory \hat{A} matrices by the following lemma.

Lemma 2.4.3 *If \hat{A} has a derogatory canonical structure, J , then one of the irreducible*

blocks of J , say $J^{(1)}$ is defective, and

$$N = Y_1 N_{11} X_1^H.$$

Proof: Suppose that all of the irreducible blocks of J were simple, i.e. $J = \lambda I$. Then, from the previous lemma, $N = y_1 n x_1^H$ (where x_1 and y_1 are vectors and n is a scalar).

$\hat{A} + \mu I$ has a repeated eigenvalue of $\lambda + \mu$.

$\hat{A} + \mu I$ will be closer to A if $\langle \mu I, N \rangle = \text{Real}\{\bar{\mu} n\}$ is greater than zero for some arbitrarily small μ . Thus $n = 0$ and $\hat{A} = A$ contradicting hypothesis.

Combining this with the result of the previous lemma gives the the conclusion. \square

We summarize with the following corollary.

Lemma 2.4.4 *The nearest matrix $\hat{A} \in \mathcal{D}$ to A is reached by a perturbation of the form*

$$N = Y p(J(0)^T) X^H,$$

where $p(x)$ is a $k - 1$ -degree polynomial with vanishing constant term, and X, Y are the right and (corresponding dual) left generalized eigenvectors of a canonical $k \times k$ Jordan block, J , in the Jordan structure of \hat{A} , and $J(0)$ is the $k \times k$ Jordan block with eigenvalue 0.

Proof: Prior lemmae have shown that N is of the form $N = Y W X^H$. The similarity transformation $e^{\sigma M} \hat{A} e^{-\sigma M} \in \mathcal{D}$ is closer to A for some arbitrarily small σ whenever the commutator, $M \hat{A} - \hat{A} M$ has positive inner product with N .

Thus $\langle [M, \hat{A}], N \rangle = \langle M, [N, \hat{A}^H] \rangle$ must vanish for all M . Giving $N \hat{A}^H - \hat{A}^H N = 0$ which implies that $W J(0)^T = J(0)^T W$, and thus $W = p(J(0)^T)$. \square

We now need only really consider the case where the Jordan structure of \hat{A} is a single canonical Jordan block.

Lemma 2.4.5 *Let the Jordan block J of \hat{A} be of size greater than 2. The polynomial $p(x)$ in*

$$N = Y p(J^T) X^H,$$

must be of the form $p(x) = ax^{k-1} + bx^{k-2}$.

Proof: Suppose that the coefficient, c , of x^j ($j < k - 2$) is nonvanishing. Then we consider the matrix $\hat{A} + XMY^H$, where $M_{kj} = s$ ($M_{pq} = 0$ elsewhere).

$\hat{A} + XMY^H$, while not necessarily having the same canonical structure as J still has λ repeated at least twice, since

$$(\hat{A} + XMY^H) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}.$$

However, $\hat{A} + XMY^H$ is closer to A so long as $\langle M, p(J(0)^T) \rangle = sc$ does not vanish for arbitrarily small s . Thus we must have $c = 0$. \square

Thus, for all A not in \mathcal{D} , $N = A - \hat{A}$ is of the form that one generically expects from either a J_2 or a J_3 canonical structure. In what follows, we shall see that if we allow \hat{A} to take values in \mathbb{C} then the $A - \hat{A}$ will be rank 1.

Theorem 2.4.6 *Let $\hat{A} \in \mathcal{D}$ be the nearest matrix with a repeated eigenvalue to the given matrix A with distinct eigenvalues. Then*

$$N = A - \hat{A} = \sigma uv^H,$$

where u and v are the left and right eigenvectors of \hat{A} and σ is a scalar.

Proof:

Assuming the canonical structure of \hat{A} is $J = J_k(\lambda)$, we have from the previous lemmatae, we have that

$$N = Y(a(J(0)^T)^{k-1} + b(J(0)^T)^{k-2})X^H,$$

$b \neq 0$.

Consider the matrix $\hat{A} + X(a'(J(0)^T)^{k-1} + b'(J(0)^T)^{k-2})Y^H$, where $b' = k\mu^{k-1}$ and $a' = -(k-1)\mu^k$. It is easy to show that $J + (a'(J(0)^T)^{k-1} + b'(J(0)^T)^{k-2})$ has a repeated (J_2) eigenvalue $\lambda + \mu$.

if for some arbitrarily small μ $\langle X(a'(J(0)^T)^{k-1} + b'(J(0)^T)^{k-2})Y^H, N \rangle$ is greater than zero, $\hat{A} + X(a'(J(0)^T)^{k-1} + b'(J(0)^T)^{k-2})Y^H$ will be closer to A than \hat{A} .

$\langle X(a'(J(0)^T)^{k-1} + b'(J(0)^T)^{k-2})Y^H, N \rangle = \text{Real}\{\bar{b}'b + \bar{a}'a\}$. Substituting for a' and b' we

have $\text{Real}\{\bar{b}'b + \bar{a}'a\} = \text{Real}\{\bar{\mu}^{k-1}(kb - (k-1)\bar{\mu}a)\}$. If we take $|\mu| < \frac{b}{a}$ and let select the phase of μ such that $\mu^{k-1}b$ is positive real, then $\langle X(a'(J(0)^T)^{k-1} + b'(J(0)^T)^{k-2})Y^H, N \rangle > 0$.

Thus, we have $\hat{A} + X(a'(J(0)^T)^{k-1} + b'(J(0)^T)^{k-2})Y^H$ closer to A than \hat{A} .

We must conclude that $b = 0$ and therefore that N is rank 1 of the form asserted. \square

The reader will note that it is not always possible to find such a' and b' if one is constrained to real matrices, but one can always do so over complex numbers. In fact, the illustration at the beginning of this section demonstrates this shortcoming of the reals. If this illustration had been drawn with complex a and b axes, then it would have been apparent that A was closer to a complex \hat{A} .

We have now established rigorously that any matrix A with distinct eigenvalues must lie along a rank 1 normal from the nearest $\hat{A} \in \mathcal{D}$. Thus \hat{A} can be determined by examining the singular values of $A - \lambda I$.

What remains are some lemmata concerning the singular values themselves so that it can be determined what sort of behavior one must look for in the singular values in order to locate \hat{A} . The previous section shows that the critical points of σ_{\min} are the points of interest in determining \hat{A} . However, one may also wonder what happens when σ_{\min} is not simple (and hence not rigorously differentiable), and whether or not one must examine the critical points of the other singular values of $A - \lambda I$. We assure the reader that this will not be trouble for us. In Section 2.6, we will be able to show that the critical points of other singular values will not produce minima, and that, the minimum will occur on a critical point where σ_{\min} is almost always simple.

2.5 Conditioning and stability

In this section, we wish to examine the conditioning of the problem of finding the $\hat{A} \in \mathcal{D}$ nearest to A . The conditioning of \hat{A} is closely related to the behavior of the singular value decomposition of $\hat{A} - \lambda I$ under first order variations. We will find that the matrices with ill-conditioned \hat{A} on \mathcal{D} are sitting close to “focal points” of \mathcal{D} analogous to focal points in geometric optics, explained in Section 2.5.2.

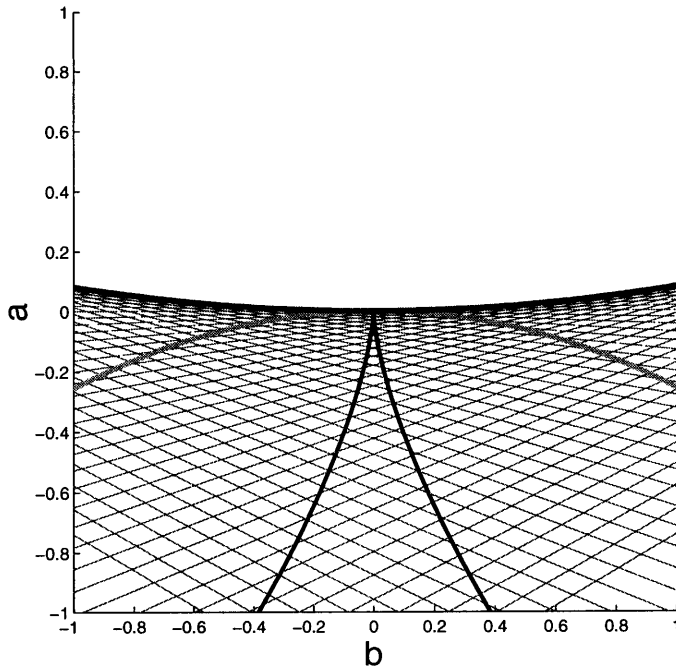


Figure 2-14: The normals of the cubic double root variety forming their parabola-like envelope. The concave down arc in the plot is another part of the envelope which occurs over cubics with complex coefficients.

2.5.1 Sensitivity analysis of $A = \hat{A} + \sigma uv^H$

Consider the normal lines passing through a matrix $\hat{A} \in \mathcal{D}$. We can parameterize these lines by the real or complex line $A = \hat{A} + \sigma uv^H$, where σ is real and the phases on u and v are arbitrary (alternatively, we could fix the phase of uv^H and let σ be complex). For most A we expect this decomposition ($A = \hat{A} + \sigma uv^H$) to be locally invertible. We define a *focal point* as some matrix A for which the decomposition is not locally unique, i.e., where the Jacobian of this decomposition is singular.

Geometrically, we may think of the set of focal points as the evolute of \mathcal{D} , i.e., the envelope of normals of \mathcal{D} . A point which is in the evolute can be thought of as the intersection of two infinitesimally separated normals.

It will be useful to do some dimension counting in both complex and real versions of \mathcal{D} so that we might better understand where the degrees of freedom are. (We shall always refer to one real parameter as one degree of freedom.)

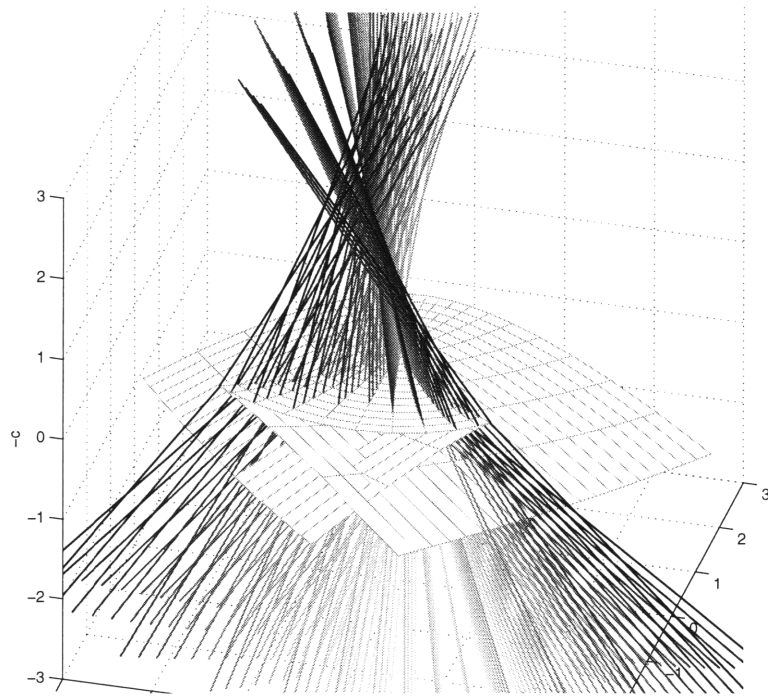


Figure 2-15: The normals of the swallowtail. A faint hood-like shape begins to emerge.

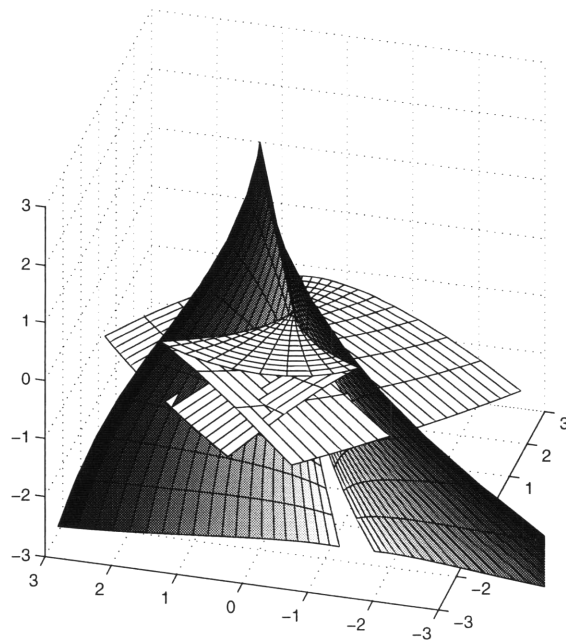


Figure 2-16: The swallowtail and its hood-like evolute of ill-conditioned problems.

The solution to the problem of finding a stationary point \hat{A} to A can be expressed in terms of decompositions of A as follows:

$$A = \sigma uv^H + \hat{A}, \quad (2.13)$$

$$A = \sigma uv^H + \lambda I + \tilde{U} \tilde{\Sigma} \tilde{V}^H, \quad (2.14)$$

$$A = \lambda I + U \Sigma V^H \quad (2.15)$$

where $u^H v = 0$, $\tilde{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_{n-1})$, $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_{n-1}, \sigma) = \begin{pmatrix} \tilde{\Sigma} & 0 \\ 0 & \sigma \end{pmatrix}$, $U = (\tilde{U} \quad u)$, $V = (\tilde{V} \quad v)$, and U, V are unitary matrices. For the real case, Equation (2.13) decomposes the n^2 degrees of freedom of A into $n(n-1)/2$ for U , $n(n-1)/2$ for V , n for Σ , and 1 for λ , which leaves exactly 1 degree of freedom extra to satisfy the constraint $u^H v = 0$. For the complex case, A has $2n^2$ degrees of freedom, U has n^2 , V has n^2 , Σ has n , and λ has 2, leaving $n+2$ surplus degrees of freedom. Since the complex decomposition is only unique up to transformation of the form $U, V \rightarrow U\Delta, V\Delta$ (Δ a diagonal unitary matrix), n of those surplus degrees of freedom are ineffective, leaving two to satisfy the constraints $\text{Real}\{u^H v\} = 0$, $\text{Imag}\{u^H v\} = 0$.

Thus, by counting dimensions we see that the decomposition which reveals the nearest $\hat{A} \in \mathcal{D}$ to A is really a singular value decomposition in which one dimension has been removed by the constraint $u^H v = 0$ and one dimension has been added by the presence of the λI term. Our strategy to vary \hat{A} will be to vary the SVD (as in [60]), and then to use the variation in λ to satisfy the varied constraint ($u^H v = 0$) equation.

Note, if we were to vary this decomposition for an element of $A \in \mathcal{D}$, we set $\sigma = 0$ in Equation (2.13). In the real case, this eliminates one degree of freedom corresponding to \mathcal{D} 's real co-dimension of 1. In the complex case, this eliminates not only the degree of freedom in σ , but also the extra unitary freedom of u and v corresponding to \hat{A} 's complex co-dimension of 1.

Theorem 2.5.1 *Assuming that the singular values of $A - \lambda I$ are distinct, the first order behavior of the decomposition*

$$A = \hat{A} - \sigma uv^H,$$

is given by

$$\dot{A} = \dot{\hat{A}} + (u^H \dot{A} v) u v^H + \sigma \tilde{U} (\tilde{U}^H \dot{u}) v^H + \sigma u (\tilde{V}^H \dot{v})^H \tilde{V}^H,$$

with

$$\begin{pmatrix} \tilde{U}^H \dot{u} \\ \tilde{V}^H \dot{v} \end{pmatrix} = \begin{pmatrix} \Phi(\sigma) & \Psi(\sigma) \\ \Psi(\sigma) & \Phi(\sigma) \end{pmatrix} \begin{pmatrix} \tilde{U}^H (\dot{A} - \dot{\lambda} I) v \\ \tilde{V}^H (\dot{A}^H - \dot{\lambda} I) u \end{pmatrix},$$

where $\Phi(\sigma) = \sigma(\sigma^2 I - \tilde{\Sigma}^2)^{-1}$, $\Psi(\sigma) = \tilde{\Sigma}(\sigma^2 I - \tilde{\Sigma}^2)^{-1}$, and $\dot{\lambda}$ is given by

$$\begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix} \begin{pmatrix} \dot{\lambda}_{real} \\ \dot{\lambda}_{imag} \end{pmatrix} = \begin{pmatrix} Real\{l^H \Psi p + l^H \Phi q + q^H \Psi r + p^H \Phi r\} \\ Imag\{l^H \Psi p + l^H \Phi q + q^H \Psi r + p^H \Phi r\} \end{pmatrix},$$

where σ_{xx} , σ_{xy} , and σ_{yy} are given by Sun's theorem.

Proof: Varying the decomposition (2.13) along a differentiable curve $A(t)$, we have

$$\dot{A} = \dot{\hat{A}} + \dot{\sigma} u v^H + \sigma \dot{u} v^H + \sigma u \dot{v}^H,$$

where $\dot{u}^H v + u^H \dot{v} = 0$. We will proceed by solving for an expression for $\dot{u}^H v + u^H \dot{v}$, finding values for σ and the phase of $u v^H$ for which the variation of the constraint is satisfied (it is convenient for us to take σ to be real and to adjust the phases of $u v^H$ implicitly).

Differentiating $(A - \lambda I)v = \sigma u$ and $u^H(A - \lambda I) = \sigma v^H$, we have that

$$(\dot{A} - \dot{\lambda} I)v - \dot{\sigma} u = \sigma \dot{u} - (A - \lambda I)\dot{v}$$

and

$$(\dot{A}^H - \dot{\lambda} I)u - \dot{\sigma} v = \sigma \dot{v} - (A^H - \bar{\lambda} I)\dot{u}.$$

We may substitute $A - \lambda = U \Sigma V^H$, obtaining

$$U^H (\dot{A} - \dot{\lambda} I)v - \dot{\sigma} e_n = \sigma U^H \dot{u} - \Sigma V^H \dot{v}$$

and

$$V^H (\dot{A}^H - \dot{\lambda} I)u - \dot{\sigma} e_n = \sigma V^H \dot{v} - \Sigma U^H \dot{u},$$

where $e_n = (0 \ \cdots \ 0 \ 1)^H$, which can be collected into $2n - 2$ equations,

$$\begin{pmatrix} \sigma & -\tilde{\Sigma} \\ -\tilde{\Sigma} & \sigma \end{pmatrix} \begin{pmatrix} \tilde{U}^H \dot{u} \\ \tilde{V}^H \dot{v} \end{pmatrix} = \begin{pmatrix} \tilde{U}^H (\dot{A} - \dot{\lambda} I) v \\ \tilde{V}^H (\dot{A}^H - \dot{\lambda} I) u \end{pmatrix} \quad (2.16)$$

and 2 identical equations,

$$u^H (\dot{A} - \dot{\lambda} I) v - \dot{\sigma} = \sigma (u^H \dot{u} - v^H \dot{v}). \quad (2.17)$$

If $\sigma = \sigma_n$ is distinct from the other σ_i then we can invert the matrix in (2.16) obtaining

$$\begin{pmatrix} \tilde{U}^H \dot{u} \\ \tilde{V}^H \dot{v} \end{pmatrix} = \begin{pmatrix} \Phi(\sigma) & \Psi(\sigma) \\ \Psi(\sigma) & \Phi(\sigma) \end{pmatrix} \begin{pmatrix} \tilde{U}^H (\dot{A} - \dot{\lambda} I) v \\ \tilde{V}^H (\dot{A}^H - \dot{\lambda} I) u \end{pmatrix}, \quad (2.18)$$

where $\Phi(\sigma) = \sigma(\sigma^2 I - \tilde{\Sigma}^2)^{-1}$ and $\Psi(\sigma) = \tilde{\Sigma}(\sigma^2 I - \tilde{\Sigma}^2)^{-1}$. Since $U^H \dot{U}$ and $V^H \dot{V}$ must be skew-symmetric, (2.17) separates into real and imaginary parts

$$\dot{\sigma} = \text{Real}\{u^H \dot{A} v\} \quad (2.19)$$

$$-i(u^H \dot{u} - v^H \dot{v}) = \text{Imag}\{u^H \dot{A} v\}, \quad (2.20)$$

where we have used $u^H v = 0$.

We now apply the differentiated constraint,

$$\dot{u}^H v + u^H \dot{v} = (U^H \dot{u})^H r + l^H (V^H \dot{v}) = 0,$$

where we have used $v = \tilde{U} r$ and $u = \tilde{V} l$ ($r = \tilde{U}^H v$ and $l = \tilde{V}^H u$). Substituting (2.18) into this, we have

$$\dot{\lambda} (2l^H \Psi r) + \dot{\lambda} (l^H \Phi l + r^H \Phi r) = l^H \Psi p + l^H \Phi q + q^H \Psi r + p^H \Phi r, \quad (2.21)$$

where $p = \tilde{U}^H \dot{A} v$ and $q = \tilde{V}^H \dot{A}^H u$, which one can solve for $\dot{\lambda}$ solving

$$\begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix} \begin{pmatrix} \dot{\lambda}_{real} \\ \dot{\lambda}_{imag} \end{pmatrix} = \begin{pmatrix} \text{Real}\{l^H \Psi p + l^H \Phi q + q^H \Psi r + p^H \Phi r\} \\ \text{Imag}\{l^H \Psi p + l^H \Phi q + q^H \Psi r + p^H \Phi r\} \end{pmatrix}.$$

One then has

$$\dot{\hat{A}} = \dot{A} - \text{Real}\{u^H \dot{A} v\} u v^H - \sigma U (U^H \dot{u}) v^H - \sigma u (V^H \dot{v})^H V^H \quad (2.22)$$

$$\dot{\hat{A}} = \dot{A} - \text{Real}\{u^H \dot{A} v\} u v^H - \sigma \tilde{U} (\tilde{U}^H \dot{u}) v^H - \sigma u (\tilde{V}^H \dot{v})^H \tilde{V}^H - \sigma (u^H \dot{u} - \dot{v}^H v) u v^H \quad (2.23)$$

$$\dot{\hat{A}} = \dot{A} - (u^H \dot{A} v) u v^H - \sigma \tilde{U} (\tilde{U}^H \dot{u}) v^H - \sigma u (\tilde{V}^H \dot{v})^H \tilde{V}^H, \quad (2.24)$$

where we have used Equation (2.20) to get (2.24) from (2.23). \square

One can interpret the formula for $\dot{\hat{A}}$ as displacement resulting purely from the change in the phase and length of the normal (second term) plus a tangential variation of \hat{A} (third and fourth terms) which results from a change in the direction of the normal from A to \hat{A} . The magnitude of the tangential part of the variation is $\sqrt{\|\dot{u}\|^2 + \|\dot{v}\|^2}$.

The set of A with \hat{A} having infinite condition number is given by finding those A for which $\dot{\hat{A}}$ can be nonzero when \dot{A} vanishes. In this case, we set the p and q terms in (2.21) to 0 and solve

$$\det \begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix} \begin{pmatrix} \dot{\lambda}_{real} \\ \dot{\lambda}_{imag} \end{pmatrix} = 0,$$

as an equation of σ equivalent to

$$|2l^H \Psi(\sigma) r| = |l^H \Phi(\sigma) l + r^H \Phi(\sigma) r|,$$

which gives the following corollary.

Corollary 2.5.2 *The nearby $\hat{A} \in \mathcal{D}$ to a generic matrix A is ill-conditioned whenever $|2l^H \Psi r| - |l^H \Phi l + r^H \Phi r|$ is small (i.e. $\begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{pmatrix}$ is nearly singular).*

2.5.2 Ill-posed problems and principal curvatures

In this section we explore the differential geometry of \mathcal{D} itself, as a co-dimension 1 manifold embedded in \mathbb{R}^{n^2} or \mathbb{C}^{n^2} . We will give an additional interpretation of the focal points of \mathcal{D} in terms of principal curvatures.

The classical study of principal curvatures is concerned with surfaces in \mathbb{R}^3 (as in [25]). In this setting, one uses the unit normal field \hat{n} of a surface M to construct a map from M to \mathbb{S}^2 . Studying the local behavior of this map ultimately results in a quadratic form on the tangent space of M at a point. The reciprocal lengths of the principal axes of this quadratic form are called the principal curvatures, their sum the mean curvature and their product the Gaussian curvature.

One way to generalize this approach which is suitable for any manifold of co-dimension 1 is to use the normal to construct a natural quadratic form on the tangent space and then examine the eigenvalues of this form.

We first look at an n -sphere of radius l embedded in \mathbb{R}^{n+1} . Geodesics on \mathbb{S}^n satisfy the equations

$$\begin{aligned}\frac{dp}{dt} &= v \\ \frac{dv}{dt} &= -\frac{v \cdot v}{l} \hat{n}\end{aligned}$$

where p is a point on the sphere and v is the velocity, satisfying $p^H v = 0$, and $\hat{n} = p/l$. One might recognize the derivative of v as the centripetal acceleration in classical mechanics.

For a more general manifold of co-dimension 1, one has (at least locally) a unit normal \hat{n} and geodesic equations of the form

$$\frac{dp}{dt} = v \tag{2.25}$$

$$\frac{dv}{dt} = -\Gamma(v, v) \tag{2.26}$$

where $\Gamma(v, w)$ is the Levi-Civita connection. However, for manifolds of co-dimension 1, the Levi-Civita connection must be of the form $\Gamma(v, w) = a(v, w)\hat{n}$ where the acceleration, $a(v, w)$,

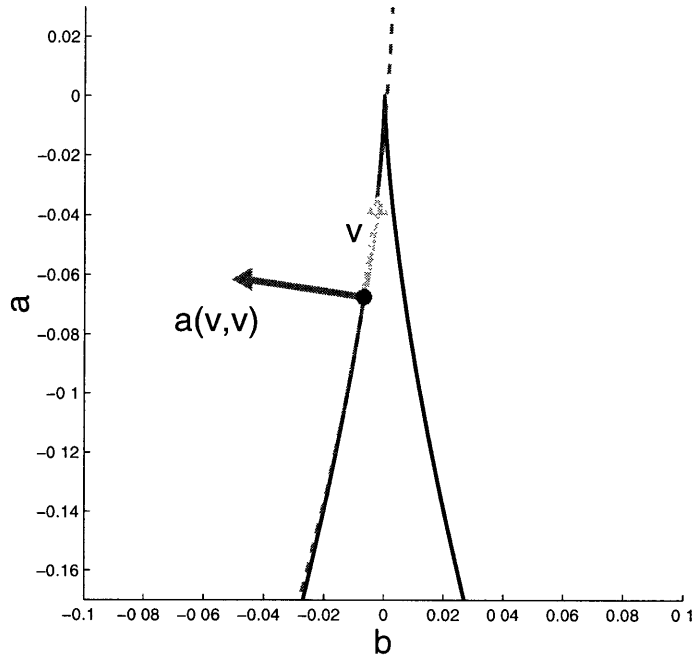


Figure 2-17: In the cubic polynomial space, a particle moves along the double root variety, with velocity v and centripetal acceleration $a(v, v)$.

is a symmetric bilinear (*not* complex sesquilinear) function defined on the tangent space at p and determined by $\frac{dv}{dt} \cdot \hat{n} + v \cdot \frac{d\hat{n}}{dt} = 0$.

If we consider $a(v, v)$ to be an instantaneous centripetal acceleration along a geodesic with velocity v through p (see Figure 2-17), then we may define the principal curvatures at p as the extrema of the function $|a(v, v)|$ over the unit ball in the tangent space at p . As with the sphere, the center of curvature is located at $p - \frac{1}{a(v, v)}\hat{n}$. We may refer to the reciprocals of the principal curvatures as the focal lengths along \hat{n} from p (see Figure 2-18)

With this in mind, we now consider the differentiable subset of \mathcal{D} . For the remainder of this section any matrix A is a point in a differentiable neighborhood of \mathcal{D} with decomposition

$$A = \tilde{U}\tilde{\Sigma}\tilde{V}^H + \lambda I$$

and normal $N = uv^H$ ($u^Hv = 0$). Any element H of the tangent space of A must satisfy $u^HHv = 0$.

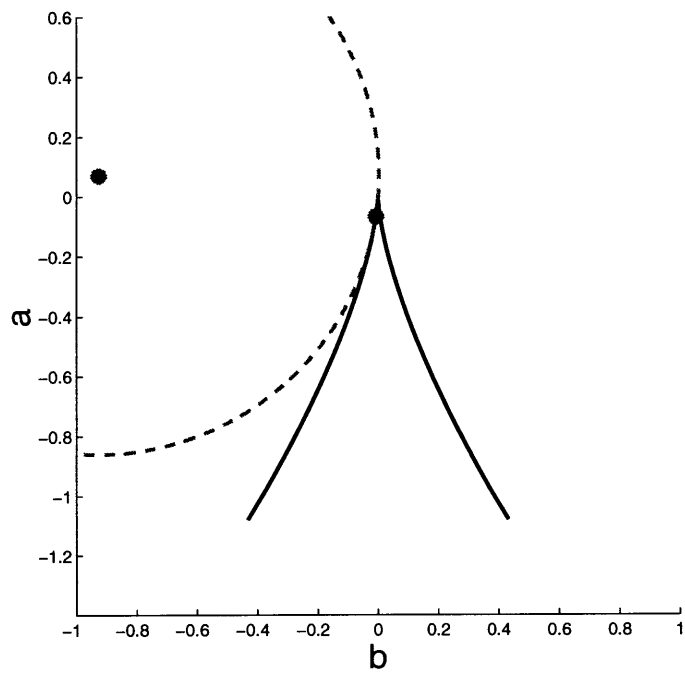


Figure 2-18: The centripetal acceleration of the particle moving along the double root variety can be associated with a circle centered at the focal point of the double root variety.

If $A(t)$ is a differentiable curve in \mathcal{D} through A then we differentiate $u^H H v = 0$ to get

$$\begin{aligned} \frac{d}{dt}(u^H H v) &= \dot{u}^H H v + u^H H \dot{v} - a(H, H) = 0 \\ (\tilde{U}^H \dot{u})^H p + q^H \tilde{V}^H \dot{v} &= a(H, H), \end{aligned}$$

where $q = \tilde{U}^H H v$ and $p = \tilde{V}^H H v$ (note that since $u^H H v = 0$ we have $H v = \tilde{U} p$ and $u^H H = q^H \tilde{V}^H$).

Since $A \in \mathcal{D}$ we may set $\sigma = 0$ and employ Equations (2.18, 2.21, 2.24) to see that

$$\begin{pmatrix} \tilde{U}^H \dot{u} \\ \tilde{V}^H \dot{v} \end{pmatrix} = \begin{pmatrix} 0 & \tilde{\Sigma}^{-1} \\ \tilde{\Sigma}^{-1} & 0 \end{pmatrix}^{-1} \begin{pmatrix} p - \dot{\lambda} r \\ q - \dot{\lambda} l \end{pmatrix},$$

with $\dot{\lambda}$ given by

$$\dot{\lambda}(2l^H \tilde{\Sigma}^{-1} r) = l^H \tilde{\Sigma}^{-1} p q^H \tilde{\Sigma}^{-1} r.$$

Substituting and simplifying, we have

$$a(H, H) = -2p^H \tilde{\Sigma}^{-1} q + \frac{(l^H \tilde{\Sigma}^{-1} p + q^H \tilde{\Sigma}^{-1} r)^2}{2l^H \tilde{\Sigma}^{-1} r}.$$

Since $\tilde{U} p v^H + u q^H \tilde{V}^H$ is a tangent vector for any p and q , we may consider $a(H, H)$ as a function of $2n - 2$ independent p and q vectors.

We see that the quadratic form $a(H, H)$ can be written as $z^T W z$, where

$$z = \begin{pmatrix} \bar{p} \\ q \end{pmatrix} \tag{2.27}$$

and

$$W = \begin{pmatrix} 0 & -\tilde{\Sigma}^{-1} \\ -\tilde{\Sigma}^{-1} & 0 \end{pmatrix} + \frac{1}{2\bar{l}^T \tilde{\Sigma}^{-1} r} \begin{pmatrix} \tilde{\Sigma}^{-1} r \\ \tilde{\Sigma}^{-1} \bar{l} \end{pmatrix} (r^T \tilde{\Sigma}^{-1} \quad \bar{l}^T \tilde{\Sigma}^{-1}). \tag{2.28}$$

By a simple orthogonal similarity transformation we have

$$Q W Q^T = Z + a a^T, \tag{2.29}$$

where $Z = \begin{pmatrix} -\tilde{\Sigma}^{-1} & 0 \\ 0 & \tilde{\Sigma}^{-1} \end{pmatrix}$ and $a = \frac{1}{2\sqrt{l^T \tilde{\Sigma}^{-1} r}} \begin{pmatrix} \tilde{\Sigma}^{-1}(r + \bar{l}) \\ \tilde{\Sigma}^{-1}(r - \bar{l}) \end{pmatrix}$, a real diagonal matrix plus a rank-1 matrix.

We now seek the extrema of $z^T W z$ where $|H|^2 = |x|^2 + |y|^2 = 1$. Note that $W^T = W$. Varying $|z^T W z|$ we see that the condition for z to be an extremum of $|z^T W z|$ is

$$Wz = \zeta \bar{z},$$

a somewhat unusual eigenvalue equation over \mathbb{C} .

Over \mathbb{R} this reduces to the problem of finding the eigenvalues of a rank-1 update, $Z + aa^T$. An eigenvalue ζ of this matrix then satisfies

$$a^T(\zeta I - Z)^{-1}a = 1,$$

which, in our terms, is

$$\begin{pmatrix} (r+l)^T \tilde{\Sigma}^{-1} & (r-l)^T \tilde{\Sigma}^{-1} \end{pmatrix} \begin{pmatrix} (\zeta I + \tilde{\Sigma}^{-1})^{-1} & 0 \\ 0 & (\zeta I - \tilde{\Sigma}^{-1})^{-1} \end{pmatrix} \begin{pmatrix} \tilde{\Sigma}^{-1}(r+l) \\ \tilde{\Sigma}^{-1}(r-l) \end{pmatrix} = 4l^T \tilde{\Sigma}^{-1} r.$$

Multiplying this out and simplifying gives

$$r^T \frac{-1}{\zeta} \left(\frac{1}{\zeta^2} I - \tilde{\Sigma}^2 \right)^{-1} r + l^T \frac{-1}{\zeta} \left(\frac{1}{\zeta^2} - \tilde{\Sigma}^2 \right)^{-1} l + 2l^T \tilde{\Sigma} \left(\frac{1}{\zeta^2} - \tilde{\Sigma}^2 \right)^{-1} r = 0.$$

$$r^T \Phi \left(\frac{-1}{\zeta} \right) r + l^T \Phi \left(\frac{-1}{\zeta} \right) l + 2l^T \Psi \left(\frac{-1}{\zeta} \right) r = 0,$$

where $\Phi(\sigma) = \sigma(\sigma^2 I - \tilde{\Sigma}^2)^{-1}$ and $\Psi(\sigma) = \tilde{\Sigma}(\sigma^2 I - \tilde{\Sigma}^2)^{-1}$. The center of curvature is then $A + \sigma uv^H$ where $\sigma = \frac{-1}{\zeta}$. We may now interpret the vanishing of the second real-axis derivative at a stationary point of the spectral portrait as equivalent to being on a focal point of the real double eigenvalue variety.

Over \mathbb{C} , we solve $QWQ^H(Qz) = \zeta Q\bar{z}$ by first observing that given an eigen-pair z, ζ , the eigen-pair $ze^{i\frac{1}{2}\angle\zeta}, \zeta e^{-i\angle\zeta}$ is also a solution. Thus, while acknowledging that λ can have any

phase. we may restrict ζ to the reals for purposes of computation, in which case we solve,

$$\begin{pmatrix} QWQ^H & -\zeta I \\ -\zeta I & Q\bar{W}Q^H \end{pmatrix} \begin{pmatrix} z \\ \bar{z} \end{pmatrix}.$$

Following a derivation of the usual rank-1 update formula, we rewrite this as

$$\begin{pmatrix} Z & -\zeta I \\ -\zeta I & Z \end{pmatrix} \begin{pmatrix} z \\ \bar{z} \end{pmatrix} + \begin{pmatrix} aa^T & 0 \\ 0 & \bar{a}\bar{a}^T \end{pmatrix} \begin{pmatrix} z \\ \bar{z} \end{pmatrix} = 0.$$

Letting $k = a^T z$ we have

$$\begin{aligned} & \begin{pmatrix} Z & -\zeta I \\ -\zeta I & Z \end{pmatrix} \begin{pmatrix} z \\ \bar{z} \end{pmatrix} + \begin{pmatrix} ak \\ \bar{a}\bar{k} \end{pmatrix} = 0 \\ & \begin{pmatrix} z \\ \bar{z} \end{pmatrix} + \begin{pmatrix} Z(Z^2 - \zeta^2 I)^{-1}a & \zeta(Z^2 - \zeta^2 I)^{-1}\bar{a} \\ \zeta(Z^2 - \zeta^2 I)^{-1}a & Z(Z^2 - \zeta^2 I)^{-1}\bar{a} \end{pmatrix} \begin{pmatrix} k \\ \bar{k} \end{pmatrix} = 0 \\ & \begin{pmatrix} k \\ \bar{k} \end{pmatrix} + \begin{pmatrix} a^T Z(Z^2 - \zeta^2 I)^{-1}a & a^T \zeta(Z^2 - \zeta^2 I)^{-1}\bar{a} \\ \bar{a}^T \zeta(Z^2 - \zeta^2 I)^{-1}a & \bar{a}^T Z(Z^2 - \zeta^2 I)^{-1}\bar{a} \end{pmatrix} \begin{pmatrix} k \\ \bar{k} \end{pmatrix} = 0. \end{aligned}$$

Note that the off diagonals are real, while the diagonals are each others complex conjugates, and that $\begin{pmatrix} k \\ \bar{k} \end{pmatrix}$ must be a fixed point of this matrix. A necessary and sufficient condition for such a k to exist is

$$\det \begin{pmatrix} a^T Z(Z^2 - \zeta^2 I)^{-1}a + 1 & a^T \zeta(Z^2 - \zeta^2 I)^{-1}\bar{a} \\ \bar{a}^T \zeta(Z^2 - \zeta^2 I)^{-1}a & \bar{a}^T Z(Z^2 - \zeta^2 I)^{-1}\bar{a} + 1 \end{pmatrix} = 0,$$

or equivalently,

$$|a^T Z(Z^2 - \zeta^2 I)^{-1}a + 1| = |\bar{a}^T \zeta(Z^2 - \zeta^2 I)^{-1}a|.$$

Substituting and simplifying we have

$$|\bar{r}^T \Phi\left(\frac{-1}{\zeta}\right)r + \bar{l}^T \Phi\left(\frac{-1}{\zeta}\right)l| = |2\bar{l}^T \Psi\left(\frac{-1}{\zeta}\right)r|.$$

Thus we have seen that an additional way to interpret these focal points is to imagine an osculating circle of radius $\frac{1}{\zeta}$, centered at the focal point and passing through \mathcal{D} along a

principal direction.

2.5.3 Stability of the fixed point algorithm

The fixed point algorithm can be viewed as a series of guesses at the repeated eigenvalue $\lambda = \lim_{i \rightarrow \infty} \mathbf{e}_i$, where $\mathbf{e}_{i+1} = f(\mathbf{e}_i)$. In iterated systems of this sort, convergence is to some fixed point $x_0 = f(x_0)$, and the linear multiplier governing convergence is $f'(x_0)$.

If $f'(x_0) > 1$ then the system diverges, either to another fixed point or to a limit cycle. If $f'(x_0) < 1$, but close to 1, then convergence can be very slow. We therefore need to examine $f'(x_0)$ to understand the stability of this algorithm. We linearize $f(x)$ about a fixed point λ .

$$f(\lambda + \delta\lambda) = \text{cluster}(A - (\sigma + \delta\sigma)(u + \delta u)(v + \delta v)^H)$$

where $u + \delta u$, $v + \delta v$, and $\sigma + \delta\sigma$ are the singular vectors and value of $A - (\lambda + \delta\lambda)I$ and u , v , and σ are the singular vectors and value of $A - \lambda I$. This may be rewritten as

$$f(\lambda + \delta\lambda) = \text{cluster}(\hat{A} - \delta\sigma uv^H - \sigma\delta uv^H - \sigma u\delta v^H).$$

Assuming that the `cluster` function simply averages the two nearest eigenvalues, we have a first variation giving

$$f(\lambda + \delta\lambda) = \lambda + \frac{1}{2} \text{tr}((y_1 \quad y_2)^H (-\delta\sigma uv^H - \sigma\delta uv^H - \sigma u\delta v^H) (x_1 \quad x_2)),$$

where y_2 and x_1 are the left and right eigenvectors of \hat{A} , $y_2^H \hat{A} = y_2^H$, and $\hat{A} x_1 = x_1$.

Since v is a right eigenvector and u is a left eigenvector, we may take $x_1 = \frac{1}{l^H \tilde{\Sigma}^{-1} r} v$, $y_2 = u$, $x_2 = \frac{1}{l^H \tilde{\Sigma}^{-1} r} \tilde{V} \tilde{\Sigma}^{-1} r$, and $y_1 = \tilde{U} \tilde{\Sigma}^{-1} l$. Thus we see that

$$\delta f = -\frac{1}{2l^H \tilde{\Sigma}^{-1} r} (\sigma l^H \tilde{\Sigma}^{-1} \tilde{U}^H \delta u + \sigma \delta v \tilde{V} \tilde{\Sigma}^{-1} r).$$

Substituting from Theorem 2.5.1 (with A fixed), we have

$$\delta f = \frac{1}{2l^{H\tilde{\Sigma}^{-1}r}}(\delta\lambda 2l^H(\tilde{\Sigma}^{-1} + \Psi)r + \delta\bar{\lambda}(l^H\Phi l + r^H\Phi r)),$$

simplifying to

$$\delta f = \delta\lambda + \frac{1}{2l^{H\tilde{\Sigma}^{-1}r}}(\delta\lambda 2l^H\Psi r + \delta\bar{\lambda}(l^H\Phi l + r^H\Phi r)),$$

which is the linearization of the iterated map f on a small perturbation $\delta\lambda$.

The behavior of this map will determine the nature of the fixed point. We recognize elements of the second derivative of the spectral portrait in this formula. Using Sun's formulae, we have

$$\delta f = \delta\lambda + \frac{1}{2l^{H\tilde{\Sigma}^{-1}r}}(\delta\lambda(\frac{\sigma_{xx} - \sigma_{yy}}{2} - i\sigma_{xy}) + \delta\bar{\lambda}(\frac{\sigma_{xx} + \sigma_{yy}}{2})),$$

from which we may assert the following theorem.

Theorem 2.5.3 *The fixed point algorithm iteration linearizes to*

$$\begin{pmatrix} \delta\lambda_{real} \\ \delta\lambda_{imag} \end{pmatrix}_{i+1} = \left(I - \begin{pmatrix} l^{H\tilde{\Sigma}^{-1}r_{real}} & l^{H\tilde{\Sigma}^{-1}r_{imag}} \\ l^{H\tilde{\Sigma}^{-1}r_{imag}} & -l^{H\tilde{\Sigma}^{-1}r_{real}} \end{pmatrix}^{-1} \begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix} \right) \begin{pmatrix} \delta\lambda_{real} \\ \delta\lambda_{imag} \end{pmatrix}_i. \quad (2.30)$$

We can make a number of qualitative statements about Equation (2.30). When the Hessian of σ_{\min} is singular (i.e., near a focal point) this iteration can stagnate. Secondly, if $l^{H\tilde{\Sigma}^{-1}r}$ is small (corresponding to the Puiseux expansion in Section 2.3.1 having poor behavior), then the iteration is more likely to diverge. Lastly, since

$$\begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix} \rightarrow \begin{pmatrix} l^{H\tilde{\Sigma}^{-1}r_{real}} & l^{H\tilde{\Sigma}^{-1}r_{imag}} \\ l^{H\tilde{\Sigma}^{-1}r_{imag}} & -l^{H\tilde{\Sigma}^{-1}r_{real}} \end{pmatrix}^{-1}$$

as $\sigma \rightarrow 0$, we see that fast convergence is assured for small enough σ .

2.5.4 Sensitivity of the critical points of the spectral portrait

With results which will be derived in Section 2.6, the problem of finding the nearest $\hat{A} \in \mathcal{D}$ to A can be reduced to locating the critical points of the spectral portrait of A , $\sigma(x, y) =$

$\sigma_{\min}(A - (x + iy)I)$) as suggested by Section 2.3.3. We now turn our attention to the sensitivity of this problem, investigating the sensitivity of the critical points of $\sigma(x, y)$ to small perturbations in A . To first order, a perturbation δA in A will cause a perturbation in the gradient of $\sigma(x, y) = (\sigma_x, \sigma_y)$ at the point (x_0, y_0) , which we may call $(\delta\sigma_x, \delta\sigma_y)$. By inverting the Hessian (given by Sun's formulae) on this perturbation, one then has the first order correction to (x_0, y_0) , which we write as $(\delta x_0, \delta y_0)$.

Theorem 2.5.4 *Let (x_0, y_0) be a stationary point of $\sigma(x, y)$ and $\Phi, \Psi, r,$ and l be defined as above. Then let $p = \tilde{U}^H \delta A v$ and $q = \tilde{V}^H \delta A^H u$. Then*

$$\begin{aligned}\delta\sigma_x(x_0, y_0) &= \text{Real}\{r^H \Phi p + r^H \Psi q + p^H \Psi l + q^H \Phi l\} \\ \delta\sigma_y(x_0, y_0) &= \text{Imag}\{r^H \Phi p + r^H \Psi q + p^H \Psi l + q^H \Phi l\}.\end{aligned}$$

Proof: From Sun's theorem, we have

$$\sigma_x + i\sigma_y = -v^H u$$

and thus

$$\delta\sigma_x + i\delta\sigma_y = -\delta v^H u - v^H \delta u.$$

Since $u^H v = 0$, we may substitute $v = \tilde{U} \tilde{U}^H v = \tilde{U} r$ and $u = \tilde{V} \tilde{V}^H u = \tilde{V} l$,

$$\delta\sigma_x + i\delta\sigma_y = -r^H \tilde{U}^H \delta u - (\tilde{V}^H \delta v)^H l.$$

From Theorem 2.5.1 we have a formula for $\tilde{U}^H \delta v$ and $\tilde{V}^H \delta u$, which we substitute to give

$$\sigma_x + i\sigma_y = r^H \Phi p + r^H \Psi q + p^H \Psi l + q^H \Phi l.$$

□

From this lemma and Sun's theorem, we see that the condition number on the critical points of the pseudospectra is given by largest singular value of the $2 \times (4n - 4)$ real matrix

$$\begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{pmatrix}^{-1} \begin{pmatrix} \delta\sigma_x \\ \delta\sigma_y \end{pmatrix},$$

where $\delta\sigma_x$ and $\delta\sigma_y$ are the real linear functions of the real and imaginary parts of p and q defined by the lemma.

2.6 Nearest \hat{A} 's come from simple σ_{\min}

In Section 2.5.2 we derived expressions for the centripetal acceleration of a geodesic along \mathcal{D} . We will now use this machinery to give us the prove the claims made at the end of Section 2.4, namely, that the only critical points of singular values one need examine are those of σ_{\min} and not any other σ_k . Once again, the reader who wishes to skip over technicalities is advised to skip this section.

First we will make some fairly general derivations over co-dimension 1 manifolds. These derivations will relate the extrema of centripetal acceleration through a point to the character of that point as a minimizer of euclidean distance. Then we will apply these results to the specific problem of minimizing $\|A - \hat{A}\|$.

We first consider a general manifold M of co-dimension 1 in \mathbb{R}^n or \mathbb{C}^n . Suppose that $p \in M$ locally minimizes the function

$$D(p, q) = \frac{1}{2} \langle p - q, p - q \rangle = \frac{1}{2} d(p, q)^2 \tag{2.31}$$

for some point q in \mathbb{R}^n or \mathbb{C}^n . Let $p(t) \in M$ be a geodesic passing through p at $t = 0$ satisfying (2.25, 2.26) with unit velocity v ($\langle v, v \rangle = 1$). Differentiating (2.31) with respect to t , we have

$$\begin{aligned} \frac{d}{dt} D(p(t), q) &= \text{Real}\{\langle p - q, v \rangle\} \\ \frac{d^2}{dt^2} D(p(t), q) &= \text{Real}\{\langle v, v \rangle + \langle p - q, -\Gamma(v, v) \rangle\} \end{aligned}$$

$$\frac{d^2}{dt^2}D(p(t), q) = 1 - \text{Real}\{\langle p - q, \Gamma(v, v) \rangle\}.$$

Setting the first of these equations to zero at $t = 0$ for all v gives $p - q = \sigma \hat{n}$ where σ is a scalar ($|\sigma| = d(p, q)$) and \hat{n} is the unit normal of M at p . This gives

$$\frac{d^2}{dt^2}D(p(t), q) = 1 - \text{Real}\{\langle \sigma \hat{n}, \Gamma(v, v) \rangle\} \quad (2.32)$$

$$\frac{d^2}{dt^2}D(p(t), q) = 1 - \text{Real}\{\bar{\sigma} a(v, v)\}, \quad (2.33)$$

where we have substituted the Levi-Civita connection, $\Gamma(v, v) = a(v, v)\hat{n}$.

In order for p to be a stationary point of $D(p, q)$ it is necessary that $q = p + \sigma \hat{n}$. However, in order for this stationary point to be a local minimum, the second derivative must be nonnegative for all v . Since $a(v, v)$ is bilinear in v we may select the phase of v so that the condition (2.33) becomes

$$1 - |\sigma| |a(v, v)| \geq 0,$$

or

$$\frac{1}{\sigma} \geq a(v, v), \quad (2.34)$$

where we understand (2.34) as a comparison between magnitudes. Since (2.34) must be true for all unit v , and since the principal curvatures of M at p are given by the extrema of $|a(v, v)|$ over the set of all unit v , we have

Lemma 2.6.1 *If $d(p, q)$ is minimized by p then it is necessary that the focal lengths of M at p be greater than or equal to $d(p, q)$; i.e., if there is a focal point on the line segment $p + s\hat{n}$ ($0 \leq s \leq 1$) connecting p to q , p cannot minimize $d(p, q)$.*

This relation of focal points to the characterization of the stationary points of $d(p, q)$ can be illustrated clearly in two dimensions. In Figures 2-19 and 2-20, we use a hemispherical surface to illustrate these lemmatae.

We may now move to the specific case of minimizing $d(\hat{A}, A)$ for $\hat{A} \in \mathcal{D}$.

Lemma 2.6.2 *Let*

$$\hat{A} = \tilde{U} \tilde{\Sigma} \tilde{V}^H + \lambda I$$

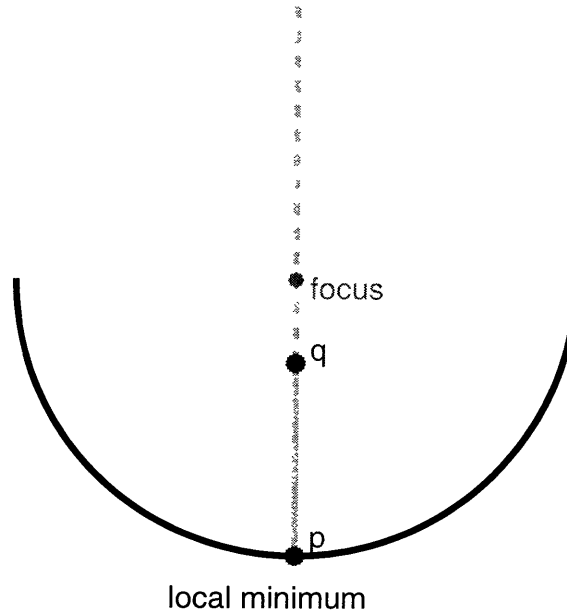


Figure 2-19: Below the focal point of a hemispherical surface, the distance minimizing p on the hemisphere to a given q is at the bottom of the surface.

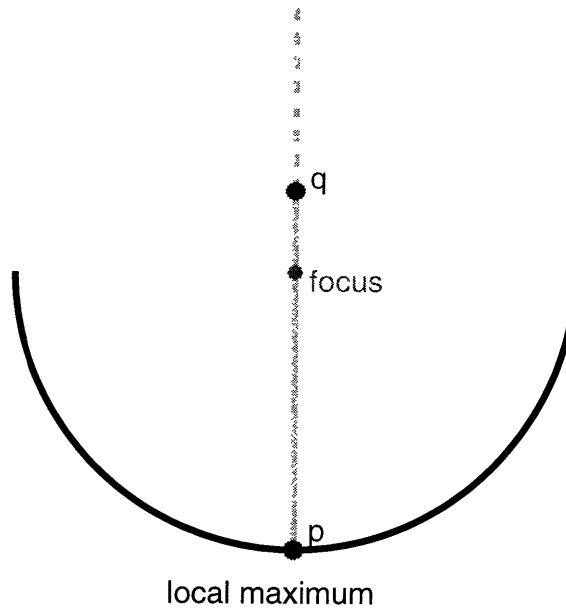


Figure 2-20: Above the focal point of the hemispherical surface, the bottom point of the hemisphere is now a maximum of distance (the nearest p on the hemisphere to q has become either of the end points of the hemisphere).

be an element of \mathcal{D} . Let $a(H, H)$ be the centripetal acceleration defined on the tangent vectors of \hat{A} by Equation (2.27). Then, taking maximums over the unit tangent ball ($\langle H, H \rangle = 1$), $\frac{1}{\sigma_{n-1}} \leq \max_H |a(H, H)|$, and, generically, $\frac{1}{\sigma_{n-1}} < \max_H |a(H, H)|$.

Proof: Using Equations (2.27, 2.28, 2.29) we write $a(H, H)$ in terms of a quadratic form

$$a(H, H) = z^T (Z + aa^T) z$$

$$\text{where } Z = \begin{pmatrix} -\tilde{\Sigma}^{-1} & 0 \\ 0 & \tilde{\Sigma}^{-1} \end{pmatrix} \text{ and } a = \frac{1}{2\sqrt{l^T \tilde{\Sigma}^{-1} r}} \begin{pmatrix} \tilde{\Sigma}^{-1}(r + \bar{l}) \\ \tilde{\Sigma}^{-1}(r - \bar{l}) \end{pmatrix}.$$

We may bound the maximum magnitude of this form from below with the 2×2 quadratic form formed from a submatrix

$$|z^T \left(\begin{pmatrix} -\frac{1}{\sigma_{n-1}} & 0 \\ 0 & \frac{1}{\sigma_{n-1}} \end{pmatrix} + aa^T \right) z| \leq \max_{z, \|z\|=1} |z^T (Z + aa^T) z|,$$

$$\text{where } a = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \frac{1}{2\sqrt{l^T \tilde{\Sigma}^{-1} r}} \begin{pmatrix} \frac{1}{\sigma_{n-1}}(r_{n-1} + \bar{l}_{n-1}) \\ \frac{1}{\sigma_{n-1}}(r_{n-1} - \bar{l}_{n-1}) \end{pmatrix}.$$

Let $z = \begin{pmatrix} i \sin(\theta) \\ \cos(\theta) \end{pmatrix}$ for some θ (we set $\theta = 0$ when working in \mathbb{R}). We have

$$z^T \left(\begin{pmatrix} -\frac{1}{\sigma_{n-1}} & 0 \\ 0 & \frac{1}{\sigma_{n-1}} \end{pmatrix} + aa^T \right) z = \frac{1}{\sigma_{n-1}} + (a_2 \cos(\theta) + ia_1 \sin(\theta))^2.$$

We see that if we select θ such that $\text{Real}\{(a_2 \cos(\theta) + ia_1 \sin(\theta))^2\} \geq 0$ then

$$\frac{1}{\sigma_{n-1}} \leq |z^T \left(\begin{pmatrix} -\frac{1}{\sigma_{n-1}} & 0 \\ 0 & \frac{1}{\sigma_{n-1}} \end{pmatrix} + aa^T \right) z| \leq \max_{z, \|z\|=1} |z^T (Z + aa^T) z|.$$

Generically, we can expect that we can select θ such that $\text{Real}\{(a_2 \cos(\theta) + ia_1 \sin(\theta))^2\} > 0$. Thus,

$$\frac{1}{\sigma_{n-1}} < |z^T \left(\begin{pmatrix} -\frac{1}{\sigma_{n-1}} & 0 \\ 0 & \frac{1}{\sigma_{n-1}} \end{pmatrix} + aa^T \right) z| \leq \max_{z, \|z\|=1} |z^T (Z + aa^T) z|,$$

and the inequality becomes strict. □

We are now in a position to prove

Lemma 2.6.3 *If λ_0 is a critical point of $\sigma_k(A - \lambda I)$ (with $\sigma_k(A - \lambda_0 I) > \sigma_n(A - \lambda_0 I)$) then (using singular vectors u and v) $\hat{A} = A - \sigma_k uv^H$ cannot locally minimize $\|A - \hat{A}\|$ over all $\hat{A} \in \mathcal{D}$.*

Proof: Since the smallest focal length (reciprocal principal curvature) of \hat{A} is less than σ_n , it is impossible for $d(\hat{A}, A) = \sigma_k > \sigma_n$ to be locally minimized. \square

We can also follow this sort of reasoning to establish that the points in the spectral portrait which are crossing points of singular values will usually not produce local minimizers of distance.

Corollary 2.6.4 *Generically, $\sigma_n(A - \lambda I)$ are simple in the neighborhood of λ_0 .*

Proof: Generically, if $\sigma_{n-1} = \sigma_n$ at $\lambda = \lambda_0$, then \hat{A} cannot be a local minimum of $d(\hat{A}, A)$. \square

2.7 Triple eigenvalues

The variety of matrices with triple eigenvalues, \mathcal{T} , is a subvariety of \mathcal{D} . It is the variety of cusp points of \mathcal{D} , the points where \mathcal{D} is not smooth. This might lead one to think that higher order derivatives of σ_{\min} might shed light on the distance to \mathcal{T} . This is not quite the case, however, though some bounds are possible. This section will investigate relations which can be made between \mathcal{T} and the focal points of \mathcal{D} .

It is easy to see that the intersection of \mathcal{D} with its evolute is \mathcal{T} . For $\hat{A} \in \mathcal{D}$ with repeated eigenvalue λ , we may take $x_1 = v$, $x_2 = \tilde{U}\tilde{\Sigma}^{-1}r$ as the right eigenvector and generalized eigenvector and $y_2 = u$, $y_1 = \tilde{V}\tilde{\Sigma}^{-1}l$ as the left eigenvector and generalized eigenvector, with

$$(y_1 \ y_2)^H (x_1 \ x_2) = l^H \tilde{\Sigma}^{-1} r \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Note that if $l^H \tilde{\Sigma}^{-1} r = 0$ then the left and right eigenvectors cannot be duals and thus there must be additional right and left generalized eigenvectors of eigenvalue λ . Let us now suppose

that \hat{A} has a focal length of 0 (thus \hat{A} is in the evolute of \mathcal{D}). Then, by the definitions of previous sections,

$$|r^H\Phi(0)r + l^H\Phi(0)l| = |2l^H\Psi(0)r|.$$

However, the LHS is zero. Thus we have $l^H\Psi(0)r = l^H\tilde{\Sigma}^{-1}r = 0$.

This is equivalent to the condition $\det \begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix} = 0$. Thus \mathcal{T} can be thought of as the locus of points satisfying

$$\begin{aligned} \sigma_{\min}(A - \lambda) &= 0 \\ \frac{d}{d\lambda}\sigma_{\min}(A - \lambda) &= 0 \end{aligned}$$

for some λ , with the additional constraint

$$\frac{d^2}{d\lambda^2}\sigma_{\min}(A - \lambda) = 0,$$

in contrast with the focal points which satisfy

$$\begin{aligned} \sigma_{\min}(A - \lambda) &= \sigma \\ \frac{d}{d\lambda}\sigma_{\min}(A - \lambda) &= 0 \\ \frac{d^2}{d\lambda^2}\sigma_{\min}(A - \lambda) &= 0. \end{aligned}$$

Unfortunately, while the first derivative of σ_{\min} is independent of the value of σ_{\min} , the second derivative is not, preventing us from being able to use this set of equations as a method for finding the nearest element of \mathcal{T} .

We can bound the distance to the nearest $\tilde{A} \in \mathcal{T}$ from A if $\hat{A} \in \mathcal{D}$ is ill-conditioned. It is easy to show that one can always find a \tilde{A} within $|\lambda - \lambda_i|(2l^H\tilde{\Sigma}^{-1}r)$ of \hat{A} , where λ_i is an eigenvalue of \hat{A} distinct from and closest to the double eigenvalue λ .

Chapter 3

The Computation of Nearest k -fold Eigenvalues

3.1 Introduction

A curious formula presented by Malyshev [49] has the property that it will, in some cases, determine the 2-norm distance of a matrix to the set of matrices with a given double eigenvalue. These results can be extended to a formula which gives the distance to the set of matrices with a given eigenvalue of any multiplicity.

There is a sense in which our extension (when successful) decomposes the given matrix A into $A = \hat{A} + \sigma N$ where \hat{A} is a matrix having $J_k(\lambda)$ structure for some given k and λ , N is a member of a set of transverse directions dependent only upon \hat{A} , and σ is a real number. We can view the set of N for a given \hat{A} as a set of normals to the set of all $J_k(\lambda)$ matrices, although such an identification is not rigorously possible without an inner product between matrices (which the 2-normed matrix space lacks).

The formula we present does not always produce the aforementioned decomposition. Examples can easily be generated in which our formula fails. While we have reason to believe that the nearness of A to the set of $J_k(\lambda)$ is an important factor, it is currently unclear under what conditions our formula is guaranteed to succeed.

3.2 Malyshev's formula for $\text{dist}(A, J_2)$

We let $\text{dist}(A, J_k(\lambda))$ denote the 2-norm distance from A to the set of matrices with a k -th order Jordan block for the eigenvalue λ . A tight estimate of this distance can be of great value in regularizing the numerical computation of the Jordan canonical form of a given matrix [40], as well as understanding the conditioning of the eigenvalues of A [67, 68].

Since $\text{dist}(A, J_k(\lambda)) = \text{dist}(A - \lambda, J_k(0))$, we need only consider nilpotent Jordan structures in what follows, setting $J_k = J_k(0)$. Malyshev, in [49], proposed the following formula for the 2-norm distance of a given matrix A to the set of matrices with a double 0 eigenvalue:

$$\text{dist}(A, J_2) = \max_{\gamma} \sigma_{2n-1} \begin{pmatrix} A & 0 \\ \gamma & A \end{pmatrix}, \quad (3.1)$$

where σ_{2n-1} is the second smallest singular value. In what follows, we see that this formula has a natural generalization to more complicated Jordan structures. We do not give a proof of the correctness of the generalized statement, and in fact it does not take long to randomly generate a failure. However, it may be that there is some appropriate generalization of (3.1) which will reliably give $\text{dist}(A, J_k)$ for any A .

3.3 Formula for $\text{dist}(A, J_k)$

Let A be an $n \times n$ matrix and B be $k \times k$. We consider matrices of the form $I \otimes A - B^T \otimes I$, where \otimes is the Kronecker product ($(X^T \otimes Y) \cdot \text{vec}(Z) = \text{vec}(YZX)$). In (3.1) A is the given matrix and B is the matrix $\begin{pmatrix} 0 & -\gamma \\ 0 & 0 \end{pmatrix}$ with γ indeterminate.

In general we will consider B to be a $k \times k$ strictly upper triangular matrix of indeterminants. Then we examine the generalization

$$\tilde{\text{dist}}(A, J_k) = \max_B \sigma(I \otimes A - B^T \otimes I), \quad (3.2)$$

where $\sigma = \sigma_{nk-k+1}$ (singular values in decreasing order). This equation, unfortunately, is not always true. However, sometimes it is.

Lemma 3.3.1 *Let \hat{A} have the eigenvalue 0 with multiplicity of at least k . Then*

$$\sigma_{nk-k+1}(I \otimes A - B^T \otimes I) = 0$$

for all $k \times k$ strictly upper triangular B .

Proof: Let X be a $(n \times k, \text{full rank})$ matrix of right eigenvectors and Y a $(n \times k, \text{full rank})$ matrix of left eigenvectors such that

$$\begin{aligned}\hat{A}X - XJ_k &= 0 \\ Y^H \hat{A} - J_k Y^H &= 0.\end{aligned}$$

We proceed by showing that the equation $\hat{A}V - VB = 0$ has at least k independent solutions. Assuming that B is generic (i.e. all elements on the first superdiagonal are nonzero), we have $B = R^{-1}J_k R$ for some upper triangular R and

$$\hat{A}V - VB,$$

for $V = XR$. Since X is full rank, V is full rank.

Let $V^{(j)} = VB^{j-1}$ (where $1 \leq j \leq k$) then

$$\hat{A}V^{(j)} - V^{(j)}B = 0$$

for all j . Since V is full rank and the superdiagonal elements of B are all nonzero, the vectors $\text{vec}(V^{(j)})$ are linearly independent solutions of the equation $(I \otimes \hat{A} - B^T \otimes I)v = 0$.

Since σ is a continuous function in B , it must vanish everywhere if it vanishes on generic B .

□

Lemma 3.3.2 $\sigma_{nk-k+1}(I \otimes A - B^T \otimes I)$ is maximized by some finite B .

Proof: It suffices to show that $\sigma_{nk-k+1}(I \otimes A - sB^T \otimes I) \rightarrow 0$ as $s \rightarrow \infty$ for all B such that $\|B\|_2 = 1$.

If A is nonsingular then we write

$$\sigma_{nk-k+1}(I \otimes A - sB^T \otimes I) = 1/\sigma_k((I \otimes A - sB^T \otimes I)^{-1}).$$

Let m be the largest power of B^T such that $\|(B^T)^m\| \neq 0$. We may substitute

$$(I \otimes A - sB^T \otimes I)^{-1} = (I \otimes A^{-1})(I \otimes I + sB^T \otimes A^{-1} + \dots + (sB^T \otimes A^{-1})^m),$$

to give (by perturbation theory)

$$\sigma_k((I \otimes A - sB^T \otimes I)^{-1}) \geq s^m \sigma_k((B^T)^m \otimes A^{-m-1}) - s^{m-1} \| (B^T)^{m-1} \otimes A^{-m} \|_2 - \dots - \| I \otimes A^{-1} \|.$$

Thus we see that as $s \rightarrow \infty$, $\sigma_k((I \otimes A - sB^T \otimes I)^{-1}) \rightarrow \infty$.

If A is singular then we may modify this argument slightly. For any $\epsilon > 0$, let A_ϵ be a nonsingular matrix such that $\|A - A_\epsilon\|_2 \leq \epsilon$. Then by the above argument there exists $s_0 > 0$ such that

$$\sigma_{nk-k+1}(I \otimes A_\epsilon - sB^T \otimes I) < \epsilon$$

for all $s \geq s_0$. Since

$$\sigma_{nk-k+1}(I \otimes A - sB^T \otimes I) \leq \sigma_{nk-k+1}(I \otimes A_\epsilon - sB^T \otimes I) + \epsilon,$$

giving

$$\sigma_{nk-k+1}(I \otimes A - sB^T \otimes I) \leq 2\epsilon.$$

for $s \geq s_0$. □

Theorem 3.3.3 *Let $\text{dist}(A, J_k)$ be the 2-norm distance from a matrix A to the set of matrices with 0 as an eigenvalue of multiplicity at least k . Then $\tilde{\text{dist}}(A, J_k) \leq \text{dist}(A, J_k)$.*

Proof: Let $A - \Delta = \hat{A}$ where \hat{A} has at least k zero eigenvalues. By lemma 3.3.1,

$$\sigma_{nk-k+1}(I \otimes \hat{A} - B^T \otimes I) = 0$$

for all B . Substituting $\hat{A} = A - \Delta$ we have

$$\sigma_{nk-k+1}(I \otimes A - I \otimes \Delta - B^T \otimes I) = 0,$$

and thus

$$\sigma_{nk-k+1}(I \otimes A - B^T \otimes I) < \|\Delta\|_2$$

for all B . Since $\|\Delta\|_2 = \text{dist}(A, J_k)$ we have $\tilde{\text{dist}}(A, J_k) = \text{dist}(A, J_k)$.

□

We see that $\tilde{\text{dist}}(A, J_k)$ is always an underestimate of $\text{dist}(A, J_k)$. If one could exhibit a Δ such that $A - \Delta = \hat{A}$ for some \hat{A} with $J_k(0)$ structure having the property $\|\Delta\|_2 = \tilde{\text{dist}}(A, J_k)$ then optimality would follow.

As we shall see, finding such a Δ is often possible.

3.4 Finding Δ

We assume that $\sigma_{nk-k+1}(I \otimes A - B^T \otimes I)$ is simple about the maximizing B . We also assume that the elements of the first superdiagonal of B are nonzero (i.e. B is nonderogatory). The reader should be warned that these are very questionable assumptions, and that in numerical experiments one can readily observe violations of both. When these assumptions are made, the construction of Δ is straightforward. When these assumptions are violated, it is unclear how to construct Δ .

If one forms $n \times k$ rectangular matrices from the vectors u and v by taking

$$U = (u_1 \quad u_2 \quad \cdots \quad u_k)$$

and

$$V = (v_1 \quad v_2 \quad \cdots \quad v_k),$$

where

$$u = \begin{pmatrix} u_1 \\ \vdots \\ u_k \end{pmatrix}$$

and

$$v = \begin{pmatrix} v_1 \\ \vdots \\ v_k \end{pmatrix},$$

then the definition of singular vectors implies the following identities:

$$\begin{aligned} AV - VB &= \sigma U \\ U^H A - BU^H &= \sigma V^H. \end{aligned}$$

One consequence of the above identities is that

$$\sigma(V^H V - U^H U) + (BU^H V - U^H V B) = 0. \quad (3.3)$$

Since B maximizes σ , σ is stationary with respect to variations in B . The variation of σ (when simple) with respect to B , by Sun's formula [60], is given by

$$\text{tr}(U^H V \delta B) = -\delta \sigma.$$

Assuming that σ is simple and is stationary with respect to variations in B , we will have

$$\text{tr}(U^H V \delta B) = 0.$$

Thus, $U^H V$ must be upper triangular. Since the first term in (3.3) is symmetric and the second term in (3.3) is strictly upper triangular, both terms must vanish, giving

$$U^H U = V^H V \quad (3.4)$$

$$BU^H V = U^H V B. \quad (3.5)$$

From (3.4) we have we have the following singular value decompositions for U and V :

$$\begin{aligned} U &= PSZ^H \\ V &= QSZ^H. \end{aligned}$$

(Where we have taken the “economy sized” SVD, i.e., all entries on the diagonal of S are nonzero, S is $p \times p$, and Z is $k \times p$ for $p \leq k$.) We may then construct a matrix $\Delta = PQ^H$ which has the properties

$$\begin{aligned} \Delta V &= U \\ U^H \Delta &= V^H. \end{aligned}$$

From these properties one can then construct a matrix $\hat{A} = A - \sigma \Delta$, having the properties

$$\begin{aligned} \hat{A}V - VB &= 0 \\ U^H \hat{A} - BU^H &= 0 \\ \|\hat{A} - A\| &= \sigma. \end{aligned}$$

What remains is to show that \hat{A} does have a 0 eigenvalue of multiplicity k .

Since B is nonderogatory, B is similar to a canonical Jordan block of size k by some upper triangular similarity transformation r .

$$r^{-1}Br = J_k = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

The r matrix is not uniquely determined. Any matrix of the form $r_2 = r_1 p(J_k)$, where $p(x)$ is any polynomial with nonvanishing constant term and $r_1 = r$ will also satisfy $Br_2 = r_2 J_k(0)$. We will therefore leave r_2 undetermined until later, when it will become clear how to exploit this freedom. For now, the reader should just keep in mind that the r_i are similarity transforms of B and that they are always upper triangular.

We can employ r_1 and r_2 to get

$$\begin{aligned}\hat{A}(Vr_1) - (Vr_1)J_k &= 0 \\ (r_2^{-1}U^H)\hat{A} - J_k(r_2^{-1}U^H) &= 0,\end{aligned}$$

or the canonical chain conditions

$$\hat{A}X - XJ_k = 0 \tag{3.6}$$

$$Y^H\hat{A} - J_kY^H = 0, \tag{3.7}$$

where $X = Vr_1$ and $Y = Ur_2^{-H}$.

We turn our attention to (3.5) which, in terms of X and Y , can be written as

$$r_2Y^HXr_1^{-1}B = Br_2Y^HXr_1^{-1}.$$

Since $B = r_1J_kr_1^{-1} = r_2J_kr_2^{-1}$, we have

$$Y^HXJ_k = J_kY^HX.$$

Thus Y^HX must be an upper triangular matrix with constant diagonals i.e. $Y^HX = q_0I + q_1J_k^1 + q_2J_k^2 + \dots + q_{k-1}J_k^{k-1}$. Since r_2 can be written as $r_2 = r_1p(J_k)$, we may fix r_2 such that $Y^HX = J_k^{k-m}$ where m is the rank of Y^HX . For the remainder of this section, we take r_2 to be fixed in this manner.

3.4.1 Generic success: $\text{tr}(U^HV) \neq 0$

Substituting from the definitions in Equations (3.6) and (3.7), we have

$$\text{tr}(U^HV) = \text{tr}(p(J_k)J_k^{k-m}).$$

Thus if $\text{tr}(U^HV) \neq 0$, we may infer that $m = k$ and $Y^HX = I$, and may interpret X and Y as the dual right and left eigenvectors of \hat{A} for a Jordan block J_k in the canonical decomposition

of \hat{A} .

3.4.2 Nongeneric success: $\text{tr}(U^H V) = 0$

Let

$$\text{tr}(p(J_k)J_k^{k-m}) = 0.$$

Since $p(J_k)$ is invertible, it must be that $m < k$. We will delay the proof that X and Y are linearly independent to first discuss the interpretation of this case.

Suppose $0 < m < k$. Then we must conclude that \hat{A} actually has Jordan blocks of size $J_{2k-m}(0)$ with right and left eigenvectors

$$\begin{aligned} P &= (p_1 \quad p_2 \quad \cdots \quad p_{2k-m}) \\ Q &= (q_1 \quad q_2 \quad \cdots \quad q_{2k-m}), \end{aligned}$$

where $p_i = x_i$ and $q_{k-m+i} = y_i$ for $1 \leq i \leq k$. This follows from a fairly obvious consideration of the canonical decomposition.

If $m = 0$ then we can say that \hat{A} has a Jordan block with eigenvalue 0 of order greater than or equal to $2k$. By taking repeated powers of a regularized \hat{A}^{-1} on the X and Y matrices one could determine the size of the blocks and fill in the rest of the eigenvectors.

Thus, when σ is stationary in the B variables and $B \sim J_k$, then \hat{A} must be a matrix that has Jordan structure $J_m(0)$ for some $m \geq k$.

These conclusions rest on X and Y being full rank, however. For the remainder of this section we will establish that they are (still assuming that B is nonderogatory).

Define $\text{rank}(X) = \text{rank}(V) = j_1$ and $\text{rank}(Y) = \text{rank}(U) = j_2$. By (3.4) $j_1 = j_2 = j$. Since the U and V matrices are actually the unit vectors, they must have at least rank 1. We now assume $0 < j < k$ and show that this leads to a contradiction.

By the chain conditions, one has

$$x_1 = \cdots = x_{k-j} = 0$$

and

$$y_{j+1} = \cdots = y_k = 0.$$

From the definitions of X and Y , we have

$$v_1 = \cdots = v_{k-j} = 0$$

and

$$u_{j+1} = \cdots = u_k = 0.$$

However, from (3.4), this also gives

$$u_1 = \cdots = u_{k-j} = 0$$

and

$$v_{j+1} = \cdots = v_k = 0.$$

Thus the rank of U and V must be $\max(0, j - (k - j))$ which cannot be satisfied for any j where $0 < j < k$, a contradiction.

3.5 The 2-norm “normals”

In the Frobenius norm, which is derived from the inner product $\langle A, B \rangle = \text{Real}\{\text{tr}(A^H B)\}$, it is possible to define the normals to the various repeated eigenvalue varieties at their generic points from the equations $\langle N, \hat{A}X - X\hat{A} \rangle = 0$ and $\langle N, x_i y_i^H \rangle = 0$ for each distinct eigenvalue λ_i of \hat{A} with right and left eigenvectors x_i and y_i . For a generic point \hat{A} on the variety of matrices with a $J_k(\lambda)$ block (λ fixed), this gives

$$N = Yp(J_k^T(0))X^H,$$

where $p(x)$ is a $k - 1$ degree polynomial in x , and X and Y are canonical, dual right and left eigenvector matrices satisfying

$$\begin{aligned}\hat{A}X - XJ_k &= 0 \\ Y^H\hat{A} - J_kY^H &= 0 \\ Y^HX &= I.\end{aligned}$$

From this it is clear that the set of normals to \hat{A} is a k dimensional vector space, complementing the $n^2 - k$ dimensional tangent space at \hat{A} , and we see that if the nearest J_k matrix to A is \hat{A} then

$$A = \hat{A} + Yp(J_k^T(0))X^H. \quad (3.8)$$

Additionally, we may recognize that if $p(x)$ has nonvanishing constant term, then it can actually be absorbed into the Y matrix, giving,

$$\begin{aligned}\hat{A}X - XJ_k &= 0 \\ Y^H\hat{A} - J_kY^H &= 0 \\ Y^HX &= p(J_k).\end{aligned}$$

Since the third equation is implied by all X and Y satisfying the chain conditions, we see that we may write any rank k normal as $N = YX^H$ where Y and X are any left and right eigenvector matrices satisfying the chain conditions. This is a subtle shift in point of view. The vectorial nature of the normal space has been obscured. Also, since $p(x)$ could have a vanishing constant term, we can only absorb $p(J_k^T(0))$ into Y by taking X and Y to be $n \times m$ matrices ($m = \text{rank of } p(J_k)$) satisfying reduced chain conditions,

$$\begin{aligned}\hat{A}X - XJ_m(0) &= 0 \\ Y^H\hat{A} - J_m(0)Y^H &= 0 \\ Y^HX &= P'(J_m(0)),\end{aligned}$$

where $P'(x) = x^{2(k-m)}p(x)$, allowing us to write any rank m normal as YX^H for matrices Y and X satisfying both reduced chain conditions and the additional requirements of the third equation.

Unfortunately, the matrix 2-norm does not arise from an inner product. It is therefore not necessarily possible to construct normals and investigate decompositions similar to (3.8). Yet, when the Malyshev method does give a simple maximum singular value with a nonderogatory B , there is a decomposition of the form

$$A = \hat{A} + \sigma N. \quad (3.9)$$

We then seek to understand how to construct, for a given generic \hat{A} with a J_k block, the set of all possible N such that \hat{A} is the nearest matrix constructed by the Malyshev method to $A = \hat{A} + \sigma N$ for some σ .

For a fixed \hat{A} assume that there exists an A such that $A = \hat{A} + \sigma N$, constructed as per the Malyshev method. We have $n \times k$ matrices, U and V satisfying

$$\begin{aligned} \hat{A}V - VB &= 0 \\ U^H\hat{A} - BU^H &= 0 \\ U^HU = V^HV &= R \end{aligned}$$

for some upper triangular, nonderogatory B , and the normal N given by $N = UR^{-1}V^H$. We may decompose $B = rJ_k r^{-1}$ obtaining for $X = Vr$ and $Y = Ur^{-H}$ the chain conditions on X and Y with $Y^HX = p(J_k)$ for some polynomial $p(x)$ assumed to have nonvanishing constant term.

Working this construction backwards, we see that given X and Y satisfying the chain conditions for \hat{A} , our task is to construct $Xr^{-1} = V$ and $Yr^H = U$ such that $U^HU = V^HV$. This r determines $B = rJ_k r^{-1}$ and $N = Y(Y^HY)^{-1}(r^Hr)^{-1}X^H N = Yr^Hr(X^HX)^{-1}X^H$. To find r we first solve for r^Hr and then perform Cholesky factorization. r^Hr must satisfy

$$(X^HX)(r^Hr)^{-1} = (r^Hr)(Y^HY). \quad (3.10)$$

If we let $X = Qu$ be the QR decomposition of X with upper triangular u and $Y = Pl$ be the QR decomposition of Y with lower triangular l , we have $r^H r = l^{-1}(lu^H ul^H)^{\frac{1}{2}} l^{-H}$ satisfying (3.10), from which we construct

$$N = P(lu^H ul^H)^{-\frac{1}{2}} lu^H Q^H$$

(note that $lu^H ul^H = (lu^H)(lu^H)^H$ is symmetric positive definite, allowing us to take the unique symmetric positive definite inverse square root). This cumbersome expression can be replaced with

$$N = PW^H Q^H,$$

where W is defined as the polar matrix in the polar decomposition $ul^H = WS$ ($S = (lu^H ul^H)^{\frac{1}{2}}$).

Since X and Y have not been assumed to be dual, we may replace Y with $Yp(J_k^T(0))$ to make this nonuniqueness manifest. This does not affect the P and Q matrices which result from the QR factorizations of Y and X , but modifies W by giving the equation

$$WS = up(J_k)l^H. \tag{3.11}$$

We see that W depends on the k parameters of $p(x)$. However, since W does not depend on the absolute scaling of the coefficients of $p(x)$, we see that the set of W is of dimension at most $k - 1$. The set of all N is the cone formed by $N = \sigma W$ for all σ , and thus is expected, in the generic case, to have k dimensions.

For comparison, we may rewrite the Frobenius normals as

$$N = PW^H Q^H,$$

where $W = up(J_k)l^H$. It is apparent that in some sense the 2-norm normals selected by Malyshev's method correspond to the Frobenius normals, but with a polar decomposition as an additional step, and it is that polar decomposition which destroys the vectorial nature of the normal space.

Lastly, one can show that the lower rank normals can be written as

$$N = PW^HQ^H,$$

where $WS = up(J_m(0))l^H$ and $Pl = Y$, $Qu = X$ are the left and right partial eigenvector matrices. In contrast to the Frobenius normals, the polar decomposition step in the 2-norm normals makes the normals of different rank in the 2-norm case separate from the normals of lower rank, i.e. they are not the limit points of the set of normals of higher rank.

3.6 A possible rationale for the 2-norm normals

When one has an inner product, one can derive the condition for normality by $\|N + T\|_F$ having a local minimum at $T = 0$, where T is any first order variation in a direction tangent to \hat{A} . Expanding the norm in terms of the inner product and differentiating gives the familiar formula

$$\langle N, T \rangle = 0.$$

Although we cannot define normals with the inner product against tangent vectors, we can, however, adopt the condition of $\|N + T\|_2$ having a local minimum at $T = 0$ and see that the set of N defined in the previous section satisfies this requirement. For rank k normals, we may take $W = ul^H$. Substituting, we see that we must examine $\|PW^HQ^H + T\|$ for a local minimum at $T = 0$ (where we have suppressed the subscript on the norm). We can write $T = PT_{11}Q^H + PT_{12}(Q^\perp)^H + P^\perp T_{21}Q^H + P^\perp T_{22}(Q^\perp)^H$. By the unitary invariance of the 2-norm, we have

$$\|PW^HQ^H + T\| = \|W^H + T_{11}\|,$$

where the other parts of T , being infinitesimal do not contribute to the norm.

The tangent space of \hat{A} is independent of the norm being employed. Thus, since every tangent of \hat{A} must satisfy $\langle T, Yp(J_k^T(0))X^H \rangle = 0$ for all $p(x)$. Thus, we have

$$\langle P^HTQ, lp(J_k^T(0))u^H \rangle = \text{Real}\{T_{11}up(J_k)l^H\} = 0. \quad (3.12)$$

Using unitary invariance again, we have

$$\|W^H + T_{11}\| = \|I + T_{11}W\|.$$

From the perturbation theorem of singular values, the singular values of I will perturb to first order by the eigenvalues of the symmetric part of $T_{11}W$. Thus, the 2-norm is guaranteed to increase if the symmetric part of $T_{11}W$ does not have all positive or all negative eigenvalues.

Consider the equation

$$\frac{1}{2}x^H(T_{11}W + W^HT_{11}^H)x = \text{Real}\{x^HT_{11}Wx\} = 0. \quad (3.13)$$

(3.13) is satisfied by some $x \neq 0$ if and only if the symmetric part of $T_{11}W$ does not have all positive or all negative eigenvalues. Expanding $W = ul^HS^{-1}$ and letting $y = S^{-\frac{1}{2}}x$, (3.13) is equivalent to finding a $y \neq 0$ such that

$$\text{Real}\{y^HS^{\frac{1}{2}}T_{11}ul^HS^{-\frac{1}{2}}y\} = 0. \quad (3.14)$$

But, we have

$$\text{Real}\{\text{tr}S^{\frac{1}{2}}T_{11}ul^HS^{-\frac{1}{2}}\} = \text{Real}\{\text{tr}T_{11}ul^H\} = 0,$$

by (3.12). Thus, the sum of the eigenvalues of the symmetric part of $S^{\frac{1}{2}}T_{11}ul^HS^{-\frac{1}{2}}$ vanishes, so we may find y satisfying (3.14), and then an x satisfying (3.13). We see, that $\|I + T_{11}W\| \geq 1$ for all tangent vectors to first order.

In conclusion, we see that the perturbations selected by the Malyshev method are all 2-norm minimizers. Thus, they can be considered “normal” even though the 2-norm does not have an associated inner product. We also see that these normals are determined only by properties of \hat{A} . In this sense we say that the normal bundle to \hat{A} is given by $Yl^{-1}W^Hu^{-H}X^H$ where u and l are the upper triangular and lower triangular matrices of QR(QL) decompositions of X and Y , and W is the polar part of $up(J_k)l^H$.

3.7 Concluding remarks

Investigation of the properties of formula (3.2) has not yet revealed the conditions for being able to construct \hat{A} after maximizing σ_{nk-k+1} . One can show for any \hat{A} with $J_k(\lambda)$ structure and normal N , that (3.2) recovers \hat{A} from $A = \hat{A} + \sigma N$ when

$$\sigma < \frac{1}{2}\sigma_{nk-k}(I \otimes \hat{A} - B^T \otimes I),$$

where $B = rJ_k(0)r^{-1}$, r defined by Equation (3.10). This leads us to expect that (3.2) may successfully locate the nearest $J_k(\lambda)$ matrix if A is close enough to the set of $J_k(\lambda)$ matrices.

It has also been observed experimentally that given \hat{A} and a normal N , for small enough σ one can reconstruct \hat{A} from $\hat{A} + \sigma N$ while for large enough σ one fails reconstruct \hat{A} . Further work will be required to see under what the the exact condition on σ and \hat{A} such that \hat{A} is recoverable from $A = \hat{A} + \sigma N$.

It is the belief of this author, that investigations along the various normal lines of \hat{A} will ultimately reveal the reasons for failure or success of the generalized Malyshev method proposed here.

Chapter 4

Solving General Nonlinear Eigenvalue Problems

(At this time, a copy of the matlab software mentioned in this draft can be found in <http://www.mit.edu/people/~ripper/Template/template.html>.)

4.1 Introduction

This section increases the scope of the algebraic eigenvalue problem by focusing on the geometrical properties of the eigenspaces. We discuss a template that solves variational problems defined on sets of subspaces. Areas where such problems arise include electronic structures computation, eigenvalue regularization, control theory, signal processing and graphics camera calibration. The underlying geometry also provides the numerical analyst theoretical insight into the common mathematical structure underlying eigenvalue algorithms (see [1, 31, 56]).

Suppose that one wishes to optimize a real valued function $F(Y)$ over Y such that $Y^*Y = I$, where Y^* is either transpose or Hermitian transpose as appropriate. Our template is designed as a means to optimize such an F .

One simple case is optimization over square orthogonal (or unitary) matrices such as the least squares simultaneous diagonalization of symmetric matrices (known as INDSCAL [21]) problem described later. Another simple case is optimization over the unit sphere, as in

symmetric Rayleigh quotient minimization. In between, we have rectangular $n \times p$ ($n \geq p$) matrices Y with orthonormal columns such as the orthogonal Procrustes problem.

Furthermore, some functions F may have the symmetry property $F(Y) = F(YQ)$ for all orthogonal (unitary) Q with a specified block diagonal structure, which causes some search directions to have no effect on the value of F . For example, if A is a symmetric matrix, $n \geq p$, and $F(Y) = \text{tr}(Y^*AY)$, then $F(Y) = F(YQ)$ for all $p \times p$ orthogonal Q . More generally, some functions F have the property that $F(Y) = F(YQ)$ where Q is orthogonal (unitary) with the **block diagonal** form

$$Q = \begin{pmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & \cdots & 0 \\ 0 & 0 & \cdots & Q_p \end{pmatrix},$$

where the Q_i are orthogonal (unitary) matrices. More complicated problems with block diagonal orthogonal (unitary) symmetry can arise in eigenvalue regularization problems (an example of such a problem is the sample nearest Jordan block problem solved in the examples).

This chapter is organized as follows. Section 4.2 will discuss the basics of calling the template code. Section 4.3 discusses the objective functions and their derivatives for the example problems explored in this chapter. Section 4.4 contains sample runs of instances of the examples problems. Section 4.5 explains the code structure and the places where a user may wish to make modifications. Section 4.6 will cover some of the basic mathematics concerning the geometry of the Stiefel manifold.

4.2 Calling `sg_min`

The matlab templates are ready for immediate use or as a departure point for generalization e.g. problems over multiple variables with orthogonality constraints, or code optimizations. In the simplest mode, the user need only supply a function to minimize $F(Y)$ (and first and second derivatives, optionally) in `F.m` (in `dF.m` and `ddF.m`) and an initial guess `Y0`. The calling sequence is then a single call to `sg_min` (named in honor of Stiefel and Grassmann).

```
[fopt, yopt] = sg_min(Y0) .
```

For example, if the function `F.m` has the form

```
function f=F(Y)
f=trace( Y' * diag(1:10) * Y * diag(1:3) );
```

we can call `sg_min(rand(10,3))` which specifies a random starting point.

We strongly recommend also providing first derivative information:

```
function df=dF(Y)
df = 2 * diag(1:10) * Y * diag(1:3);
```

The code can do finite differences, but it is very slow and problematic. Second derivative information can also be provided by the user (this is not nearly as crucial for speed as providing first derivative information, but may improve accuracy):

```
function ddf=ddF(Y,H)
ddf = 2 * diag(1:10) * H * diag(1:3);
```

A sample test call where $F(Y)$ is known to have optimal value 10 is

```
>> rand('state',0); % initialize random number generator
>> fmin = sg_min(rand(10,3))
```

iter	grad	F(Y)	flops	step type
0	1.877773e+01	3.132748e+01	2192	none
	invdgrad: Hessian not positive definite, CG terminating early			
1	1.342892e+01	2.011465e+01	146438	newton step
	invdgrad: Hessian not positive definite, CG terminating early			
2	1.130914e+01	1.368046e+01	290568	newton step
	invdgrad: Hessian not positive definite, CG terminating early			
3	5.974785e+00	1.063044e+01	434822	newton step
4	1.135397e+00	1.006835e+01	681414	newton step
5	5.526059e-02	1.000009e+01	929492	newton step

argument	description
rc	{'real', 'complex' }
mode	{'frcg', 'newton' }
metric	{'flat', 'euclidean', 'canonical' }
motion	{'approximate', 'exact' }
verbose	{'verbose', 'quiet' }
gradtol	first convergence tolerance
ftol	second convergence tolerance
partition	partition of p indicating symmetries of f

Table 4.1: A short list of the optional arguments for `sg_min`

```
6      5.087895e-05    1.000000e+01    1203160    newton step
7      1.706261e-05    1.000000e+01    1518165    newton step
```

```
fmin =
10.0000
```

The full calling sequence to `sg_min` is

```
[fopt, yopt]=sg_min(Y0,rc,mode,metric,motion,verbose,gradtol,ftol,partition),
```

where `Y0` is required and the optional arguments are (see Table 4.1):

- **rc** The `rc` argument specifies whether the matrices will have complex entries or not. Although most of the code is insensitive to this issue, `rc` is vital for counting the dimension of the problem correctly. When omitted, `sg_min` guesses based on whether `Y0` has nonzero imaginary part.
- **mode** The `mode` argument selects the optimization method will be used. `'frcg'` selects Fletcher-Reeves conjugate gradient (popular in the electronic structures community), and `'newton'` selects Newton's method with a conjugate gradient Hessian inverter (following steepest descent directions whenever the steepest descent line minimum is less than the Newton line minimum). While `'newton'` is the default, the user may wish to try the `'frcg'`, which can be less expensive computationally.

- **metric** The `metric` argument selects the kind of geometry with which to endow the constraint surface. This ultimately affects the definition of the covariant Hessian. 'flat' projects the result of applying the unconstrained Hessian onto the tangent space, while 'euclidean' and 'canonical' add on *connection* terms specific to their geometries. 'euclidean' is the default.
- **motion** The `motion` argument selects whether line movement along the manifold will be by the analytic solution to the geodesic equations of motions for the metric selected, or by a computationally less expensive approximation to the solution (default). (For a 'flat' metric, there is no geodesic equation, so this argument has no effect in that case.)
- **verbose** The `verbose` argument determines whether the function will display reports on each iteration while the function executes. When this argument is 'verbose' (the default) data will be displayed and also recorded in the global `SGdata`. When this argument is 'quiet' then no convergence data is displayed or recorded.
- **gradtol** and **ftol** We declare convergence if either of two conditions are true:
 $\text{grad}/\text{gradinit} < \text{gradtol}$ (default $1e-6$) or $(f\text{-fold})/f < \text{ftol}$ (default $1e-8$),
where `gradinit` is the initial magnitude of the gradient and `fold` is the value of $F(Y)$ at the last iteration.
- **partition** The `partition` is a cell array whose elements are vectors of indices that represent a disjoint partition of $1:p$. If F has no symmetry, then the partition is `num2cell(1:p)`. If $F(Y) = F(YQ)$ for all orthogonal Q , then the partition is $\{1:p\}$. The partition is $\{1:2, 3:5\}$ if the symmetry in F is $F(Y) = F(YQ)$ for orthogonal Q with sparsity structure

$$\begin{pmatrix} \times & \times & & & & \\ \times & \times & & & & \\ & & \times & \times & \times & \\ & & \times & \times & \times & \\ & & \times & \times & \times & \end{pmatrix}.$$

The user could equally well specify $\{3:5, 1:2\}$ or $\{[5\ 3\ 4], [2\ 1]\}$ for the partition, i.e. a partition is a set of sets. The purpose of the partition is to project away the null-space of the Hessian resulting from any block diagonal orthogonal symmetries of $F(Y)$. If the argument is omitted, our code will pick a partition by determining the symmetries of F (using its `partition.m` function). However, the user should be aware that a wrong partition can destroy convergence.

4.3 Sample problems and their differentials

This section serves as a sample guide on the manipulation of objective functions of matrices with orthonormal columns. We have found a few common tricks worth emphasizing.

Once one has a formula for the objective function $F(Y)$, we define the formula for $dF(Y)$ implicitly by $\text{tr}(V^*dF(Y)) = \frac{d}{dt}F(Y(t))|_{t=0}$ where $Y(t) = Y + tV$ (or any curve $Y(t)$ for which $\dot{Y}(0) = V$). The reader may recall that $\text{tr}(A^*B) = \sum_{i,j} A_{ij}B_{ij}$, so it functions just like the real inner product for vectors¹ and the implicit definition of $dF(Y)$ is actually the directional derivative interpretation of the gradient of $F(Y)$ as an unconstrained function in a Euclidean space.

For most of the functions we have used in our examples, the easiest way to obtain the formula for dF is to actually use the implicit definition.

For example, if $F(Y) = \text{tr}(AY^*BY)$ one then has

$$\text{tr}(V^*dF(Y)) = \text{tr}(AV^*BY + AY^*BV).$$

Since the value of the trace is invariant under cyclic permutations of products and transposes, we may rewrite this equation as

$$\text{tr}(V^*dF(Y)) = \text{tr}(V^*BYA + V^*B^*YA^*),$$

and, since V is unrestricted, this implies that $dF(Y) = BYA + B^*Y^*A^*$.

¹over complex numbers, we always take the real part of $\text{tr}(A^*B)$.

The process we recommend is:

- try to write $F(Y)$ as a trace
- compute $\frac{d}{dt}F(Y(t))|_{t=0}$ where we let $V = \dot{Y}(t)$
- use trace identities to rewrite every term to have a V^* in the front
- strip off the V^* leaving the $dF(Y)$

As a check, we recommend using the finite difference `dF.m` code supplied in the subdirectory `finitediff` to check derivations before proceeding.

The software needs a function called `ddF.m` which returns $\frac{d}{dt}dF(Y(t))|_{t=0}$ for $\dot{Y}(0) = H$. The sort of second derivative information required by the software is easier to derive than the first. If one has an analytic expression for $dF(Y)$, then one need only differentiate.

If, for some reason, the computation for `ddF.m` costs much more than two evaluations of $dF(Y)$ with `dF.m`, the reader may just consider employing the finite difference function for `ddF.m` found in `finitediff` (or simply use `ddF.m` as a check).

It is, however, strongly suggested that one use an analytic expression for computing $dF(Y)$, as the finite difference code for it requires a large number of function evaluations ($2np$).

4.3.1 The Procrustes problem

The Procrustes problem (see [32]) is the minimization of $\|AY - YB\|_F$ for constant A and B over the manifold $Y^*Y = I$. This minimization determines the nearest matrix \hat{A} to A for which

$$Q^* \hat{A} Q = \begin{pmatrix} B & * \\ 0 & * \end{pmatrix},$$

i.e. the columns of B span an invariant subspace of \hat{A} .

The differential of $F(Y) = \frac{1}{2}\|AY - YB\|_F^2 = \frac{1}{2}\text{tr}(AY - YB)^*(AY - YB)$ is given by

$$dF(Y) = A^*(AY - YB) - (AY - YB)B^*.$$

This can be derived following the process outlined above. Observe that

$$\begin{aligned}
\frac{d}{dt}F(Y(t))|_{t=0} &= \frac{1}{2}\text{tr}((AV - VB)^*(AY - YB) + (AY - YB)^*(AV - VB)) \\
&= \text{tr}((AV - VB)^*(AY - YB)) \\
&= \text{tr}(V^*(A^*(AY - YB)) - V^*(AY - YB)B^*).
\end{aligned}$$

The second derivative of $F(Y)$ is given by the equation,

$$\frac{d}{dt}dF(Y(t))|_{t=0} = A^*(AH - HB) - (AH - HB)B^*,$$

where $\dot{Y}(0) = H$, which can be obtained by varying the expression for dF .

4.3.2 Nearest Jordan structure

Now suppose that the B block in the Procrustes problem is allowed to vary with Y . Moreover, suppose that $B(Y)$ is in the nearest staircase form to Y^*AY , that is:

$$B(Y) = \begin{pmatrix} \lambda_1 I & * & * \\ 0 & \lambda_2 I & * \\ 0 & 0 & \dots \end{pmatrix}$$

for fixed block sizes, where the *-elements are the corresponding matrix elements of Y^*AY and the $\lambda_i I$ blocks are either fixed or determined by some heuristic, e.g. taking the average trace of the blocks they replace in Y^*AY . Then a minimization of $\|AY - YB(Y)\|_F$ finds the nearest matrix a a particular Jordan structure, where the structure is determined by the block sizes, and the eigenvalues are λ_i . When the λ_i are fixed, we call this the *orbit* problem, and when the λ_i are selected by the heuristic given we call this the *bundle* problem.

Such a problem can be useful in regularizing the computation of Jordan structures of matrices with ill-conditioned eigenvalues.

The form of the differential of $F(Y)$, surprisingly, is the same as that of $F(Y)$ for the

procrustes problem.

$$dF(Y) = A^*(AY - YB(Y)) - (AY - YB(Y))B(Y)^*.$$

This is because $\text{tr}(-(AY - YB(Y))^*Y\dot{B}) = \text{tr}((B(Y) - Y^*AY)^*\dot{B}) = 0$ for the B selected as above for either orbit or bundle case, where $\dot{B} = \frac{d}{dt}B(Y(t))|_{t=0}$.

In contrast, the form of the second derivatives is a bit more complicated, since the B now depend on Y .

$$\frac{d}{dt}dF(Y(t))|_{t=0} = \frac{d}{dt}dF_{Proc}(Y(t))|_{t=0} - A^*Y\dot{B} - AY\dot{B}^* + YB\dot{B}^* + Y\dot{B}B^*,$$

where $\dot{Y}(0) = H$, dF_{Proc} is just short for the Procrustes (B constant) part of the second derivative, and $\dot{B} = \frac{d}{dt}B(Y(t))|_{t=0}$ which is the staircase part of $Y^*AH + H^*AY$ (with trace averages or zeros on the diagonal depending on whether bundle or orbit).

4.3.3 Trace minimization

In this case we consider a ‘‘Rayleigh quotient’’ style of iteration to find the eigenspaces of the smallest eigenvalues of a symmetric positive definite matrix A . That is, we can minimize $F(Y) = \frac{1}{2}\text{tr}(Y^*AY)$ [2, 36, 54]. The minimizer Y^* will be an $n \times p$ matrix whose columns span the eigenspaces of the lowest p eigenvalues of A .

For this problem

$$dF(Y) = AY.$$

It is easily seen that

$$\frac{d}{dt}dF(Y(t))|_{t=0} = AH,$$

where $\dot{Y}(0) = H$.

4.3.4 Trace minimization with a nonlinear term

We consider a related nonlinear eigenvalue problem, to the trace minimization of section 3.3, minimize $F(Y) = \frac{1}{2}(\text{tr}(Y^*AY) + g(Y))$, where $g(Y)$ is some nonlinear term.

This sort of minimization occurs in electronic structures computations such as Local Density Approximations (LDA) where the columns of Y represent electron wavefunctions, A is a fixed Hamiltonian, and $g(Y)$ represents the energy of the electron-electron interactions (see, for example, [4, 57]). In this case, the optimal Y represents the state of lowest energy of the system.

The only additions to the differentials of the trace minimization are terms from $g(Y)$ which vary from problem to problem. For our example, we have taken $g(Y) = \frac{K}{4} \sum_i \rho_i^2$ for some coupling constant K , where $\rho_i = \sum_j |Y_{ij}|^2$ (the charge density).

In this case

$$dF(Y) = dF_{\text{tracemin}} + K \text{diag}(\rho)Y,$$

and

$$\frac{d}{dt}dF(Y(t))|_{t=0} = \frac{d}{dt}dF_{\text{tracemin}}(Y(t))|_{t=0} + K \text{diag}(\rho)H + K \text{diag}\left(\frac{d}{dt}\rho\right)Y,$$

where $\dot{Y}(0) = H$, and $\frac{d}{dt}\rho_i|_{t=0} = \sum_j Y_{ij}H_{ij}$.

4.3.5 Simultaneous Schur decomposition problem

Consider two matrices A and B which in the absence of error have the same Schur vectors, i.e. there is a $Y \in O(n)$ such that Y^*AY and Y^*BY are both block upper triangular. Now suppose that A and B are somewhat noisy, from measurement errors, or some other kind of lossy filtering. In that case the Y that upper triangularizes A might not upper triangularize B as well. How does one find the best Y ?

This is a problem that was presented to us by W. Schilders [55] who phrased it as a least squares minimization of $F(Y) = \frac{1}{2}(\|\text{low}(Y^*AY)\|_F^2 + \|\text{low}(Y^*BY)\|_F^2)$, where $\text{low}(M)$ is a mask returning the block lower triangular part of M , where M is broken up into 2×2 blocks.

For this problem the differential is a bit tricky and its derivation instructive,

$$\begin{aligned} \text{tr}(V^*dF(Y)) &= \text{tr}(\text{low}(Y^*AY)^*\text{low}(Y^*AV + V^*AY) + \text{low}(Y^*BY)^*\text{low}(Y^*BV + V^*BY)) \\ \text{tr}(V^*dF(Y)) &= \text{tr}(\text{low}(Y^*AY)^*(Y^*AV + V^*AY) + \text{low}(Y^*BY)^*(Y^*BV + V^*BY)) \\ \text{tr}(V^*dF(Y)) &= \text{tr}(V^*(AY\text{low}(Y^*AY)^* + A^*Y\text{low}(Y^*AY) + BY\text{low}(Y^*BY)^* + B^*Y\text{low}(Y^*BY))) \end{aligned}$$

$$dF(Y) = AY\text{low}(Y^*AY)^* + A^*Y\text{low}(Y^*AY) + BY\text{low}(Y^*BY)^* + B^*Y\text{low}(Y^*BY)$$

where the second equation results from observing that $\text{tr}(\text{low}(M)^*\text{low}(N)) = \text{tr}(\text{low}(M)^*N)$, and the third from properties of the trace.

With second derivatives given by,

$$\begin{aligned} \frac{d}{dt}dF(Y(t))|_{t=0} &= AH\text{low}(Y^*AY)^* + A^*H\text{low}(Y^*AY) + AY\text{low}\left(\frac{d(Y^*AY)}{dt}\right)^* \\ &\quad + A^*Y\text{low}\left(\frac{d(Y^*AY)}{dt}\right) + BH\text{low}(Y^*BY)^* + B^*H\text{low}(Y^*BY) \\ &\quad + BY\text{low}\left(\frac{d(Y^*BY)}{dt}\right)^* + B^*Y\text{low}\left(\frac{d(Y^*BY)}{dt}\right), \end{aligned}$$

where $\dot{Y}(0) = H$, $\frac{d(Y^*AY)}{dt}|_{t=0} = H^*AY + Y^*AH$, and $\frac{d(Y^*BY)}{dt}|_{t=0} = H^*BY + Y^*BH$.

4.3.6 INDSCAL

Another problem like simultaneous Schur problem, involving sets of matrices with similar structures, is the problem of finding the best set of eigenvectors for a set of symmetric matrices given that the matrices are known to be simultaneously diagonalizable, but may have significant errors in their entries from noise or measurement error. Instances of this problem arise in the Psychometric literature, where it is called an INDSCAL problem [21].

Phrased in terms of a minimization, one has a set of symmetric matrices A_i and wishes to find $Y \in O(n)$ that minimizes $F(Y) = \frac{1}{2} \sum_i \|\text{off}(Y^*A_iY)\|_F^2$, where $\text{off}(M)$ returns the off-diagonal entries of M .

Similar to Schur problem (though with symmetric matrices and a somewhat different matrix component) we have

$$dF(Y) = \sum_i 2Y A_i \text{off}(Y^* A_i Y),$$

and

$$\frac{d}{dt}dF(Y(t))|_{t=0} = \sum_i 2H A_i \text{off}(Y^* A_i Y), \sum_i 2Y A_i \text{off}(H^* A_i Y + Y^* A_i H),$$

where $\dot{Y}(0) = H$.

4.4 Running our examples

We have provided implementations for the sample problems described.

We present the examples and their verbose outputs. We have included these outputs so the reader can check that his/her copy of the package is executing properly. The user should be running matlab (version 5 or compatible) in the following directory:

```
>> !ls
Fline.m          dimension.m      invdgrad.m      partition.m     tangent.m
README          dtangent.m      ip.m            sg_frcg.m
connection.m     finitediff      move.m          sg_min.m
dgrad.m         grad.m          nosym.m         sg_newton.m
```

(note, there may also be the additional subdirectories `example/`, `docs/`, or `@cell/`).

Each example problem has a subdirectory in the `examples` subdirectory.

```
>> !ls examples
indscal    jordan      ldatoy      procrustes    simschur      tracemin
```

Each of these subdirectories contains an implementation for $F(Y)$ ($dF(Y)$ and $ddF(Y, H)$), a `parameters` function to set a global, `FParameters`, of fixed parameters of $F(Y)$ (this function must be called before any other), a `guess` function to generate an initial guess, and possibly some auxiliary functions for computing $F(Y)$, related quantities, or instances of a specific problem.

```
>> !ls examples/ldatoy
F.m          dF.m          guess.m
Kinetic.m    ddF.m         parameters.m
>> !ls examples/jordan
Block.m      dBlock.m      ddF.m         guess.m        post.m
F.m          dF.m          frank.m       parameters.m
```


We executed the examples in the two supported optimization modes ('frcg', and 'newton') and either 'flat' or 'euclidean'. For different instances of the same problem, different modes might perform best, so the reader should not feel that a particular mode will exhibit superior performance in all instances.

4.4.1 A sample Procrustes problem

In this example, we attempt to find a Y which minimizes $\|AY - YB\|_F$ for a pair of randomly selected A (12×12) and B (4×4).

```
>> !cp examples/procrustes/*.m .
>> randn('state',0);
>> [A,B] = randprob;
>> parameters(A,B);
>> Y0 = guess;
>> [fn,Yn] = sg_min(Y0,'newton','euclidean');
```

iter	grad	F(Y)	flops	step type
0	1.967151e+01	3.145130e+01	4365	none
	invdgrad: Hessian not positive definite, CG terminating early			
1	1.201377e+01	1.334642e+01	299347	newton step
	invdgrad: Hessian not positive definite, CG terminating early			
2	6.998543e+00	7.885178e+00	549026	newton step
	invdgrad: Hessian not positive definite, CG terminating early			
3	6.754473e+00	4.082520e+00	828756	newton step
	invdgrad: Hessian not positive definite, CG terminating early			
4	7.037159e+00	2.156004e+00	1143569	newton step
	invdgrad: Hessian not positive definite, CG terminating early			
5	2.716002e+00	1.264874e+00	1481879	steepest step
	invdgrad: Hessian not positive definite, CG terminating early			
6	2.611537e+00	7.828101e-01	1779148	newton step
7	4.749149e-01	3.540531e-01	2339394	newton step

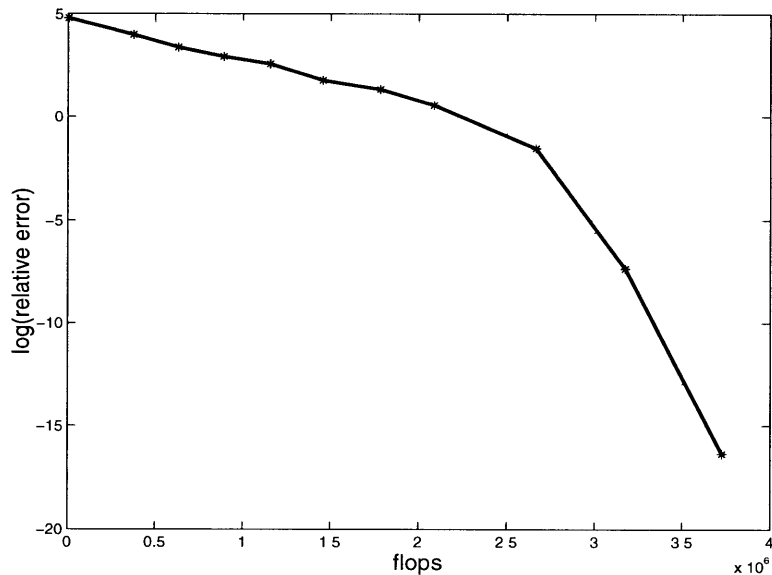


Figure 4-1: Procrustes Problem

8	2.098621e-02	3.361052e-01	2858664	newton step
9	7.350907e-05	3.360631e-01	3426059	newton step
10	1.832068e-05	3.360631e-01	3991138	newton step

Figure 1 shows the convergence curve for this run.

4.4.2 A sample Jordan block problem

We now look for a nearby matrix with a Jordan structure $J_4(\alpha) \oplus J_2(\beta) \oplus$ (don't care) to the 12×12 Frank matrix, a matrix whose Jordan structure is known to be very difficult.

```
>> !cp examples/jordan/*.m .
>> A = frank(12);
>> evs = sort(eig(A)); eigvs = [mean(evs(1:4)) mean(evs(5:6))];
>> parameters(A,eigvs,[4; 2],'bundle')
>> Y0 = guess;
>> [fn,Yn] = sg_min(Y0,'newton','euclidean');
iter   grad           F(Y)           flops           step type
```

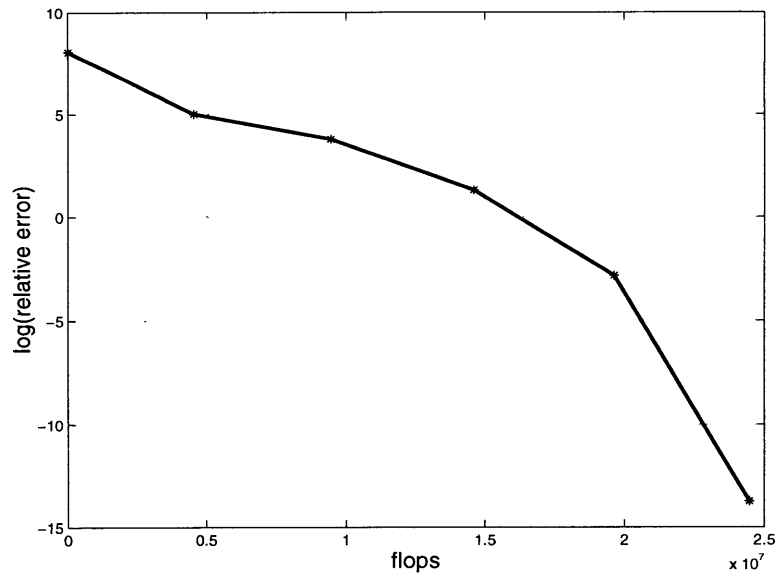


Figure 4-2: Jordan Problem

0	1.775900e-02	9.122159e-06	15535	none
	invdgrad: Hessian not positive definite, CG terminating early			
1	1.797005e-03	4.382300e-07	4539120	newton step
	invdgrad: max iterations reached inverting the hessian by CG			
2	1.615236e-03	1.320791e-07	9451244	newton step
	invdgrad: max iterations reached inverting the hessian by CG			
3	2.100793e-04	1.364935e-08	14602068	newton step
4	7.793143e-05	3.098557e-09	19631237	newton step
5	3.826000e-07	2.924257e-09	24466406	newton step
6	1.766973e-08	2.924253e-09	26592864	newton step

Figure 2 shows the convergence curve for this run.

4.4.3 A sample trace minimization problem

In this problem we minimize $\text{trace}(Y^*AY)$ where A is a 12×12 second difference operator in 1-dimension and Y is 12×4 .

Note: we do not recommend that this problem be solved this way, since, for constant A , it can be done faster with conventional eigenvalue solvers.

```
>> !cp examples/tracemin/*.m .
>> A = Kinetic(12);
>> parameters(A);
>> Y0 = guess(4);
>> [fn,Yn] = sg_min(Y0,'frcg','euclidean');
```

iter	grad	F(Y)	flops
0	2.236291e+00	3.833127e+00	3357
1	1.413602e+00	1.903903e+00	222295
2	9.159910e-01	1.307258e+00	350264
3	5.663336e-01	1.057510e+00	490398
4	4.366693e-01	9.440163e-01	622177
5	2.676809e-01	8.724658e-01	763418
6	1.943933e-01	8.462110e-01	899391
7	1.200285e-01	8.319293e-01	1054873
8	5.106617e-02	8.279622e-01	1205566
9	2.633155e-02	8.272484e-01	1370430
10	1.009583e-02	8.270559e-01	1547221
11	4.054257e-03	8.270313e-01	1737513
12	1.416079e-03	8.270275e-01	1940319
13	7.175948e-04	8.270269e-01	2154843
14	4.258507e-04	8.270268e-01	2428070
15	3.065818e-04	8.270267e-01	2700816
16	1.260472e-04	8.270267e-01	2929162
17	3.596692e-05	8.270267e-01	3170795

Figure 3 shows the convergence curve for this run.

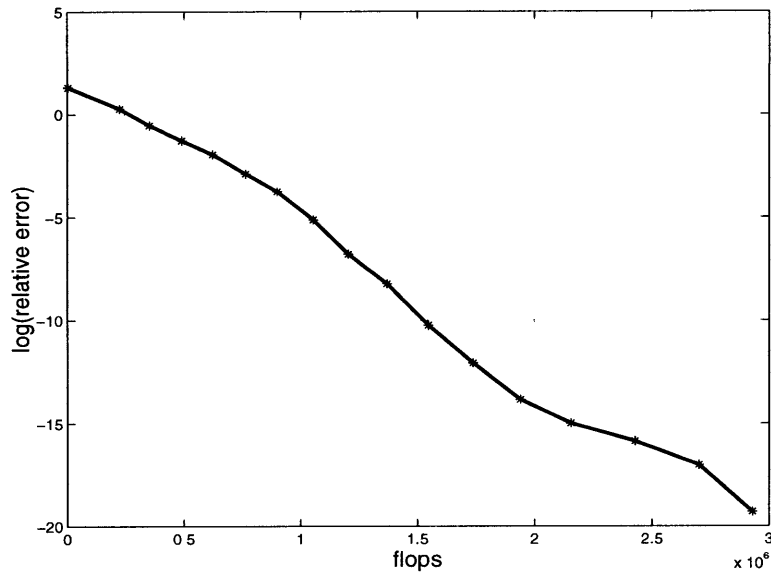


Figure 4-3: Trace Minimization Problem

4.4.4 A sample LDA toy problem (trace minimization with non-linear term)

We attempt to solve a toy Local Density Approximation problem described in section 3, for four electrons on a 1D grid with twelve points and a coupling constant of 20. In this case, we use the first four eigenvectors of the linear problem as a starting point.

```
>> !cp examples/ldatoy/*.m .
>> K = Kinetic(12);
>> parameters(K,20);
>> Y0 = guess(4);
>> [fn,Yn] = sg_min(Y0,'newton','euclidean');
```

iter	grad	F(Y)	flops	step type
0	2.432521e+00	7.750104e+00	4104	none
invdgrad: Hessian not positive definite, CG terminating early				
1	4.769809e-01	7.546426e+00	303003	steepest step
2	4.665924e-02	7.531450e+00	725498	newton step

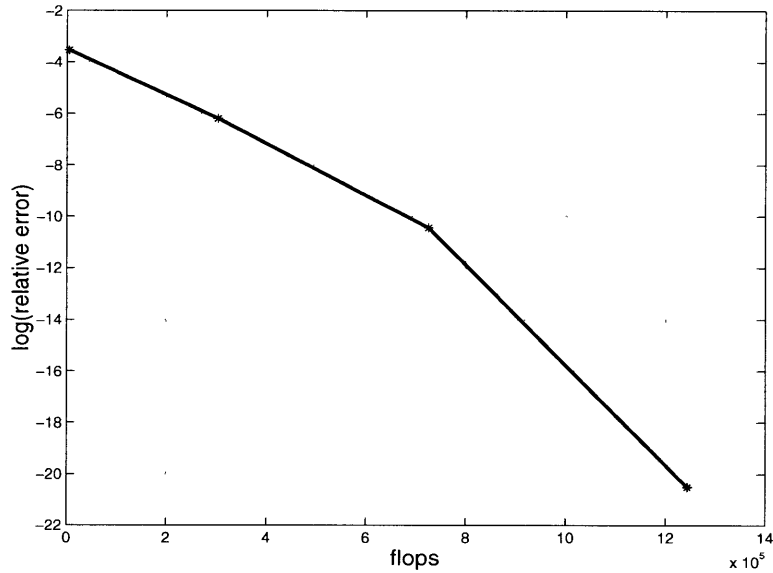


Figure 4-4: LDA Toy Problem

3	3.363281e-04	7.531232e+00	1243707	newton step
4	1.678208e-06	7.531232e+00	1868470	newton step

Figure 4 shows the convergence curve for this run.

4.4.5 A sample simultaneous Schur decomposition problem

In this case we use the sample problem given to us by Schilders.

```
>> !cp examples/simschur/*.m .
>> [A,B] = noisy;
>> parameters(A,B);
>> Y0 = guess;
>> [fn,Yn] = sg_min(Y0,'newton','flat');
```

iter	grad	F(Y)	flops	
0	2.205361e-01	1.687421e-02	54763	none
1	6.688045e-03	4.928702e-03	4808445	newton step
2	1.386562e-04	4.913259e-03	8560957	newton step

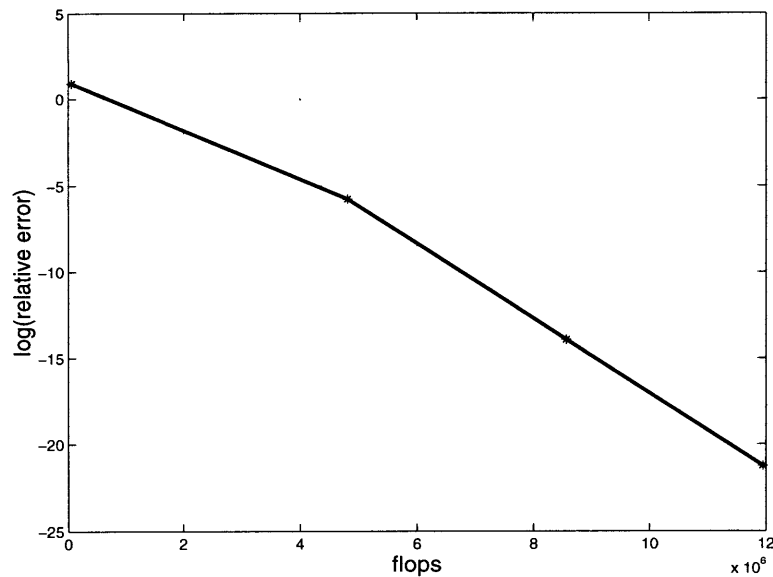


Figure 4-5: Simultaneous Schur Problem

3	2.377737e-06	4.913255e-03	11891025	newton step
4	2.314313e-07	4.913255e-03	14846552	newton step

Figure 5 shows the convergence curve for this run.

4.4.6 A sample INDSCAL problem (simultaneous diagonalization)

We attempt to simultaneously diagonalize two noisy versions of $\text{diag}(1:10)$ and $\text{diag}(1:10)^2$. Our initial point is the QR-factorization of the average of the eigenvectors of the two matrices.

```
>> !cp examples/indscal/*.m .
>> randn('state',0);
>> [A,B] = noisy;
>> parameters(A,B);
>> Y0 = guess;
>> [fn,Yn] = sg_min(Y0,'newton','euclidean');
```

iter	grad	F(Y)	flops	
0	1.871820e+03	5.837012e+02	29736	none
	invdgrad: Hessian not positive definite, CG terminating early			
1	1.141235e+03	1.996926e+02	2514846	steepest step
	invdgrad: Hessian not positive definite, CG terminating early			
2	6.017399e+02	7.364112e+01	8191352	newton step
	invdgrad: Hessian not positive definite, CG terminating early			
3	3.157921e+02	2.351357e+01	11094356	newton step
	invdgrad: Hessian not positive definite, CG terminating early			
4	1.908635e+02	1.450185e+01	15838109	steepest step
	invdgrad: Hessian not positive definite, CG terminating early			
5	1.385145e+02	1.136202e+01	20790540	steepest step
	invdgrad: Hessian not positive definite, CG terminating early			
6	1.274498e+02	9.160689e+00	25701418	newton step
7	2.162910e+00	5.662713e-03	34638359	newton step
8	3.022615e-03	2.224608e-06	42237728	newton step
9	1.708804e-03	2.223022e-06	43731038	newton step

Figure 6 shows the convergence curve for this run.

4.5 Making modifications (the structure of the code)

This template is written to function as is, but can be tailored to specific problems, resulting in substantial improvements in performance. There are also functions which could be in-lined and computations which can be reused of which we have not taken advantage for the sake of having a more readable code. Many components are the usual routines found in the literature (our line minimization routine is matlab's `fmin` for example). Certainly it is possible for improvements to be made in the optimization routines themselves and we welcome suggestions from optimization experts.

To make modifications as painless as possible, we present a section sketching the basic

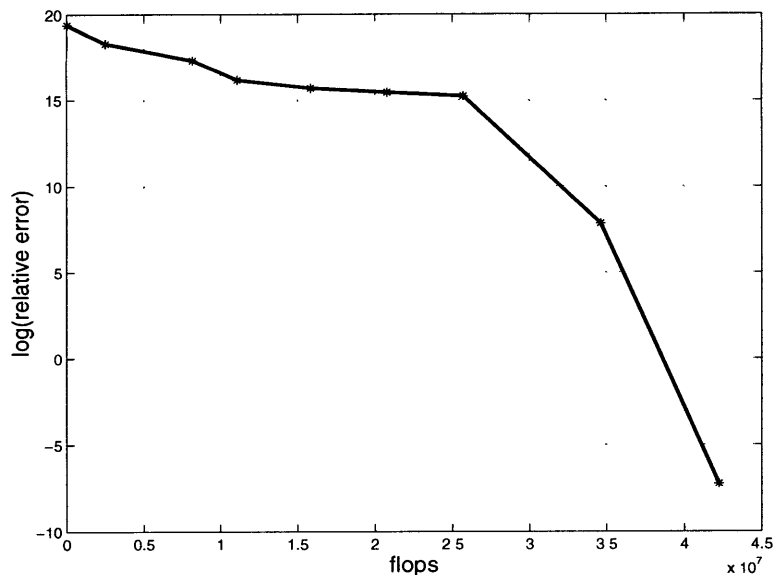


Figure 4-6: (INDSCAL) Simultaneous Diagonalization Problem

structure of the `sg_min` code. Readers who do not wish to modify any of the functions beyond `F.m`, `dF.m`, and `ddF.m` can skip this section.

4.5.1 Subroutine dependencies

Figure 7 shows the dependencies of the various subroutines on each other. We have grouped together the routines based on four loosely defined roles:

- Objective Function** These are the routines which implement the objective function and its derivatives. This group is the only group that a user interesting only in the basic functionality need ever modify to adapt the template. Functions in this group are used by functions in the Geometrized Objective Function group and High Level Algorithm group.
- Geometric Implementation** These routines implement the geometric features of the Stiefel manifold. This includes the projection of unconstrained vectors onto the constraint (i.e. tangent) space (`tangent`), line movement (`move`), and the connection term used for covariant differentiation (`connection`). These routines are independent of the

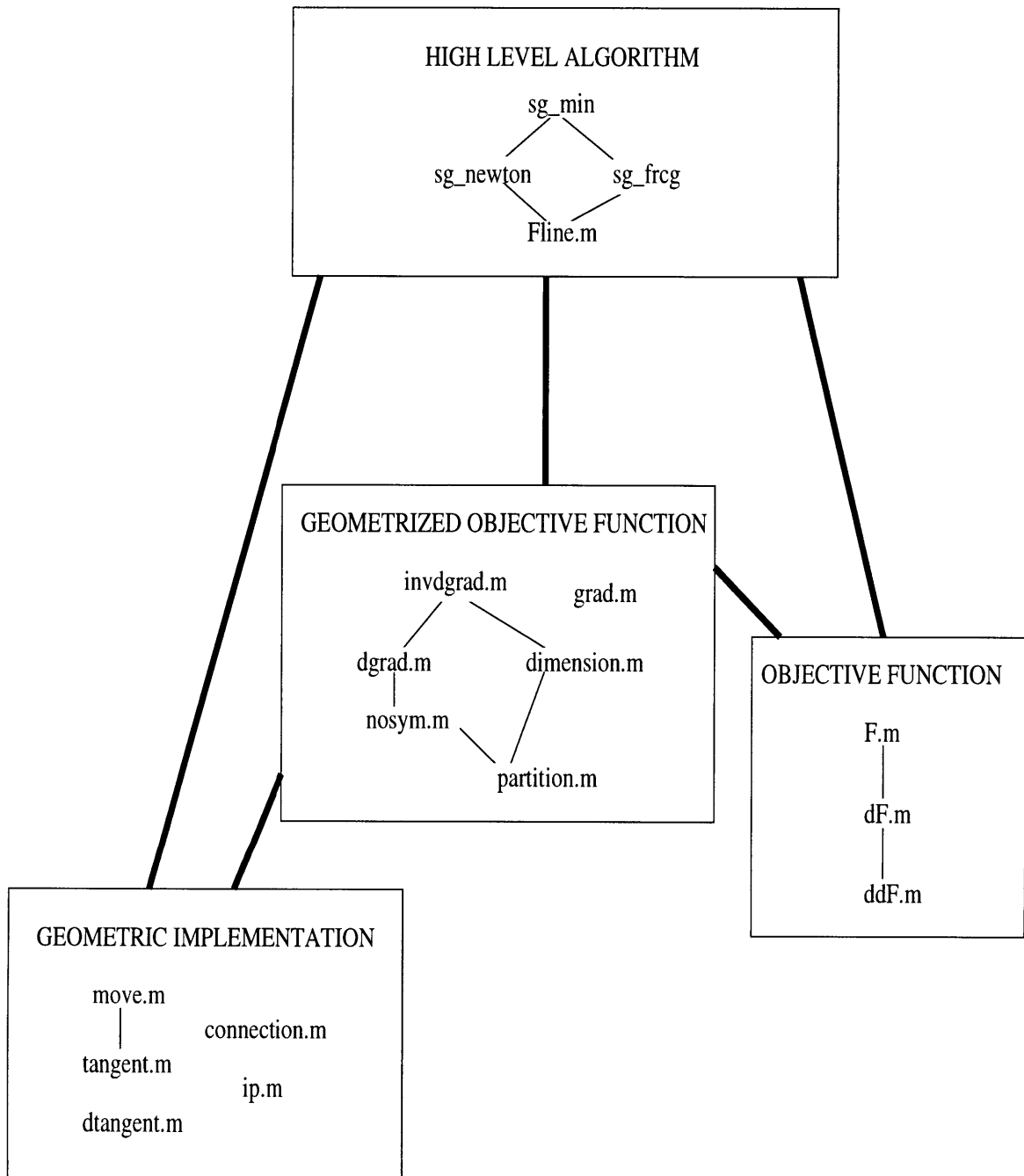


Figure 4-7: Our code tree: dependencies of various modules

Objective Function group, but are essential to the Geometrized Objective Function group.

- **Geometrized Objective Function** This group uses the routines of the Geometric Implementation to project the differential and second differential of $F(Y)$ onto the constraint (i.e. tangent) surface producing the geometrically covariant gradient (`grad`) and Hessian (`dgrad`). Additionally, the routine for inverting the covariant Hessian by conjugate gradient (`invdgrad`) is located here. This group provides the raw tools out of which one builds implementations in the High Level Algorithm group. Lastly, functions which detect (`partition`) and remove (`nosym`) block diagonal orthogonal symmetries are found in the group.
- **High Level Algorithm** This group implements the constrained versions of various optimization algorithms. It is here search directions are selected, line minimizations are performed (using `Fline`), and convergence criteria are defined.

Lastly, every function reads from a global `SGParameters` structure whose fields contain information about the manifold and the computation. In our examples we have used a separate global `FParameters` to store information related to the computation of $F(Y)$. The fields of `SGParameters` are set in `sg_min`.

4.5.2 Under the hood

`sg_min` is built from a couple fairly elementary optimization codes which have been put through a geometrization that allows them to sit on the Stiefel manifold. The basic elements of a geometrization are the rules for how to take inner products, how to turn unconstrained differentials into constrained gradients, how to differentiate gradient fields covariantly, and how to move about on the manifold.

The `sg_min` routine parses the arguments and sets the defaults. Finally, it calls either `sg_newton` or `sg_frcg`. The source for these routines (in pseudocode) follows:

```

function [fn,Yn]= sg_newton(Y)
    ginitial = grad(Y); g = ginitial;
    f = F(Y); fold = 2*f;
    while (||g|| > eps) & (||g||/||ginitial|| > gradtol) & (fold/f-1 | > ftol)
        sdir = -g;
    % steepest descent direction and line minimization on different scales
        fsa = minimize F(move(Y,sdir,sa)) for sa ∈ [-1, 1] · | $\frac{f}{\langle g, sdir \rangle}$ |;
        fsb = minimize F(move(Y,sdir,sb)) for sb ∈ [-1, 1] · | $\frac{\langle g, sdir \rangle}{\langle sdir, dgrad \cdot sdir \rangle}$ |;

        ndir = -dgrad-1 · g;
    % newton direction and line minimization
        fna = minimize F(move(Y,ndir,na)) for na ∈ [-1, 1] · | $\frac{f}{\langle g, ndir \rangle}$ |;
        fnb = minimize F(move(Y,ndir,nb)) for nb ∈ [-1, 1] · | $\frac{\langle g, ndir \rangle}{\langle ndir, dgrad \cdot ndir \rangle}$ |;

    % compare the best newton function value with the best steepest
        if (fsa < fsb) st=sa; fst=fsa; else st=sb; fst=fsb; end
        if (fna < fnb) nt=na; fnt=fna; else nt=nb; fnt=fnb; end
        if (fst < fnt)
            dir = sdir; t = st; fnew = fst;
        else
            dir = ndir; t = nt; fnew = fnt;
        end
    % move to the new point
        Y = move(Y,dir,t); fold= f; f = fnew;
        g=grad(Y);
    end
    fn = f;
    Yn = Y;

```

```

function [fn,Yn]= sg_frcg(Y)
    ginitial = grad(Y); g = ginitial;
    f = F(Y); fold = 2*f;
    while (||g|| > eps) & (||g||/||ginitial|| > gradtol) & (|fold/f-1| > ftol)
        dir = -g;
    % get a Hessian conjugate direction
        if (not first iteration)
            dir = dir -  $\frac{\text{dir} \cdot (\text{dgrad} \cdot \text{dir}_{old})}{\text{dir}_{old} \cdot (\text{dgrad} \cdot \text{dir}_{old})} * \text{dir}_{old}$ ;
        end
    % line minimization
        fcga = minimize F(move(Y,dir,cga)) for cga ∈ [-1, 1] ·  $\left| \frac{\langle \mathbf{f}, \mathbf{dir} \rangle}{\langle \mathbf{g}, \mathbf{dir} \rangle} \right|$ ;
        fcgb = minimize F(move(Y,dir,cgb)) for cgb ∈ [-1, 1] ·  $\left| \frac{\langle \mathbf{g}, \mathbf{dir} \rangle}{\langle \mathbf{dir}, \text{dgrad} \cdot \mathbf{dir} \rangle} \right|$ ;

        if (fcga < fcgb) t=cga; fnew=fcga; else t=cgb; fnew=fcgb; end
    % move to the new point
        [Y,dirold] = move(Y,dir,t); fold= f; f = fnew;
        g=grad(Y);
    end
    fn = f;
    Yn = Y;

```

These are fairly generic routines for optimization. At this level of description, one would not necessarily be able to tell them from unconstrained routines (see [34, 37, 52, 50]. What places them on the Stiefel manifold are the definitions of `grad`, `dgrad`, `ip` (the dot product), and `move`, which have been made in such a way that the constraints of the Stiefel manifold are respected.

4.5.3 What to modify

Here is a sketch of specific modifications a user may wish to make on the code.

- **Fline** Currently line minimizations are being performed by the matlab `fmin` function in conjunction with our `Fline` function. However, `fmin` is not necessarily the best

function to employ for line minimization. If one wished to use a function other than `fmin` then one could modify `sg_newton.m` and `sg_frcg.m` appropriately.

- **High Level Algorithms** The user may wish to use a minimization routine different from or more sophisticated than `sg_frcg` or `sg_newton`, or one may just have a different stopping criteria [6] that we did not anticipate. It is possible to adapt flat optimization algorithms to new `sg` algorithms without having to know too many details. Most unconstrained minimizations contain line searches ($y \rightarrow y + t*v$), Hessian applications ($H*v$) and inner products ($v'w$). To properly geometrize the algorithm, one has to do no more than replace these components with `move(Y,V,t)`, `dgrad(Y,V)`, and `ip(Y,V,W)` respectively.
- **Hessian Inversion** We currently provide codes to invert the covariant Hessian in the function `invdgrad` (which uses a conjugate gradient method). Any matrix inversion routine which can stably invert black-box linear operators can be used in place of these and no other modifications to the code would be required. Another feature the user may wish to add is an inversion routine to produces dog-leg steps.
- **Workspaces** To make the code more readable we have not implemented any workspace variables even though there are many computations which could be reused. For example, in the objective function group, for many of the sample problems, products of the form $A*Y$ or $Y*B$ or complicated matrix functions of Y such as $B(Y)$ in the Jordan example could be saved in a workspace. Some of the terms in the computation of `grad` and `dgrad` could likewise be saved (in fact, our current implementation essentially recomputes the gradient every time the covariant Hessian is applied).
- **In-lining** Although we have separated the `tangent` and `dtangent` functions from the `grad` and `dgrad` functions, one often finds that when these functions are in-lined, that some terms cancel out and therefore do not need to be computed. One might also consider in-lining the `move` function when performing line minimizations so that one could reuse data to make function evaluations and point updates faster.

- **Sphere or $O(n)$ Geometry** Some problems are always set on the unit sphere or on $O(n)$, both of which have simpler geometries than the Stiefel manifold. Geodesics on both are easier to compute, as are tangent projections. To take advantage of this, one should modify the Geometric Implementation group.
- **Globals** In order to have adaptable and readable code, we chose to use a global structures, `SGParameters` and `FParameters`, to hold data which could be put in the argument lists of all the functions. The user may wish to modify his/her code so that this data is passed explicitly to each function as individual arguments.
- **Preconditioning** The Hessian solve by conjugate gradient routine, `invdgrad`, does not have any preconditioning step. Conjugate gradient can perform very poorly without preconditioning, and the preconditioning of black box linear operators like `dgrad` is a very difficult problem. The user might try to find a good preconditioner for `dgrad`, but this may be difficult.

4.6 Geometric technicalities

This section is a brief explanation of the geometric concepts used to produce the methods employed by `sg_min` and its subroutines.

4.6.1 Manifolds

A manifold, M , is a collection of points which have a differential structure. In plain terms, this means that one is able to take derivatives of some reasonable class of functions, the C^∞ functions, defined on M . What this class of differentiable functions is can be somewhat arbitrary, though some technical consistency conditions must be satisfied. We will not be discussing those conditions in this section.

We consider the manifold $\text{Stief}_R(n, k)$ (for purposes of clearer explanation we restrict our discussion to the real version of the Stiefel manifold) of points which are written as $n \times k$ matrices satisfying the constraint $Y^*Y = I$. We will select for our set of C^∞ functions those real valued functions which are restrictions to $\text{Stief}(n, k)$ of functions of nk variables which

are $C^\infty(R^{n \times k})$ about $\text{Stief}(n, k)$ in the $R^{n \times k}$ sense. It should not be difficult to convince oneself that the set of such functions must satisfy any technical consistency condition one could hope for.

Additionally, $M = \text{Stief}(n, k)$ is a smooth manifold. This means that about every point of $\text{Stief}(n, k)$ we can construct a local coordinate system which is C^∞ . For example, consider the point

$$Y = \begin{pmatrix} I \\ 0 \end{pmatrix}$$

and a point in the neighborhood of Y ,

$$\tilde{Y} = \begin{pmatrix} A \\ B \end{pmatrix}.$$

For small B , we can solve for A in terms of the components of B and the components of an arbitrary small antisymmetric matrix Δ by solving $S^*S = I - B^*B$ for symmetric S and letting $A = e^\Delta S$. This can always be done smoothly and in a locally 1-to-1 fashion for small enough B and Δ . Since the components of B are all smooth functions (being restrictions of the coordinate functions of the ambient $R^{n \times k}$ space), and since the solution for A is $C^\infty(R^{(n-k) \times k} \oplus R^{k \times k})$ for small enough B and Δ , we have shown that any point of $\text{Stief}(n, k)$ can be expressed smoothly as a function of $(n-k)k + k(k-1)/2 = nk - k(k+1)/2$ variables. The only difference between $\begin{pmatrix} I \\ 0 \end{pmatrix}$ is a euclidean rigid motion; therefore, this statement holds for all points in $\text{Stief}(n, k)$.

Summarizing, a manifold is a set of points, with a set of differentiable functions defined on it as well as a sense of local coordinatization in which the dependent coordinates can be represented as smooth functions of the independent coordinates. Specifically, we see that there are always $nk - k(k+1)/2$ independent coordinates required to coordinatize neighborhoods of points of $\text{Stief}(n, k)$. This number is called the *dimension* of the manifold, and it should be the same for every point of a manifold (if not, then the manifold is either disconnected or not actually smooth).

4.6.2 The difference between a tangent and a differential

Given a smooth point x on any manifold and a smooth path $p(s)$ in the manifold such that $p(0) = x$, one can construct a differential operator on the C^∞ functions defined near x by the rule

$$D_p f = \frac{d}{ds} f(p(s))|_{s=0}.$$

Now, it turns out that these differential operators at x form a finite dimensional vector space, and the dimension of this vector space is equal to the dimension of the manifold. This vector space is called the *tangent space* of M at x , and is often denoted $T_x(M)$.

For the Stiefel manifold, and, generally, all smooth constraint manifolds, the tangent space at any point is easy to characterize. The constraint equation $Y^*Y = I$ can be thought of as $k(k+1)/2$ independent functions on $\text{Stief}(n, k)$ which must all have constant value. If H is a tangent vector of $\text{Stief}(n, k)$ at Y , it must then be that

$$H^*Y + Y^*H = 0,$$

which is obtained by taking $\frac{d}{dt} Y(t)^*Y(t) = 0|_{t=0}$ (where $\dot{Y}(0) = H$). In a constraint manifold, the tangents at Y can equivalently be identified with those infinitesimal displacements of Y which preserve the constraint equations to first order.

For H to be a tangent, its components must satisfy $k(k+1)/2$ independent equations. These equations are all linear in the components of H , and thus the set of all tangents is a $nk - k(k+1)/2$ dimensional subspace of the vector space $R^{n \times k}$.

A differential operator D_H may be associated with an $n \times k$ matrix, H , given by

$$D_H f = \frac{d}{dt} f(Y + tH)|_{t=0}.$$

D_H is the directional derivative operator in the H direction. One may observe that for H to be a tangent vector, the tangency condition is equivalent to $D_H(Y^*Y) = 0$.

While tangents are $n \times k$ matrices and, therefore, have associated differential operators, *differentials* are something else. Given a C^∞ function f and a point Y , one can consider the

equation $D_H f$ for $H \in R^{n \times k}$ (H is not necessarily tangent). This expression is linear in H and takes on some real value. It is thus possible to represent it as a linear function on the vector space $R^{n \times k}$,

$$D_H f = \text{tr}(H^* Z),$$

for some appropriate $n \times k$ matrix Z whose values depend on the first order behavior of f near Y . We identify this Z matrix with df , the differential of f at Y . This is the same differential which is computed by the dF functions in the sample problems.

For any constraint manifold the differential of a smooth function f can be computed without having to know anything about the manifold itself. One can simply use the differentials as computed in the ambient space (the unconstrained $R^{n \times k}$ derivatives in our case). If one then restricts one's differential operators to be only those in tangent directions, then one can still use the unconstrained df in $\text{tr}(H^* df)$ to compute $D_H f$ for $H \in T_Y(\text{Stief}(n, k))$. This is why it requires no geometric knowledge to produce the dF functions.

4.6.3 Inner products, gradients, and differentials

In flat spaces, we often identify the differential of a function with its gradient. However, when dealing with a more general setting, one can run into problems making sense out of such a definition.

For example, the gradient is a vector, and it should be possible to think of vectors as infinitesimal displacements of points. In $\text{Stief}(n, k)$, any infinitesimal displacement δY must satisfy $\delta Y^* Y + Y^* \delta Y = 0$. Thus, df may not always be a vector, since it does not necessarily satisfy this equation. A gradient should be an infinitesimal displacement that points in the direction of the displacement which will give the greatest increase in f .

If the tangent space has an inner product, though, one can find a useful way to identify the df uniquely with a tangent vector. Let $ip(H_1, H_2)$ be a symmetric nondegenerate bilinear form on the tangent space of $\text{Stief}(n, k)$ at Y . Then one can define the gradient, G , implicitly by,

$$ip(G, H) = \text{tr}(H^* df) = D_H f.$$

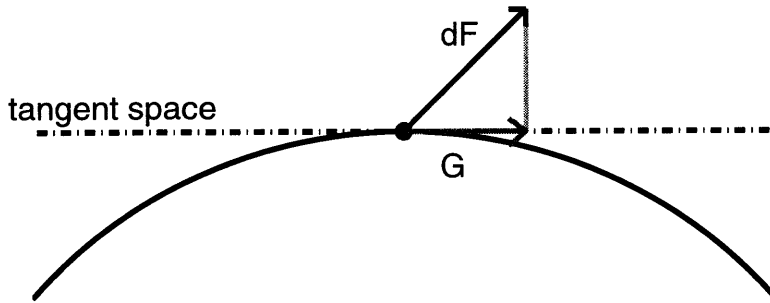


Figure 4-8: The unconstrained differential of $F(Y)$ can be projected to the tangent space to obtain the covariant gradient, G , of F .

Since ip is a nondegenerate form, this is sufficient to define G . The function `tangent` carries out this projection from differentials to tangents (shown in Figure 4-8). This operation is performed by `grad` to produce the gradient of the objective function.

4.6.4 Getting around $\text{Stief}(n, k)$

We've now laid the groundwork for a meaningful definition of the gradient in the setting of a constraint manifold. At this point, one could run off and try to do a steepest descent search to maximize one's objective functions. Trouble will arise, however, when one discovers that there is no sensible way to combine a point and a displacement to produce a new point because, for finite s , $Y + sH$ violates the constraint equations, and thus does not give a point on the manifold.

For any manifold, one updates Y by solving a set of differential *equations of motion* of the form

$$\begin{aligned}\frac{d}{dt}Y &= H \\ \frac{d}{dt}H &= -\Gamma(H, H).\end{aligned}$$

The Γ term is crafted to ensure that H remains a tangent vector for all times t , thus keeping the path on the manifold. It is called the *connection*. (The connection term also depends on Y .)

To see how these equations could be satisfied, we take the infinitesimal constraint equation

for H ,

$$H^*Y + Y^*H = 0,$$

and differentiate it with respect to t , to get

$$\Gamma(H, H)^*Y + Y^*\Gamma(H, H) = 2H^*H.$$

This can be satisfied generally by $\Gamma(H, H) = Y(H^*H) + T(H, H)$ where $T(H, H)$ is some arbitrary tangent vector.

In the next subsection, we will have reason to consider $\Gamma(H_1, H_2)$ for $H_1 \neq H_2$. For technical reasons, one usually requests that $\Gamma(H_1, H_2) = \Gamma(H_2, H_1)$ (this is called the *torsion free* property), and that $ip(\Gamma(H_1, H_3), H_2) + ip(H_1, \Gamma(H_2, H_3)) = 0$ (this is called the *metric compatibility* property). These two properties uniquely determine the $T(H_1, H_2)$ term, and thereby uniquely specify the connection. On any manifold, the unique connection with these properties is called the *Levi-Civita* connection.

The connection for the Stiefel manifold using two different inner products (the Euclidean and the canonical) can be found in the work by Edelman, Arias, and Smith (see [1, 31, 56]). The function `connection` computes $\Gamma(H_1, H_2)$ in the template software.

Usually the solution of the equations of motions on a manifold are very difficult to carry out. For the Stiefel manifold, analytic solutions exist and can be found in the aforementioned literature, though we have found very little performance degradation between moving along paths via the equations of motion and simply performing a QR factorization on $Y + sH$, as long as the displacements are small. Since QR factorization is cheaper, the software has this as its default (`motion= 'approximate'`). We have provided the two equations of motion solvers (for the Euclidean and canonical connections) as well as movement by QR factorization in `move` to satisfy both the purists and the pragmatists respectively.

4.6.5 Covariant differentiation

With the notion of a gradient and a notion of movement that respects the constraints of the manifold, one might wish to begin with an optimization routine of some sort. A steepest

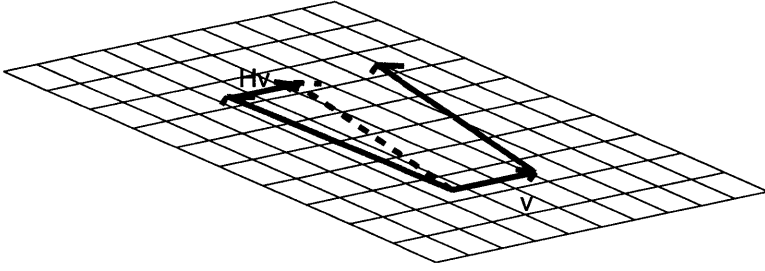


Figure 4-9: In a flat space, comparing vectors at nearby points is not problematic.

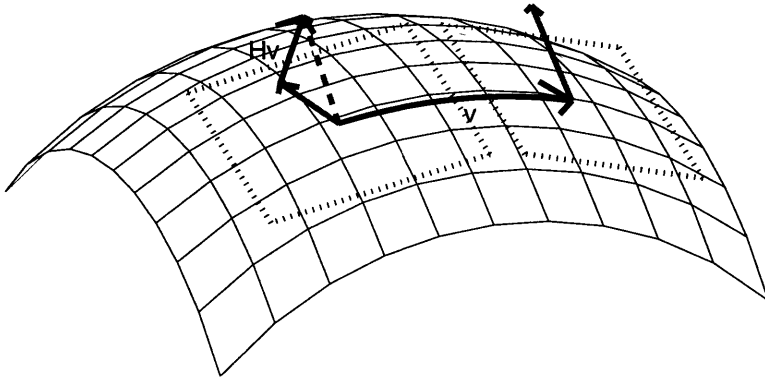


Figure 4-10: In a curved manifold, comparing vectors at nearby points can result in vectors which do not lie in the tangent space.

descent could be implemented from these alone. However, in order to carry out sophisticated optimizations, one usually wants some sort of second derivative information about the function.

In particular, one might wish to know by how much one can expect the gradient to change if one moves from Y to $Y + \epsilon H$. This can actually be a difficult question to answer on a manifold. Technically, the gradient at Y is a member of $T_Y(\text{Stief}(n, k))$, while the gradient at $Y + \epsilon H$ is a member of $T_{Y+\epsilon H}(\text{Stief}(n, k))$. While taking their difference would work fine in a flat space (see Figure 4-9), if this were done on a curved space, it could give a vector which is not a member of the tangent space of either point. (see Figure 4-10).

A more sophisticated means of taking this difference is to first move the gradient at $Y + \epsilon H$ to Y in some manner which translates it in a parallel fashion from $Y + \epsilon H$ to Y , and then compare the two gradients within the same tangent space. One can check that for $V \in T_{Y+\epsilon H}(\text{Stief}(n, k))$ the rule

$$V \rightarrow V + \epsilon \Gamma(V, H),$$

where $\Gamma(V, H)$ is the Levi-Civita connection, takes V to an element of $T_Y(\text{Stief}(n, k))$ and preserves inner product information (to first order in ϵ). This is the standard rule for parallel transport which can be found in the usual literature ([14, 69, 44, 41], and others).

Using this rule to compare nearby vectors to each other, one then has the following rule for taking derivatives of vector fields:

$$D_H G = \frac{d}{ds} G(Y + sH)|_{s=0} + \Gamma(G, H),$$

where G is any vector field (but we are only interested in derivatives of the gradient field). This is the function implemented by `dgrad` in the software.

In an unconstrained minimization, the second derivative of the gradient $g = \nabla f$ along a vector \vec{h} is the Hessian $[\frac{\partial^2 f}{\partial x_i \partial x_j}]$ times \vec{h} . Covariantly, we then have the analogy,

$$[\frac{\partial^2 f}{\partial x_i \partial x_j}] \vec{h} = (\vec{h} \cdot \nabla) g \sim D_H G.$$

4.6.6 Inverting the covariant Hessian (technical considerations)

Since the tangent space of $\text{Stief}(n, k)$ is a subspace of $R^{n \times k}$, the covariant Hessian must be inverted stably on this subspace. This requires any algorithms designed to solve $D_H G = V$ for H to really be pseudoinverters in a least squares or some other sense.

A second consideration is that many useful functions $f(Y)$ have the property that $f(YQ) = f(Y)$ for all block diagonal orthogonal Q (i.e.

$$Q = \begin{pmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & \cdots & 0 \\ 0 & 0 & \cdots & Q_p \end{pmatrix},$$

where the Q_i are orthogonal matrices). In this case, all tangent vectors of the form

$$H = Y \begin{pmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ 0 & 0 & \cdots & A_p \end{pmatrix},$$

where the H_i are antisymmetric, have the property $D_H f = 0$. These extra symmetry vectors are then null vectors of the linear system $D_H G = V$. Because of these null directions, the effective dimension of the tangent space is reduced by the dimension of this null space (this is the dimension returned by the `dimension` function).

Thus, we see that in order to invert the covariant Hessian, we must take care to use a stable inversion scheme which will project out components of H which do not satisfy the infinitesimal constraint equation and those which are in the direction of the additional symmetries of f . The `invdgrad` function carries out a stable inversion of the covariant Hessian by a conjugate gradient routine, with the `dgrad` function calling the function `nosym` to project out any extra symmetry components.

Chapter 5

Multiscale Computation with Interpolating Wavelets

5.1 Introduction

Wavelets offer a means of approximating functions that allows selective refinement. If regions of an image or a signal have exceptionally large variations, one need only store a set of coefficients, determined by function values in neighborhoods of those regions, in order to reproduce these variations accurately. In this way, one can have approximations of functions in terms of a basis that has spatially varying resolution. This approach reduces the memory storage required to represent functions and may be used for data compression.

Physical applications often involve multiple physical fields which interact in space with non-linear couplings. Efficient implementations must minimize not only storage to represent the these fields but also the processing required to describe their interactions. It is highly desirable to perform the needed operations with a fixed, limited number of floating point operations for each expansion coefficient and to minimize the number of points in space at which physical interactions must be evaluated.

As a concrete example of a realistic application, consider the computation of the quantum mechanical electronic structure of a collection of atoms in three dimensions. For other examples of physical applications, the reader may wish to consult [7], [9], [35], *et al.*. Arias

and coworkers [15],[5], and other works which have appeared after the original submission of this manuscript nearly one year ago [65],[64], have studied the use of multiresolution bases in quantum mechanical computations. (For a review, see [3].) It is a consequence of quantum physics that, near atomic nuclei, electronic wave functions vary rapidly and, that far from the nuclei, the wave functions tend to be much more smooth. From this observation, one can anticipate that the fine scale coefficients in a multiresolution analysis of the electronic wave functions and associated physical fields will be significant only near the atomic cores, allowing for truncation. This problem is thus an ideal candidate for multiresolution techniques.

Within density functional theory [42], the quantum physics of electrons and nuclei involves two types of fields, the Schrödinger wave function $\{\psi_i(r)\}$ for each electron i and the electrostatic potential $\phi(r)$ arising from the average electronic density $n(r) \equiv \sum_i |\psi_i(r)|^2$. Within the local density approximation (LDA) [45], the solution for the correct values of these fields is obtained at the saddle-point of lowest energy of the Lagrangian functional

$$\begin{aligned} \mathcal{L}_{LDA}(\{\psi_i\}, \phi) &= \frac{1}{2} \sum_i \int d^3r \|\nabla \psi_i(r)\|^2 + \int d^3r V_{\text{nuc}}(r)n(r) + \int d^3r \epsilon_{xc}(n(r))n(r) \\ &- \int d^3r \phi(r)n(r) - \frac{1}{8\pi} \int d^3r \|\nabla \phi(r)\|^2. \end{aligned} \quad (5.1)$$

Here, we work in units $\hbar = m = e = 1$, $V_{\text{nuc}}(r)$ is the total potential which each electron feels due to the presence of the atomic nuclei, and $\epsilon_{xc}(n)$ is a non-algebraic function known only through tabulated values. (For a review of density functional theory, see [51].)

In practice, one finds the fields $\{\psi_i(r)\}$, $\phi(r)$ by

- expanding the fields in terms of unknown coefficients within some basis set

$$\begin{aligned} \psi_i(x) &= \sum_{\alpha} c_{\alpha,i} b_{\alpha}(x) \\ \phi(x) &= \sum_{\alpha} d_{\alpha} b_{\alpha}(x); \end{aligned} \quad (5.2)$$

- evaluating Equation (5.1) in terms of the unknown coefficients c and d ;
- determining analytically the gradients of the resulting $\mathcal{L}_{LDA}(c, d)$ with respect to those coefficients; and

- proceeding with conjugate gradients to locate the saddle point.

All follows directly once one has expressed the Lagrangian as a function of the expansion coefficients.

In doing this, we note that each term represents a local coupling in space, but that one coupling, $\phi(r)n(r)$, is cubic in the field coefficients c and d , and another, $\epsilon_{xc}(n(r))n(r)$, is only known in terms of tabulated values. Expanding the product of two wavelets in terms of wavelets on finer levels would make possible the treatment of the cubic coupling to some level of approximation. (See, for example, [10].) However, this route becomes increasingly difficult for higher order interactions and is hopeless for non-algebraic or tabulated interactions, such as $\epsilon_{xc}(n(r))$. For higher order interactions it is natural, and for non-algebraic and tabulated interactions necessary, to evaluate the interactions at some set of points in space and then recover expansion coefficients for the result. One then relies upon the basis set to provide interpolation for the behavior at non-sample points.

The benefits of both truncated wavelet bases and interpolation on dyadically refined grids are given by the use of interpolating scaling functions [27], [11], [12], [24], [26] (or interpolets [70], [3]), which are functions with the following properties (from [26], pp. 6-7).

Let $\phi(x)$ be an interpolet, then

(INT1) **cardinality:** $\phi(k) = \delta_{0,k}$ for all $k \in Z^n$

(INT2) **two-scale relation:** $\phi(x/2) = \sum_{y \in Z^n} c_y \phi(x - y)$

(INT3) **polynomial span:** For some integer $m \geq 0$, any polynomial $p(x)$ of degree m can be represented as a formal sum $\sum_{y \in Z^n} a(y) \phi(x - y)$.

Cardinality allows the fast conversion between uniform samples and interpolating scaling functions and has subtle yet profound consequences for the resulting multiresolution basis. In particular, as is evident from our algorithms below, the expansion coefficient for a basis function on a particular scale is independent of the samples of the function for points associated with finer scales. Consequently, the expansion coefficients which we obtain for functions maintained in our basis are identical to what would be obtained were function samples available on a complete grid of *arbitrarily* fine resolution. This eliminates all error in

the evaluation of non-linear, non-algebraic and tabulated interactions beyond the expansion of the result in terms of a finite set of basis functions.

The c_y in the two-scale relation are referred to as scaling coefficients, and cardinality actually implies that $c_y = \phi(y/2)$. The two-scale relation allows the resolution to vary locally in a mathematically consistent manner.

The polynomial span condition captures, in some sense, the accuracy of the approximation. By cardinality, we actually have $a(y) = p(y)$. We shall call m the polynomial order.

Interpolets thus can be thought of as a bridge between computations with samples on dyadically refined grids and computations in a multiresolution analysis. The former point of view is useful for performing local nonlinear operations, while the latter is useful for the application of local linear operators.

This manuscript explores $O(N)$ algorithms that calculate transforms and linear operators for grids of variable resolution but return, for the coefficients considered, *exactly* the same results as would be obtained using a full, uniform grid at the highest resolution *without* the need to introduce artificial temporary augmentation points to the grid during processing. We thus show that with relatively mild conditions on the variability of the resolution provided by the grid, interpolet bases provided the ultimate economy in the introduction of grid points: only as many samples in space need be considered as functions used in the basis. The four transforms (forward, inverse, and the dual to each) mapping between coefficients and functions samples which we discuss here are particular to interpolet bases. For the application of operators in such bases, we show that the familiar non-standard multiply of Beylkin, Coifman and Rokhlin[8] shares with the transforms the property of correctness without the need to introduce additional grid points. Furthermore, we weaken the condition on grid variability by using a modification of the non-standard multiply. We generalize the non-standard multiply so that communication may proceed between nearby but non-adjacent levels and thereby obtain less stringent conditions on the variability of the grid. All of theoretical results in this manuscript are presented in a general d -dimensional space. Illustrative examples for the purpose of discussion will be given in $d = 1$ and $d = 2$ dimensions. The examples of applications in the final section will be in $d = 3$ dimensions. Our focus is entirely on

C_k	$2^k \cdot Z^n$ for $k \geq 0$, Z^n for $k < 0$.
D_k	$C_{k-1} - C_k$ for $k \geq 0$, for $k < 0$.
θ_k	$\min(k, m)$ where m is the largest integer such that 2^m divides all the components of y .
$\mathcal{F}_k(S)$	the space of functions over $S \subset Z^n$.
$\mathcal{I}_k(\phi, S)$	the space of linear combinations of $\phi((x - y)/2^{\theta_k(y)})$ for $y \in S \subset Z^n$.
ι_k^ϕ	the mapping from $S \rightarrow \mathcal{I}_k(\phi, S)$ which takes $y \rightarrow \phi((x - y)/2^{\theta_k(y)})$ and linearly extended to a map from $\mathcal{F}_k(S) \rightarrow \mathcal{I}_k(\phi, S)$.
\mathcal{F}_k^S	$\mathcal{F}_k(Z^n)/\mathcal{F}_k(Z^n - S)$.
$\mathcal{I}_k^S(\phi)$	$\mathcal{I}_k(\phi, Z^n)/\mathcal{I}_k(\phi, Z^n - S)$.
\tilde{v}	the zero-lift representative of $v \in \mathcal{F}_k^S$, i.e. $\tilde{v} \in \mathcal{F}_k$, such that $\tilde{v}(y) = v(y)$, $y \in S$ and $\tilde{v}(y) = 0, y \notin S$.
V^*	the dual space of the vector space V .
J_{k_1, k_2}	the map, $\mathcal{F}_{k_1} \rightarrow \mathcal{F}_{k_2}$, given by $J_{k_1, k_2} = \iota_{k_2}^{-1} \circ \iota_{k_1}$.
J_k	short for $J_{0, k}$.
J_{-k}	short for $J_{k, 0}$.
$\langle f \mathcal{O} g \rangle$	the matrix element $\int f(x) \mathcal{O} g(x) d^n x$.

Table 5.1: Notation for multiscale computations.

interpolet bases, and so it remains an open question whether these results hold true or can be adapted to more general wavelet systems.

Our organization is as follows. In Sections 2 and 3, we explain how to manipulate and construct interpolet expansions and some aspects of how well they perform. These sections will present nothing new to the experienced wavelet user, but will explain our notational conventions (which are summarized in Table 5.1) and recapitulate common theorems ([19], [53], [20], [58], et al.) for wavelet novices. In Section 4, we describe how nonuniform bases can be conceptualized in the framework of interpolet expansions and then use our results to develop algorithms for the transforms. Section 5 details the algorithm for ∇^2 and other operators. Section 6 gives some practical details for the reader interested in implementing these algorithms. Finally, Section 7 compares, in three dimensions, timings of these implementations with the timings of naïve algorithms. This final section also explores the convergence of a preconditioned conjugate gradients algorithm in the solution of Poisson's equation for the full three dimensional electrostatic potential arising from the nuclei in the nitrogen molecule.

5.2 Introduction to interpolets

There is a unique set of interpolets on R having symmetry and minimal support for a given polynomial order $m = 2l - 1$ (the *Deslauriers-Dubuc functions* [24]). These are the functions with which this article is primarily concerned (our results carry over to more general interpolets, and no use will actually be made of minimal support or symmetry).

To determine the c_y 's, one sets $c_{2j} = \delta_{m0}$ and $c_y = c_{-y}$. One may solve the Vandermonde system,

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 3^2 & \cdots & (2l-1)^2 \\ 1 & 3^4 & \cdots & (2l-1)^4 \\ & & \cdots & \\ 1 & 3^{2l-2} & \cdots & (2l-1)^{2l-2} \end{pmatrix} \begin{pmatrix} c_1 \\ c_3 \\ \cdot \\ \cdot \\ c_{2l-1} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \\ \cdot \\ \cdot \\ 0 \end{pmatrix}$$

to obtain the remaining c_y 's. These coefficients satisfy the conditions for polynomial order $2l - 1$.

The scaling coefficients for $m = 1$ are

$$c_{-1} = c_1 = 0.5, c_0 = 1,$$

and for $m = 3$ (the example used for Figure 5-1.) they are

$$c_{-3} = c_3 = -\frac{1}{16}, c_{-1} = c_1 = \frac{9}{16}, c_0 = 1.$$

One may then take tensor products of ϕ 's and c_y 's to form interpolets in higher dimensions.

5.2.1 Interpolet Multiresolution Analysis

We are concerned with recursive representations of functions from samples at integer points on both uniform and refined grids. There are many definitions which make the exposition more clear.

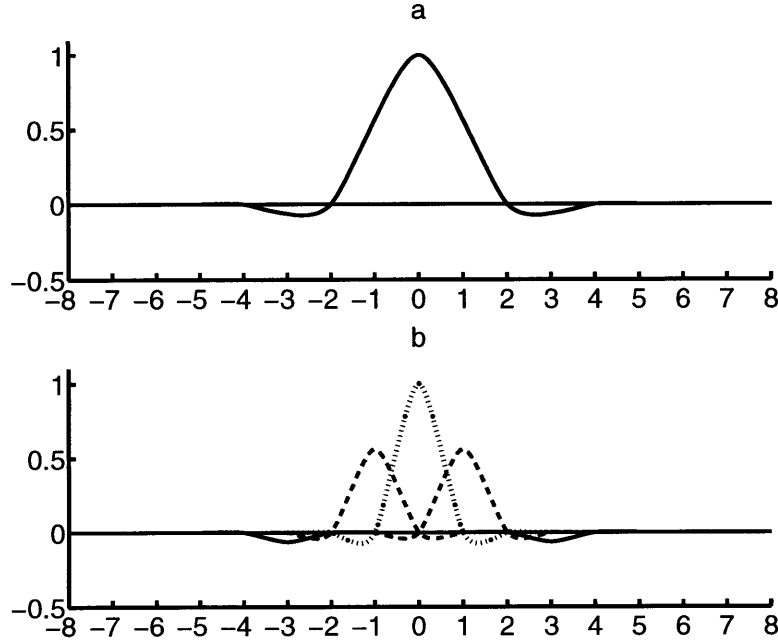


Figure 5-1: (a) shows $\phi(x/2) = \frac{-1}{16}\phi(x-3) + \frac{9}{16}\phi(x-1) + \phi(x) + \frac{9}{16}\phi(x+1) + \frac{-1}{16}\phi(x+3)$. (b) shows the functions $\frac{-1}{16}\phi(x-3), \frac{9}{16}\phi(x-1), \phi(x), \frac{9}{16}\phi(x+1), \frac{-1}{16}\phi(x+3)$.

Definition 5.2.1 For $k > 0$, let $C_k = 2^k Z^n$, and let $D_k = C_{k-1} - C_k$. For $k \leq 0$, let $C_k = Z^n$, and $D_k = \emptyset$.

We consider C_k to be the set of coarse lattice points on the lattice $2^{k-1}Z^n$ and D_k , the detail lattice points, to be those points on $2^{k-1}Z^n$ which are not coarse. Note: $D_k \cup C_k = C_{k-1}$, and $Z^n = C_k \cup D_k \cup D_{k-1} \cup \dots \cup D_1$ is a partition of Z^n .

Definition 5.2.2 We let $\theta_k(y) = \min(k, m)$ where m is the largest integer such that 2^m divides all of the components of y . We call $\theta_k(y)$ the level of the point y .

Given the partition $Z^n = C_k \cup D_k \cup D_{k-1} \cup \dots \cup D_1$, we have

$$\theta_k(y) = \begin{cases} k, & y \in C_k \\ l-1, & y \in D_l. \end{cases}$$

Definition 5.2.3 Let $S \subset Z^n$. Let $\phi(x)$ be an interpolet. Let $\mathcal{I}_k(\phi, S)$ be the space of functions given by formal sums of the form $\sum_{y \in S} a(y)\phi(\frac{x-y}{2^{\theta_k(y)}}$.

Where ϕ and S are understood, we may simply write \mathcal{I}_k .

Definition 5.2.4 Let $S \subset Z^n$. Let $\mathcal{F}_k(S)$ be the vector space of R - or C - valued functions on Z^n which are zero at any point not in S (i.e. with support contained in S).

Where S is understood, we may simply write \mathcal{F}_k . Note: $\mathcal{F}_k(S) = \mathcal{F}_k(S \cap C_k) \oplus \mathcal{F}_k(S \cap D_k) \oplus \mathcal{F}_k(S \cap D_{k-1}) \oplus \cdots \oplus \mathcal{F}_k(S \cap D_1)$.

The meaning of the k subscript will be established by the next definition, which will link vectors in \mathcal{F}_k with functions in \mathcal{I}_k . It is for this reason that while the \mathcal{F} 's are technically identical, they are semantically different. In practice, the \mathcal{F} 's are the actual data structures being stored on the computer.

Definition 5.2.5 Let $\phi(x)$ be an interpolant. Let $\iota_k^\phi : S \rightarrow \mathcal{I}_k(\phi, S)$ be defined by

$$\iota_k^\phi y = \phi\left(\frac{x-y}{2^k \theta_k(y)}\right).$$

This definition extends linearly to the mapping $\iota_k^\phi : \mathcal{F}_k(S) \rightarrow \mathcal{I}_k(\phi, S)$ defined by:

$$\iota_k^\phi v = \sum_{y \in S} v(y) (\iota_k^\phi y)$$

i.e.

$$(\iota_k^\phi v)(x) = \sum_{y \in S \cap C_k} v(y) \phi((x-y)/2^k) + \sum_{y \in S \cap D_k} v(y) \phi((x-y)/2^{k-1}) + \cdots + \sum_{y \in S \cap D_1} v(y) \phi(x-y).$$

The set S can be thought of as the set of points in a refined grid. The ι_k^ϕ identifications allow one to think of S as a set of functions, $\{(\iota_k^\phi y) | y \in S\}$, which form a basis of $\mathcal{I}_k(\phi, S)$. We will sometimes refer to S as a refined grid and sometimes as a basis with this identification understood.

One should think of the \mathcal{F}_k as spaces of coefficients for function expansions in the corresponding \mathcal{I}_k spaces, in the basis S . The ι_k^ϕ simply associate a set of coefficients in \mathcal{F}_k with a function in \mathcal{I}_k . When ϕ is understood, we may write just ι_k .

We are now in a position to state the basic theorems of interpolant expansions on uniform grids.

Theorem 5.2.6 *Let $\phi(x)$ be an interpolant on R^n . Then each mapping $\iota_k : \mathcal{F}_k(S) \rightarrow \mathcal{I}_k(\phi, S)$ ($k = 1, 2, \dots$) is an isomorphism.*

Proof: Since the map ι_k is surjective, it is only necessary to show that ι_k is injective. By the definition, $\iota_k v = 0$ if and only if there exist $v \in \mathcal{F}_k$ such that

$$0 = \sum_{y \in S \cap C_k} v(y) \phi((x-y)/2^k) + \sum_{y \in S \cap D_k} v(y) \phi((x-y)/2^{k-1}) + \dots + \sum_{y \in S \cap D_1} v(y) \phi(x-y).$$

Let $z \in S \cap C_k$. By (INT1), we have $\phi((z-y)/2^k) = \delta_{(z-y)/2^k, 0} = \delta_{z,y}$, for $y \in S \cap C_k$, and also $\phi((z-y)/2^l) = 0$, for $y \in S \cap D_l$. So, $(\iota_k v)(z) = v(z)$, $z \in S \cap C_k$, therefore $v(z) = 0$, $z \in S \cap C_k$. This being so, one then has $(\iota_k v)(z) = v(z)$, $z \in S \cap D_k$, so $v(z) = 0$, $z \in S \cap D_k$. Once again, $(\iota_k v)(z) = v(z)$, $z \in S \cap D_{k-1}$, thus we must have $v(z) = 0$, $z \in S \cap D_{k-1}$. Continuing in this manner, we deduce that $v(y) = 0$, $y \in S$, thus $v = 0$.

□

Corollary 5.2.7

$$\mathcal{I}_k(S) = \mathcal{I}_k(S \cap C_k) \oplus \mathcal{I}_k(S \cap D_k) \oplus \dots \oplus \mathcal{I}_k(S \cap D_1)$$

Since the sum is direct, the expansion is unique.

This corollary is a consequence of observations in the above proof.

Theorem 5.2.8 *Let $\phi(x)$ be an interpolant on R^n . Then*

$$\forall k, \mathcal{I}_k(\phi, Z^n) = \mathcal{I}_{k-1}(\phi, Z^n).$$

Consequently,

$$\forall k_1, k_2, \mathcal{I}_{k_1}(\phi, Z^n) = \mathcal{I}_{k_2}(\phi, Z^n).$$

Proof: To prove that $\mathcal{I}_k \subset \mathcal{I}_{k-1}$ we note that $\mathcal{I}_k \subset \mathcal{I}_{k-1} \cup \mathcal{I}_k(C_k)$. Thus we just need to show $\mathcal{I}_k(C_k) \subset \mathcal{I}_{k-1}$.

Translating by $z \in C_k$ and inserting powers of 2 where appropriate, one can rewrite (INT2) for ϕ as

$$\phi((x-z)/2^k) = \phi((x-z)/2^{k-1}) + \sum_{y \in D_k} c_{y/2^{k-1}} \phi((x-z-y)/2^{k-1}).$$

The terms in the right hand side are elements \mathcal{I}_{k-1} . Thus $\mathcal{I}_k \subset \mathcal{I}_{k-1}$.

To prove $\mathcal{I}_{k-1} \subset \mathcal{I}_k$ note that any element of \mathcal{I}_{k-1} can be expressed as:

$$f(x) = \sum_{y \in C_{k-1}} a(y) \phi((x-y)/2^{k-1}) + \sum_{y \in D_{k-1}} a(y) \phi((x-y)/2^{k-2}) + \cdots + \sum_{y \in D_1} a(y) \phi(x-y)$$

All the terms in this expansion but the first are elements of \mathcal{I}_k . Since $C_{k-1} = C_k \cup D_k$ we may split the first sum up as,

$$\sum_{y \in C_{k-1}} a(y) \phi((x-y)/2^{k-1}) = \sum_{y \in C_k} a(y) \phi((x-y)/2^{k-1}) + \sum_{y \in D_k} a(y) \phi((x-y)/2^{k-1})$$

The second term is also an element of \mathcal{I}_k .

Rewriting (INT2) one has ($y \in C_k$):

$$\phi((x-y)/2^{k-1}) = \phi((x-y)/2^k) - \sum_{z \in D_k} c_{z/2^{k-1}} \phi((x-z-y)/2^{k-1}).$$

$y \in C_k, z \in D_k$, so $y+z \in D_k$, thus the right hand side is made up of elements of \mathcal{I}_k . Thus, $\mathcal{I}_{k-1} \subset \mathcal{I}_k$.

□

5.2.2 Interpolet transforms

Corollary 5.2.9 *Interpolet Decomposition*

The set of isomorphisms ι_k induces a set of isomorphisms

$$J_{k_1, k_2} : \mathcal{F}_{k_1} \rightarrow \mathcal{F}_{k_2},$$

$$J_{k_1, k_2} = \iota_{k_2}^{-1} \circ \iota_{k_1}.$$

We refer to these isomorphisms as *interpolet transforms*. It is our convention to let $J_k = J_{0, k}$ and $J_{-k} = J_{k, 0}$. The reader will note that the J_i are linear transformations on the coefficient spaces, and are thus the primary object of computation.

We now turn to a study of the J 's. It is clear from the definition that for $k_1 < k_2$, $J_{k_1, k_2} = J_{k_2-1, k_2} \circ J_{k_2-2, k_2-1} \circ \cdots \circ J_{k_1, k_1+1}$, and similarly for $k_1 > k_2$. Thus, we need only study the $J_{k, k+1}$ and $J_{k+1, k}$ mappings.

Theorem 5.2.10 *Computation Theorem*

Let $v \in \mathcal{F}_k(\mathbb{Z}^n)$.

$$(J_{k, k+1}v)(y) = \begin{cases} v(y), & y \notin D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) - \sum_{z \in C_{k+1}} c_{(y-z)/2^k} v(z)$.

$$(J_{k+1, k}v)(y) = \begin{cases} v(y), & y \notin D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) + \sum_{z \in C_{k+1}} c_{(y+z)/2^k} v(z)$.

Proof: For $v \in \mathcal{F}_k$ we have

$$\iota_k v = \sum_{y \in C_k} v(y) \phi((x-y)/2^k) + \sum_{y \in D_k} v(y) \phi((x-y)/2^{k-1}) + \cdots + \sum_{y \in D_1} v(y) \phi(x-y)$$

expanding the first term,

$$\iota_k v = \sum_{y \in C_{k+1}} v(y) \phi((x-y)/2^k) + \sum_{y \in D_{k+1}} v(y) \phi((x-y)/2^k) + \sum_{y \in D_k} v(y) \phi((x-y)/2^{k-1}) + \dots$$

using (INT2), $\phi((x-z)/2^k) = \phi((x-z)/2^{k+1}) - \sum_{y \in D_{k+1}} c_{(y-z)/2^k} \phi((x-y)/2^k)$,

$$\iota_k v = \sum_{y \in C_{k+1}} v(y) \phi((x-y)/2^{k+1}) + \sum_{y \in D_{k+1}} v'(y) \phi((x-y)/2^k) + \sum_{y \in D_k} v(y) \phi((x-y)/2^{k-1}) + \dots$$

$$(\iota_{k+1}^{-1} \iota_k v)(y) = \begin{cases} v(y), & y \notin D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) - \sum_{z \in C_{k+1}} c_{(y-z)/2^k} v(z)$.

The proof for $J_{k+1,k}$ is similar.

□

Similar to what one might get with wavelets, we see that we can compute the coefficients of interpolet expansions on uniform lattices by a pyramid algorithm. Computationally, this procedure can be carried out by first computing the D_1 coefficients with $J_{0,1}$, then by computing the D_2 coefficients from the C_1 data with $J_{1,2}$, and so on. In this sense, it is no different from standard multiresolution decompositions.

A feature of the interpolet decomposition is that the transformations all have a particular *lower triangular* form. That is, if we write $v \in \mathcal{F}_k$ as a vector with its C_{k+1} components first, its D_{k+1} components second, and the rest of its components third, then the transformation takes the form,

$$J_{k,k+1} v = \begin{pmatrix} I & 0 & 0 & 0 & 0 \\ M & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{pmatrix} \begin{pmatrix} v_{C_{k+1}} \\ v_{D_{k+1}} \\ v_{D_k} \\ \cdot \\ \cdot \end{pmatrix}.$$

The inverse, $J_{k+1,k}$, is obtained by replacing M with $-M$.

5.3 Accuracy of interpolet approximation

Given a function $f(x)$ on R^n , one can form an interpolet approximation to f by the formula:

$$f(x) \sim \sum_{y \in C_0} f(y) \phi(x - y) = \iota_0 f,$$

where f on the right hand side is thought of as a function restricted to $C_0 = Z^n$ (a more cumbersome but more precise notation is $\iota_0\{f(y)\}_{y \in Z^n}$). This approximation has the property that $(\iota_0 f)(z) = f(z), z \in Z^n$.

Starting from the expansion $\iota_0 f \in \mathcal{I}_0(\phi, Z^n)$ one can construct equivalent expansions, $\iota_k(J_k f) \in \mathcal{I}_k(\phi, Z^n)$. The coefficients $J_k f$ are referred to as the interpolet transform of the function f .

If $f(x)$ is sufficiently smooth, then we can expect that the coefficients $(J_k f)(y), y \in D_l, l \leq k$, will be small. This statement is captured rigorously by the following lemma and theorem.

Lemma 5.3.1 *Let ϕ be an interpolet with polynomial order of m then*

$$p(x) \in \mathcal{I}_N(\phi, C_N)$$

for any integer, N , and any polynomial, p , of degree m .

Proof: $p(2^N x)$ is a polynomial of degree m . By (INT3), $p(x)$ can thus be represented by a formal sum in $\mathcal{I}_0(\phi, C_0)$, namely

$$p(2^N x) = \sum_{y \in Z^n} p(2^N y) \phi(x - y).$$

By changing variables, we may rewrite this as

$$\begin{aligned} p(x) &= \sum_{y \in Z^n} p(2^N y) \phi(x/2^N - y) \\ &= \sum_{y \in Z^n} p(2^N y) \phi((x - 2^N y)/2^N) \\ &= \sum_{y \in C_N} p(y) \phi((x - y)/2^N) \end{aligned}$$

$$= \sum_{y \in C_N} p(y) \phi((x - y)/2^{\theta_N(y)}).$$

□

Theorem 5.3.2 (*Residual Theorem*)

Let $f(x)$ be a polynomial function of degree m . Let ϕ be an interpolet with a polynomial order of m . Then

$$(J_k f)(y) = \begin{cases} f(y), & y \in C_k \\ 0, & y \in D_l \end{cases}$$

Proof:

By (INT3), $f(x) \in \mathcal{I}_0(\phi, C_0)$. By the lemma, we also have $f(x) \in \mathcal{I}_k(\phi, C_k)$. Recalling that $J_k f$ gives the unique expansion coefficients of $f(x)$ in the decomposition of $\mathcal{I}_0(\phi, C_0)$ given by $\mathcal{I}_k(\phi, C_k) \oplus \mathcal{I}_k(\phi, D_k) \oplus \cdots \oplus \mathcal{I}_k(\phi, D_1)$, we see that the $\mathcal{I}_k(\phi, D_l)$ coefficients must vanish, while the $\mathcal{I}_k(\phi, C_k)$ coefficients are given by the lemma, namely $f(y)$ for $y \in C_k$.

□

The coefficients $(J_k f)(y), y \in D_l, l \leq k$, are called *residuals*. The Residual Theorem suggests that the magnitude of the residual coefficients $(J_k f)$ at a point $y \in D_l$ are expected to be proportional to the $(m + 1)$ th derivative of $f(x)$ at y . See Figure 5-2.

5.4 Truncated bases

Typically, one truncates an expansion by eliminating elements of a basis, setting their coefficients to 0. One is said to be working in a truncated basis when one works within the subspace formed by the remaining basis elements. In the notation of this chapter, this corresponds to taking expansions in $\mathcal{I}_k(\phi, S)$ with coefficients in $\mathcal{F}_k(S)$.

One may also view a truncated basis as the set of expansions one gets when the coefficients of some of the basis elements have been set to “don’t care” values. Mathematically, this is accomplished by quotienting out the “don’t care” elements.

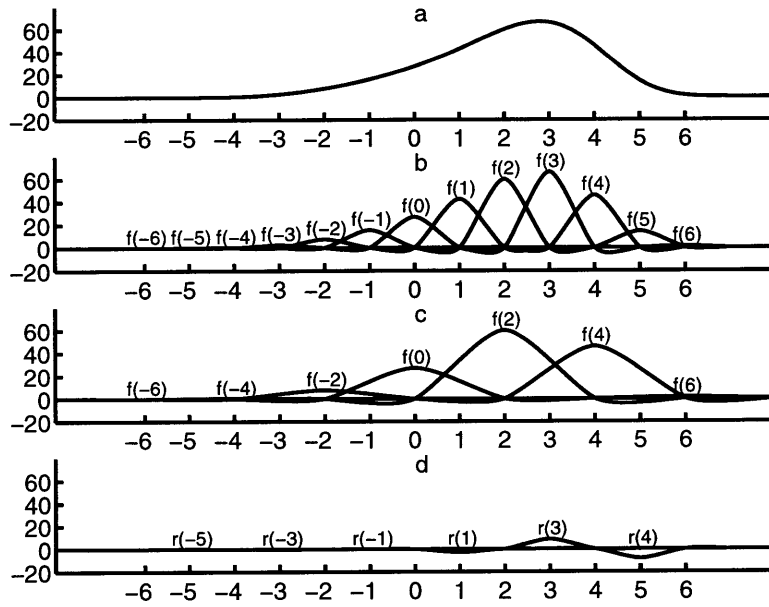


Figure 5-2: (a) the smooth function $f(x) = (\frac{x}{2} + 3)^3 e^{-(\frac{x}{4})^4}$. (b) the approximation to $f(x)$ in $\mathcal{I}_0(Z)$. (c) component of the approximation in $\mathcal{I}_1(C_1)$. (d) component of the approximation in $\mathcal{I}_1(D_1)$.

Definition 5.4.1 *Let*

$$\mathcal{I}_k^S = \mathcal{I}_k(\phi, Z^n) / \mathcal{I}_k(\phi, Z^n - S)$$

and

$$\mathcal{F}_k^S = \mathcal{F}_k(Z^n) / \mathcal{F}_k(Z^n - S).$$

When the identification of $F_k(S)$ with F_k^S can be made will be dealt with later in this chapter. For now, one may view these definitions as a trick to make the proofs less complicated and for understanding exactly why and in what sense the algorithms are correct. Once again, we think of \mathcal{F}_k^S as a grid on which the elements outside of S have “don’t care” values, and $\mathcal{F}_k(S)$ as a grid on which the elements outside of S vanish. The ι_k continue to be isomorphisms (since $\iota_k(\mathcal{F}_k(Z^n - S)) = \mathcal{I}_k(\phi, Z^n - S)$).

However, it is not necessarily true that $\mathcal{I}_{k_1}^S = \mathcal{I}_{k_2}^S$. When this condition fails, then it is no longer possible to define $J_{k_1, k_2} = \iota_{k_2}^{-1} \circ \iota_{k_1}$.

To be sure, one could still define some sort of J_{k_1, k_2} by setting the elements in $Z^n - S$ to zero, then applying the full grid version of J_{k_1, k_2} , and then considering only the elements

in S of the answer. This definition by itself has some drawbacks. Mathematically speaking, this is the same as $J_{k_1, k_2} = p \circ \iota_{k_2}^{-1} \circ \iota_{k_1} \circ r$, where $r : \mathcal{F}_{k_1}^S \rightarrow \mathcal{F}_{k_1}$, is some lift to the full grid, and $p : \mathcal{I}_{k_2} \rightarrow \mathcal{I}_{k_2}^S$ is the standard projection operator onto the quotient. Generally, if one follows this approach one will no longer have $J_{k_1, k_2} = J_{k_2-1, k_2} \circ J_{k_2-2, k_2-1} \circ \cdots \circ J_{k_1, k_1+1}$ because \mathcal{I}_i^S are not all equal. In terms of diagrams, where we once had

$$\mathcal{F}_{k_1} \xrightarrow{\iota_{k_1}} \mathcal{I}_{k_1} = \mathcal{I}_{k_2} \xrightarrow{\iota_{k_2}^{-1}} \mathcal{F}_{k_2},$$

we now have

$$\mathcal{F}_{k_1}^S \xrightarrow{\iota_{k_1}} \mathcal{I}_{k_1}^S \stackrel{?}{=} \mathcal{I}_{k_2}^S \xrightarrow{\iota_{k_2}^{-1}} \mathcal{F}_{k_2}^S.$$

What one needs is a condition on S such that $\mathcal{I}_{k_1}(Z^n - S) = \mathcal{I}_{k_2}(Z^n - S)$. If this were true, then the definition of the operator as $J_{k_1, k_2} = \iota_{k_2}^{-1} \circ \iota_{k_1}$ would actually be independent of the values of the elements of $Z^n - S$. In that case the quotient spaces are identical.

Definition 5.4.2 *We say that the set S is a good basis when it satisfies the condition $\forall k_1, k_2, \mathcal{I}_{k_1}(Z^n - S) = \mathcal{I}_{k_2}(Z^n - S)$, and thus $\mathcal{I}_{k_1}^S = \mathcal{I}_{k_2}^S$.*

To get a handle on this definition, one sees that this is achieved when $\mathcal{I}_k(Z^n - S) = \mathcal{I}_{k+1}(Z^n - S)$. For this to be so, whenever $y \in Z^n - S$, every z such that $\phi(x/2^{\theta_N(z)} - z)$ is in the two-scale expansion for $\phi(x/2^{\theta_N(y)} - y)$, must also be a member of $Z^n - S$.

This can be captured in the following table (in which we let $\theta_k(y) = \theta_k(z) + 1$).

in expansion?	$z \in S$	$z \in Z^n - S$
$y \in S$	ok	ok
$y \in Z^n - S$	not ok	ok

The good basis condition for fast synthesis and reconstruction has also been discovered by Cohen and Danchin (see *S-trees* in a coming work[17]) which appeared after the original submission of our manuscript.

For some of the algorithms presented, we may employ additional conditions based on the supports of the functions themselves (not just the support of their expansions).

Definition 5.4.3 *We say that functions f and g touch whenever $\text{supp}\{f\} \cap \text{supp}\{g\}$ has nonzero measure.*

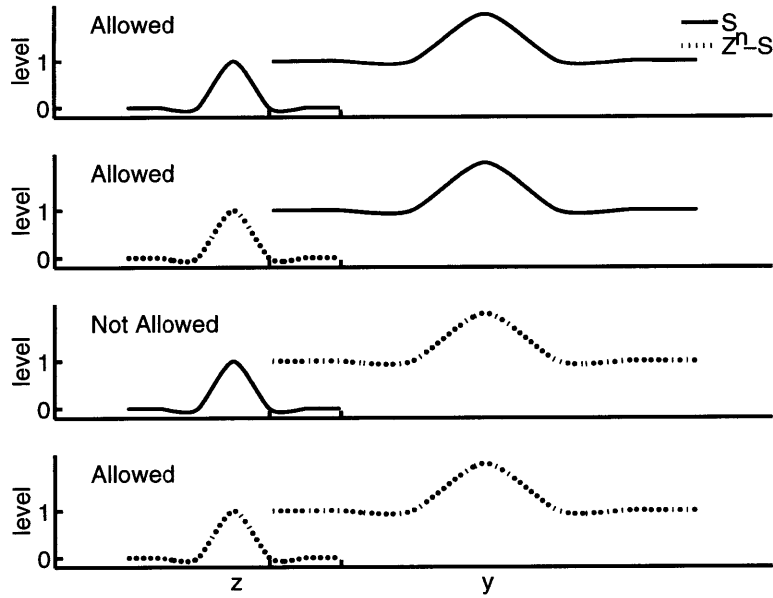


Figure 5-3: A visual summary of the 1-level touching condition. Solid plots represent functions centered at points in S . Dotted plots represent functions centered at points in $Z^n - S$. Tick marks delimit the overlap of the two functions.

For $p > 0$ we say that S has the p -level touching property when it satisfies the condition, that for $y \in Z^n - S$, $z \in Z^n$, $\theta_k(z) \leq \theta_k(y) - p$, and $\iota_k y$ touches $\iota_k z$ implies $z \in Z^n - S$.

A less formal way of phrasing this definition for the case of 1-level touching is that if a level l point, y , is a member of $Z^n - S$ then any point, z , at level $l - 1$ or lower for which $\iota_k y$ touches $\iota_k z$ must also be in $Z^n - S$. For 2-level touching, one only considers any points at level $l - 2$ or lower, and so on for p -level touching.

The allowed touching possibilities can be summarized the following table (in which we let $\theta_k(y) \geq \theta_k(z) + p$), or Figure 5-3 (for $p = 1$).

touch?	$z \in S$	$z \in Z^n - S$
$y \in S$	ok	ok
$y \in Z^n - S$	not ok	ok

One example of a 1-level touching good basis for one dimensional 3rd order interpolets

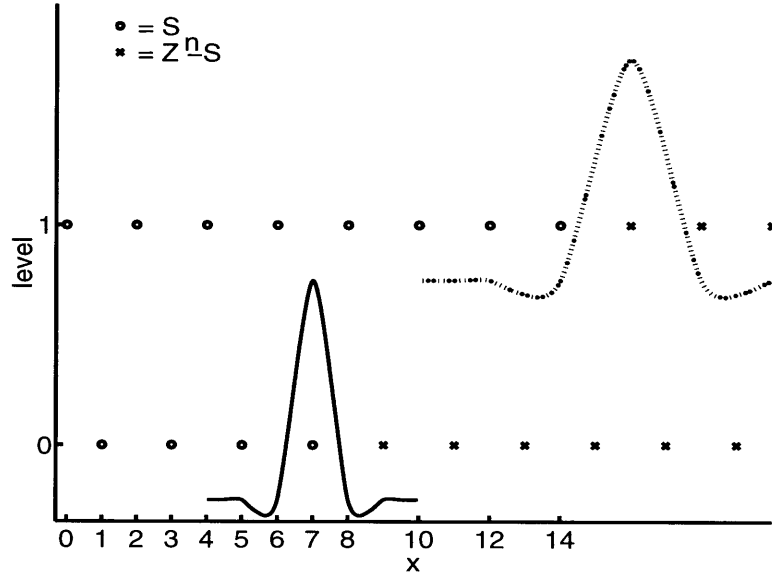


Figure 5-4: Our example of the 1-level touching good basis in one dimension. Note that the two functions plotted do not touch. Tick marks denote the set $S \subset Z$.

(the ones being used as an example) is the set of interpolets centered at the points

$$S = C_n \cup \left(\bigcup_{l=0}^{n-1} \{-7 \cdot 2^l, -5 \cdot 2^l, \dots, 5 \cdot 2^l, 7 \cdot 2^l\} \right)$$

(see Figure 5-4). The support of the interpolet on level 0 at y is $[-3 + y, 3 + y]$, the union of all the supports of the interpolets in S on level 0 is $[-10, 10]$. The support of an interpolet on level 1 at y is $[-6 + y, 6 + y]$, thus the only interpolets on level 1 which touch the interpolets on level 0 are precisely those ones at points $-14, \dots, 14$, which are precisely the ones included in S . No interpolet not included on level 1 touches an interpolet included on level 0 so the definition is satisfied. The argument proceeds similarly on higher levels. In three dimensions, this example corresponds to nested concentric cubes of size $15^3 \cdots 2^l$ at each level $l < n$. Figure 5-5 shows a randomly generated generic example.

An example of a 2-level touching good basis for 3rd order is the set of interpolets centered at the points

$$S = C_n \cup \left(\bigcup_{l=0}^{n-1} \{-5 \cdot 2^l, -3 \cdot 2^l, \dots, 3 \cdot 2^l, 5 \cdot 2^l\} \right).$$

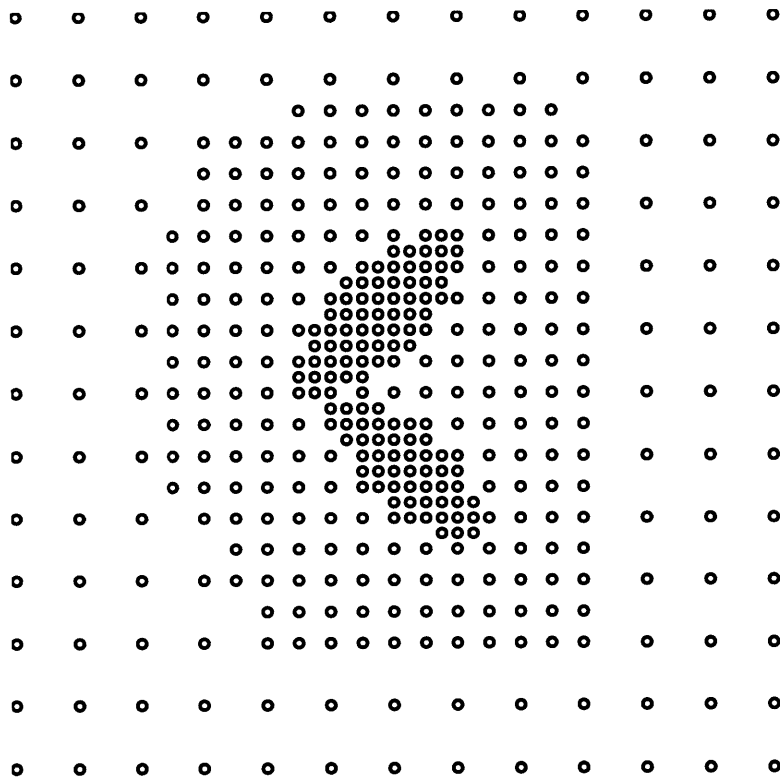


Figure 5-5: A generic example of a truncation which meets our definitions of good and 1-level touching. In black are the points of $S \subset \mathbb{Z}^n$.

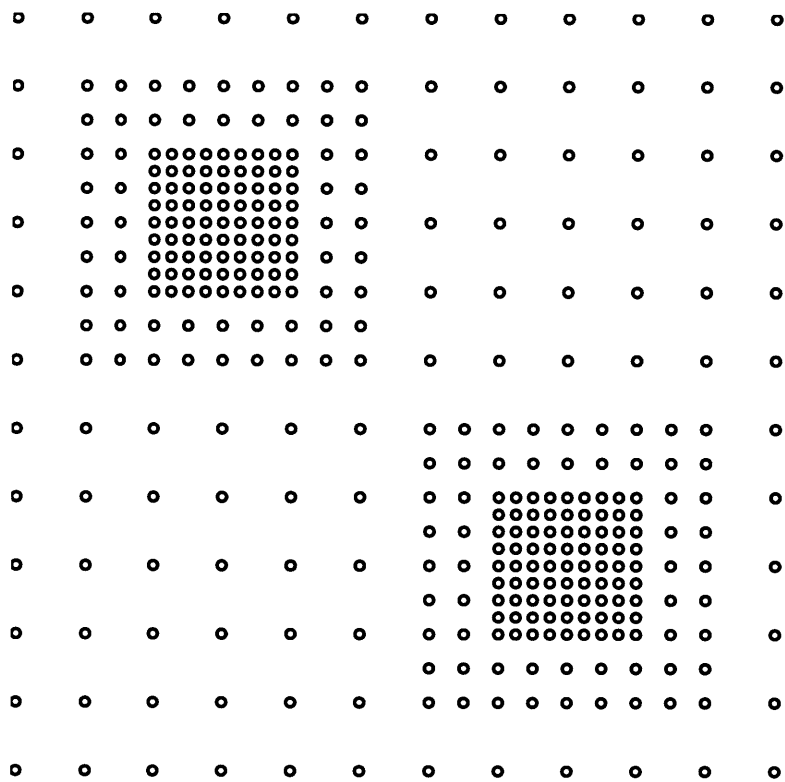


Figure 5-6: An example of a 2-level touching good basis which can be used for a diatomic molecule (two atomic cores). The points in $S \subset Z^n$ are where residual values could be significant.

Note (for $n \geq 2$) that this set is not 1-level touching since the level 1 interpolet centered at $y = 12$ is not included, while an interpolet it touches, namely the level 0 interpolet centered at $y = 5$, is included. Figure 5-6 shows an example of a 2-level touching good basis.

Finally, the set of points

$$S = C_n \cup \left(\bigcup_{l=0}^{n-1} \{-3 \cdot 2^l, -1 \cdot 2^l, 1 \cdot 2^l, 3 \cdot 2^l\} \right).$$

forms a good basis, but is not 1-level touching or 2-level touching, but it is 3-level touching.

The above examples are meant to suggest that the *good basis* and the *p-level touching* definitions can be thought of, informally, as conditions telling one how quickly one can change from one resolution to another. Essentially, any nested set of refined regions can satisfy these conditions so long as the margins around the set of points at a given resolution are wide

enough.

From a computational point of view, what these conditions do is ensure that data-paths which carry coefficient information between different resolutions are not broken by zeroed coefficients at intermediate levels.

It is clear from the preceding discussion, in a good basis, one has $J_{k_1, k_2} = J_{k_2-1, k_2} \circ J_{k_2-2, k_2-1} \circ \cdots \circ J_{k_1, k_1+1}$. We may now generalize the computation theorem to a truncated basis.

Theorem 5.4.4 *Good Basis Computation Theorem*

Let S be a good basis. $v \in \mathcal{F}_k(S)$, $y \in S$

and \tilde{v} is any member of the equivalence class of v

$$(J_{k, k+1} \tilde{v})(y) = \begin{cases} v(y), & y \in S - D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) - \sum_{z \in S \cap C_{k+1}} c_{(y-z)/2^k} v(z)$.

$$(J_{k+1, k} \tilde{v})(y) = \begin{cases} v(y), & y \in S - D_{k+1} \\ v'(y), & y \in D_{k+1} \end{cases}$$

where $v'(y) = v(y) + \sum_{z \in S \cap C_{k+1}} c_{(y+z)/2^k} v(z)$.

Proof:

In a good basis, the computations of all J_{k_1, k_2} are independent of the representative. Thus, this algorithm computed on \tilde{v} gives a member of the same class as would be computed on v . □

Thus, the pyramid algorithm of the uniform basis, Z^n , has a counterpart in a good basis S , allowing the computation of the expansion coefficients in $\mathcal{F}_k(S)$ from the values of the function in $\mathcal{F}_0(S)$ (and also has the lower triangular structure).

In a good basis, one thus has the ability to perfectly reconstruct the multiscale coefficients of a function for the basis functions associated with the points of the refined grid S by simply applying the pyramid algorithm on zero-lifts at each stage of the algorithm. The above

theorem establishes this as true, even though we do not necessarily expect the data zeroed during the lift to be small. (The function may have significant sample values throughout the domain of the representation). Also, with exact recovery of sample values, it is easy to perform local nonlinear point-wise operations of the form $f(x) \rightarrow G(f(x))$ (e.g. $e^{f(x)}$), or point-wise multiplication (i.e. $f(x), g(x) \rightarrow f(x)g(x)$), despite the truncation.

The reader may note that this result is "analysis-free" in that we have sparsified the computation, not by proving that the coefficients vanish outside of the truncation for some class of functions, but by showing the the coefficients we wish to compute have no dependency on the coefficients we omitted. Computationally, this means the data-structure in the computer requires no intermediate augmentations (contrast with [38]).

The advantages conferred by the additional the p -level properties are seen in the context of operator evaluation, and will be the subject of the next two sections.

5.5 Multilevel algorithms for ∇^2 and other operators

Given $v, w \in \mathcal{F}_k(Z^n)$, we may compute the stiffness matrix of a model operator, ∇^2 ,

$$\langle \iota_k v | \nabla^2 | \iota_k w \rangle = \int (\iota_k v)(x) \nabla^2 (\iota_k w)(x) d^n x$$

by changing the expansions,

$$\langle \iota_0 J_{-k} v | \nabla^2 | \iota_0 J_{-k} w \rangle = \sum_{y, z \in Z^n} (J_{-k} v)(y) (J_{-k} w)(z) \int \phi(x - y) \nabla^2 \phi(x - z) d^n x.$$

This reduces the computation to the computation of the matrix elements $\langle \phi(x - (y - z)) | \nabla^2 | \phi(x) \rangle$ which can be done by solving the associated eigen-problem obtained by applying (INT2).

In particular, let $L_y^0 = \langle \phi(x - y) | \nabla^2 | \phi(x) \rangle$, then

$$L_y^0 = \sum_{z_1, z_2 \in Z^n} c_{z_1} c_{z_2} \langle \phi(2x - 2y - z_1 + z_2) | \nabla^2 | \phi(2x) \rangle$$

$$L_y^0 = \sum_{z_1, z_2 \in Z^n} 2^{2-n} c_{z_1} c_{z_2} L_{2y+z_1-z_2}^0$$

which we solve by standard methods (found in [58], for example). In subsequent sections of this article, we will also define $L_y^+ = \langle \phi(x-y) | \nabla^2 | \phi(x/2) \rangle$, $L_y^{++} = \langle \phi(x-y) | \nabla^2 | \phi(x/4) \rangle$, $L_y^- = \langle \phi((x-y)/2) | \nabla^2 | \phi(x) \rangle$, and $L_y^{--} = \langle \phi((x-y)/4) | \nabla^2 | \phi(x) \rangle$, which can be computed from L^0 by employing (INT2). Although it is true by Hermiticity that $L_y^- = L_y^+$ and $L_y^{--} = L_y^{++}$, we will make no use of this fact.

One can write a matrix expression $\langle \iota_k v | \nabla^2 | \iota_k w \rangle = v^t J_{-k}^t L J_{-k} w$, where the L is the Toeplitz matrix with coefficients L_y^0 . In practice, one typically formulates the preceding as a computation of $u = J_{-k}^t D J_{-k} w$ for some $w \in \mathcal{F}_k(Z^n)$. Then $\langle \iota_k v | \nabla^2 | \iota_k w \rangle = v^t u$. u may be thought of as an element of $\mathcal{F}_k(Z^n)^*$, the dual space of $\mathcal{F}_k(Z^n)$. The task is then to compute the coefficients $u(y)$, $y \in Z^n = C_k \oplus D_k \oplus \dots \oplus D_1$. Any algorithm for computing $x^t A y$ can be adapted to an algorithm for $A y$, and for purposes of making proofs, it is somewhat easier to keep thinking of the computation as $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$, which is the point of view we shall take.

However, computing $\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \rangle = \tilde{v}^t J_{-k}^t L J_{-k} \tilde{w}$, by just applying the transforms, and the Toeplitz matrix is problematic, since this process makes it necessary to either represent v and w on a uniform grid, or to compute a matrix element between each pair of functions in the truncated expansion which touch. In the first case, one ends up with an $O(N)$ computation for (typically) a very large N . In the second case, one chooses between extremes which are $O(N)$ and quite complicated or simple and $O(N^2)$.

The following sections outline the design of multilevel algorithms for ∇^2 for both 1-level and 2-level touching bases. Both algorithms are derived according to the following format:

break up the expansion of $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$ into a decomposition over elements at the same level and adjacent levels.

rewrite the expansions in terms of the matrix elements between elements of those levels and the transforms of higher/lower level elements.

implement the algorithm by computing those terms separately.

establish correctness in a p -level truncated basis.

Although only the 1-level and 2-level algorithms have been explored in any detail, this same process will generally work to produce $O(N)$ p -level algorithms for any p .

5.5.1 ∇^2 in 1-level decomposition

The 1-level decomposition of $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$ is

$$\langle \iota_k v | \nabla^2 | \iota_k w \rangle = \langle \iota_k v | \nabla^2 | \iota_k w \rangle^+ + \langle \iota_k v | \nabla^2 | \iota_k w \rangle^0 + \langle \iota_k v | \nabla^2 | \iota_k w \rangle^-,$$

where

$$\begin{aligned} \langle \iota_k v | \nabla^2 | \iota_k w \rangle^0 &= \sum_{\theta_k(y)=\theta_k(z)} \langle \iota_k y | \nabla^2 | \iota_k z \rangle v(y)w(z) \\ \langle \iota_k v | \nabla^2 | \iota_k w \rangle^+ &= \sum_{\theta_k(y)<\theta_k(z)} \langle \iota_k y | \nabla^2 | \iota_k z \rangle v(y)w(z) \\ \langle \iota_k v | \nabla^2 | \iota_k w \rangle^- &= \sum_{\theta_k(y)>\theta_k(z)} \langle \iota_k y | \nabla^2 | \iota_k z \rangle v(y)w(z). \end{aligned}$$

That is, we express the product as contributions from levels to the same level, higher levels, and lower levels. We will now investigate each of these terms individually. In the language of matrices, these respectively correspond to diagonal, above diagonal, and below diagonal blocks of the ∇^2 matrix.

$$\langle \iota_k v | \nabla^2 | \iota_k w \rangle = v^t \begin{pmatrix} \langle | \rangle^0 & \langle | \rangle^+ & \langle | \rangle^+ & \langle | \rangle^+ \\ \langle | \rangle^- & \langle | \rangle^0 & \langle | \rangle^+ & \langle | \rangle^+ \\ \langle | \rangle^- & \langle | \rangle^- & \langle | \rangle^0 & \langle | \rangle^+ \\ \langle | \rangle^- & \langle | \rangle^- & \langle | \rangle^- & \langle | \rangle^0 \end{pmatrix} w$$

Diagonal blocks: $\langle | \rangle^0$

For some fixed $l, 0 \leq l \leq k$ a term of the form

$$\sum_{\theta_k(z)=\theta_k(y)=l} \langle \phi((x-y)/2^l) | \nabla^2 | \phi((x-z)/2^l) \rangle v(y)w(z)$$

contributes to $\langle | \rangle^0$.

$$\text{However, } \langle \phi((x-y)/2^l) | \nabla^2 | \phi((x-z)/2^l) \rangle = 2^{l(n-2)} \langle \phi(x-y) | \nabla^2 | \phi(x-z) \rangle = 2^{l(n-2)} L_{z-y}^0.$$

Thus we have

$$\langle \iota_k v | \nabla^2 | \iota_k w \rangle^0 = \sum_l 2^{l(n-2)} \sum L_{z-y}^0 v(y) w(z).$$

Super-diagonal blocks: $\langle | \rangle^+$

For some fixed $l, 0 \leq l \leq k$ a term of the form

$$\sum_{\theta_k(z)=l' > l = \theta_k(y)} \langle \phi((x-y)/2^l) | \nabla^2 | \phi((x-z)/2^{l'}) \rangle v(y) w(z)$$

contributes to $\langle | \rangle^+$.

Applying the inverse transform $(J_{k,l+1})$ to the w coefficients in the sum allows us to write this term as

$$\sum_{\theta_{l+1}(z)=l+1 > l = \theta_k(y)} \langle \phi((x-y)/2^l) | \nabla^2 | \phi((x-z)/2^{l+1}) \rangle v(y) (J_{k,l+1} w)(z).$$

Thus we have

$$\langle \iota_k v | \nabla^2 | \iota_k w \rangle^+ = \sum_l 2^{l(n-2)} \sum L_{z-y}^+ v(y) (J_{k,l+1} w)(z).$$

Sub-diagonal blocks: $\langle | \rangle^-$

For some fixed $l, 0 \leq l \leq k$ a term of the form

$$\sum_{\theta_k(z)=l < l' = \theta_k(y)} \langle \phi((x-y)/2^{l'}) | \nabla^2 | \phi((x-z)/2^l) \rangle v(y) w(z)$$

contributes to $\langle | \rangle^-$.

This time applying the inverse transform $(J_{k,l+1})$ to the v coefficients in the sum allows us to write this term as

$$\sum_{\theta_k(z)=l < l+1 = \theta_{l+1}(y)} \langle \phi((x-y)/2^{l+1}) | \nabla^2 | \phi((x-z)/2^l) \rangle (J_{k,l+1} v)(y) w(z).$$

Thus we have

$$\langle \iota_k v | \nabla^2 | \iota_k w \rangle^- = \sum_l 2^{l(n-2)} \sum L_{z-y}^-(J_{k,l+1} v)(y) w(z).$$

Implementations

The above observations demonstrate the correctness of the following algorithm:

input: $v, w \in \mathcal{F}_k(Z^n)$

output: $ans = \langle \iota_k v | \nabla^2 | \iota_k w \rangle \in R$

Let $wtmp = w$, let $vtmp = v$, let $ans = 0$

for $l = 0$ to k

$$ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=\theta_k(z)=l} L_{z-y}^0 v(y) w(z)$$

end for

for $l = k - 1$ down-to 0

$$ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=l, \theta_{l+1}(z)=l+1} L_{z-y}^+ v(y) wtmp(z)$$

$$wtmp \leftarrow J_{l+1,l}(wtmp)$$

end for

for $l = k - 1$ down-to 0

$$ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_{l+1}(y)=l+1, \theta_k(z)=l} L_{z-y}^- vtmp(y) w(z)$$

$$vtmp \leftarrow J_{l+1,l}(vtmp)$$

end for

Note, we have made use of the fact that $J_{k,l} = J_{l+1,l} \cdots J_{k,k-1}$ so that at the beginning of each iteration in the second (last) loop $wtmp = J_{k,l+1} w$ ($vtmp = J_{k,l+1} v$).

We observe that this algorithm is $O(N)$ in time and space.

We may adapt this to an algorithm to compute $u \in \mathcal{F}_k(Z^n)^*$ such that $u(v) = \langle \iota_k v | \nabla^2 | \iota_k w \rangle$.

input: $w \in \mathcal{F}_k(Z^n)$

output: u such that $u(v) = \langle \iota_k v | \nabla^2 | \iota_k w \rangle$

Let $wtmp = w$, let $u = 0 \in \mathcal{F}_k(Z^n)^*$, let $utmp = 0 \in \mathcal{F}_0(Z^n)^*$

for $l = 0$ to k

$$u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=\theta_k(z)=l} L_{z-y}^0 w(z)$$

end for

for $l = k - 1$ down-to 0

$$u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=l, \theta_{l+1}(z)=l+1} L_{z-y}^+ wtmp(z)$$

$$wtmp \leftarrow J_{l+1,l} wtmp$$

end for

for $l = 0$ to $k - 1$

$$utmp \leftarrow J_{l+1,l}^t utmp$$

$$utmp(y) \leftarrow utmp(y) + 2^{l(n-2)} \sum_{\theta_{l+1}(y)=l+1, \theta_k(z)=l} L_{z-y}^- w(z)$$

end for

$$u \leftarrow u + utmp$$

The third loop is the result of transposing the linear operator $J_{l+1,l}$ in the last loop in the previous algorithm. We have also used the fact that $J_{k,l}^t = J_{k,k-1}^t \cdots J_{l+1,l}^t$ to ensure that at the beginning of each iteration in the third loop, $utmp \in \mathcal{F}_{l+1}(Z^n)^*$.

It is easy to check that this is an $O(N)$ algorithm in both time and space. It is very important for atomic structures computation that this algorithm scales linearly with the number of atoms. Without such a scaling, one can only compute electronic configurations for small molecules.

The reader may note a similarity between this algorithm and other matrix-vector multiplies used to apply operators in a uniform wavelet basis. In fact, the 1-level algorithm presented above is identical to the nonstandard multiply found in [8] and developed for orthonormal wavelet bases. The nonstandard multiply was introduced by Beylkin, Coifman, and Rokhlin to sparsify integral operators whose kernels were smooth or vanishing off the diagonal, while keeping a uniform basis.

However, in contrast to that program of sparsification, interpolets allow one to sparsify the basis, and, with out introducing additional grid points, still be able to apply the nonstandard multiply routines, with any local operator. With interpolets, we remove any elements from the expansion that we believe will be insignificant, still having a good approximation to our function at the points we retain. Beylkin *et al.* [8] express the matrix elements of the operator itself in a nonstandard orthonormal basis and then remove those matrix elements which are determined to be very small to produce a sparse matrix.

The use of interpolating scaling functions has achieved some degree of simplicity and

convenience in carrying out fast point-wise operations. Although there is no associated difficulty in electronic structure calculations[3], for other applications, the loss of orthogonality might be too great an expense. In those cases, one might consider employing compactly supported approximations to orthogonal interpolating functions found in [9]. It appears that with some additional complexity one might be able to extend the present algorithms to other wavelet bases. The additional complexity of other schemes and the need for fast point-wise operations in our applications are the chief reasons we do not consider doing this in the present work. Finally, there is a large body of work the reader may wish to consult ([10], [35], [18], [46], and [47]) for adaptive refinement techniques when, in contrast to the case of electronic structure calculations, the behavior of the needed refinement is not known *a priori*.

Correctness in a 1-level touching good basis

The above decomposition of the product and the associated algorithm is what we seek to extend to a good truncated basis. In practice, one takes the zero-lift representatives of v and $w \in \mathcal{F}_k^S$ and computes $\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \rangle$. By the computation theorem of good truncated bases, the value of $(J_{k_1, k_2} \tilde{v})(y)$, $y \in \mathcal{F}_k^S$ is independent of the representative (likewise for w), however, we must also address the issue that $(J_{k_1, k_2} \tilde{v})(y) \neq 0$, $y \notin S$ (i.e. $J_{k_1, k_2} \tilde{v} \neq \widetilde{J_{k_1, k_2} v}$), and thus $y \notin S$ may have a contribution to the decomposition above, requiring us to augment S in order to get the right answer.

Theorem 5.5.1 *If one replaces Z^n with S everywhere in the Multilevel algorithm for $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$, then the algorithm computes*

$$\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \rangle.$$

Proof: As mentioned in the remarks, the multilevel algorithm requires $J_{k, l} = J_{l+1, l} \cdots J_{k, k-1}$, which is true in a good bases.

The $\langle | \rangle^0$ computation proceeds identically for either Z^n or S , so the first loop will contribute correctly the term $\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \rangle^0$.

To check the $\langle \cdot \rangle^+$ contribution, one observes that the necessary term is

$$\langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \rangle^- = \sum_l 2^{l(n-2)} \sum_{y \in Z^n, z \in Z^n: \theta_{l+1}(y)=l+1, \theta_k(z)=l} L_{z-y}^-(J_{k,l+1} \tilde{v})(y) \tilde{w}(z).$$

Suppose that $\exists y \notin S$ and $z \in S$ such that $L_{z-y}^+ \neq 0$. This implies that $\text{supp}\{\iota_{l+1}y\} \cap \text{supp}\{\iota_k z\}$ has nonzero measure. Since $\text{supp}\{\iota_{l+1}y\} \subset \text{supp}\{\iota_k y\}$ we conclude that $\text{supp}\{\iota_k y\} \cap \text{supp}\{\iota_k z\}$ has nonzero measure. This cannot be so if S is 1-level touching, and since $\tilde{w}(z) = 0, z \notin S$ we may restrict the sums over y and z in the contribution,

$$2^{l(n-2)} \sum_{y \in S, z \in S: \theta_{l+1}(y)=l+1, \theta_k(z)=l} L_{z-y}^-(J_{k,l+1} \tilde{v})(y) \tilde{w}(z).$$

The proof for $\langle \cdot \rangle^-$ is identical with v 's and w 's reversed. □

Immediately we have the following:

Corollary 5.5.2 *If one replaces Z^n with S everywhere in the Multilevel algorithm for u such that $u(v) = \langle \iota_k v | \nabla^2 | \iota_k w \rangle$, then the algorithm computes*

$$u \in (\mathcal{F}_k^S)^*, u(v) = \langle \iota_k \tilde{v} | \nabla^2 | \iota_k \tilde{w} \rangle.$$

The computation of $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$ serves as a template for another common computation one may wish to perform, namely $\langle \iota_k v | \iota_k w \rangle = \int (\iota_k v)(x) (\iota_k w)(x) d^n x$, i.e. the $L^2(R^n)$ inner product of $\iota_k v$ and $\iota_k w$.

5.5.2 Computing other operators

To compute $\langle \iota_k \tilde{v} | \iota_k \tilde{w} \rangle$, one simply replaces L^0, L^+ , and L^- with $G_y^0 = \langle \phi(x-y) | \phi(x) \rangle$, $G_y^+ = \langle \phi(x-y) | \phi(x/2) \rangle$, and $G_y^- = \langle \phi((x-y)/2) | \phi(x) \rangle$, and then replaces the factors of $2^{l(n-2)}$ with 2^{ln} . After that, the algorithms and theorems for ∇^2 carry over directly.

The above procedure can be used for creating a multilevel algorithm for any operator, \mathcal{O} , which is

local : $\text{supp}\{\mathcal{O}f\} \subset \text{supp}\{f\}$

translation invariant : $\mathcal{O}f(x+a) = (\mathcal{O}f)(x+a)$

homogeneous : $\mathcal{O}f(sx) = s^\alpha(\mathcal{O}f)(sx)$

by forming the appropriate coefficients, $\mathcal{O}_y^{\{0,+,-\}}$, and inserting appropriate factors of $2^{l(\alpha+d)}$.

However, locality is the only property which is really required for $O(N)$ multilevel algorithms so long as one can compute $\mathcal{O}_{l,m,m'}^{\{0,+,-\}}$ efficiently.

As an example of a local, homogeneous, but not translationally invariant operator, we shall discuss the coefficients for the multilevel algorithm for the \hat{x}_1 operator in two dimensions.

We first consider the coefficients $\hat{x}_{1l,m,m'}^{\{0,+,-\}}$ given by

$$\hat{x}_{1l,m,m'}^{\{0,+,-\}} = \int \phi((x_1-m_1)/2^{l(+1)})\phi((x_2-m_2)/2^{l(+1)})x_1\phi((x_1-m'_1)/2^{l(+1)})\phi((x_2-m'_2)/2^{l(+1)})dx_1dx_2,$$

i.e.

$$\begin{aligned}\hat{x}_{1l,m,m'}^0 &= \int \phi((x_1-m_1)/2^l)\phi((x_2-m_2)/2^l)x_1\phi((x_1-m'_1)/2^l)\phi((x_2-m'_2)/2^l)dx_1dx_2 \\ \hat{x}_{1l,m,m'}^+ &= \int \phi((x_1-m_1)/2^l)\phi((x_2-m_2)/2^l)x_1\phi((x_1-m'_1)/2^{l+1})\phi((x_2-m'_2)/2^{l+1})dx_1dx_2 \\ \hat{x}_{1l,m,m'}^- &= \int \phi((x_1-m_1)/2^{l+1})\phi((x_2-m_2)/2^{l+1})x_1\phi((x_1-m'_1)/2^l)\phi((x_2-m'_2)/2^l)dx_1dx_2.\end{aligned}$$

Separating the x_1 and x_2 integrations, we see from this that we may write $\hat{x}_{1l,m,n}^{\{0,+,-\}} = 2^{l(d-1)}G_{(m_1-n_1)/2^l}^{\{0,+,-\}}X_{l,m_2,n_2}^{\{0,+,-\}}$ where

$$X_{l,m,n}^{\{0,+,-\}} = \int \phi((x-m)/2^{l(+1)})x\phi((x-n)/2^{l(+1)})dx$$

and G is defined above. The problem of computing the \hat{x}_1 coefficients has been reduced to computing the X coefficients and then doing a multiply with the already known G coefficients.

In addition, one has

$$\int \phi((x - m)/2^{l(+1)})x\phi((x - n)/2^{l(+1)})dx = n \int \phi((x + n - m)/2^{l(+1)})\phi(x/2^{l(+1)})dx +$$

$$\int \phi((x + n - m)/2^{l(+1)})x\phi(x/2^{l(+1)})dx,$$

and thus

$$\int \phi((x - m)/2^{l(+1)})x\phi((x - n)/2^{l(+1)})dx = n2^l G_{(m-n)/2^l}^{\{0,+,-\}} + 2^{2l} S_{(m-n)/2^l}^{\{0,+,-\}}$$

where

$$S_y^0 = \int \phi(x - y)x\phi(x)dx$$

$$S_y^+ = \int \phi(x - y)x\phi(x/2)dx$$

$$S_y^- = \int \phi((x - y)/2)x\phi(x)dx.$$

Thus we see that for the \hat{x}_1 operator,

$$\hat{x}_{1,m,n}^{\{0,+,-\}} = 2^l G_{(m_2-n_2)/2^l}^{\{0,+,-\}} (n2^l G_{(m-n)/2^l}^{\{0,+,-\}} + 2^{2l} S_{m-n}^{\{0,+,-\}}),$$

giving an efficient means to compute the multilevel coefficient for this operator.

5.5.3 ∇^2 in 2-level decomposition

The previous algorithm for 1-level touching good bases can be expanded to 2-level touching good bases. One may wish to do this because one finds that the 1-level touching property is too stringent and requires one to augment one's basis set far too much to be practical computationally.

Much of the reasoning for the 2-level case can be found in the details of the 1-level case, so the exposition here will be more compact. The resulting algorithm will be correct for 2-level touching good bases.

The 2-level decomposition of $\langle \iota_k v | \nabla^2 | \iota_k w \rangle$ is

$$\begin{aligned} \langle \iota_k v | \nabla^2 | \iota_k w \rangle &= \langle \iota_k v | \nabla^2 | \iota_k w \rangle^0 + \langle \iota_k v | \nabla^2 | \iota_k w \rangle^+ + \langle \iota_k v | \nabla^2 | \iota_k w \rangle^- \\ &\quad + \langle \iota_k v | \nabla^2 | \iota_k w \rangle^{++} + \langle \iota_k v | \nabla^2 | \iota_k w \rangle^{--} \end{aligned}$$

where

$$\begin{aligned} \langle \iota_k v | \nabla^2 | \iota_k w \rangle^0 &= \sum_{\theta_k(y)=\theta_k(z)} \langle \iota_k y | \nabla^2 | \iota_k z \rangle v(y)w(z) \\ \langle \iota_k v | \nabla^2 | \iota_k w \rangle^+ &= \sum_{\theta_k(y)+1=\theta_k(z)} \langle \iota_k y | \nabla^2 | \iota_k z \rangle v(y)w(z) \\ \langle \iota_k v | \nabla^2 | \iota_k w \rangle^- &= \sum_{\theta_k(y)-1=\theta_k(z)} \langle \iota_k y | \nabla^2 | \iota_k z \rangle v(y)w(z) \\ \langle \iota_k v | \nabla^2 | \iota_k w \rangle^{++} &= \sum_{\theta_k(y)+1 < \theta_k(z)} \langle \iota_k y | \nabla^2 | \iota_k z \rangle v(y)w(z) \\ \langle \iota_k v | \nabla^2 | \iota_k w \rangle^{--} &= \sum_{\theta_k(y)-1 > \theta_k(z)} \langle \iota_k y | \nabla^2 | \iota_k z \rangle v(y)w(z) \end{aligned}$$

Which corresponds to the matrix decomposition:

$$\langle \iota_k v | \nabla^2 | \iota_k w \rangle = v^t \begin{pmatrix} \langle | \rangle^0 & \langle | \rangle^+ & \langle | \rangle^{++} & \langle | \rangle^{++} \\ \langle | \rangle^- & \langle | \rangle^0 & \langle | \rangle^+ & \langle | \rangle^{++} \\ \langle | \rangle^{--} & \langle | \rangle^- & \langle | \rangle^0 & \langle | \rangle^+ \\ \langle | \rangle^{--} & \langle | \rangle^{--} & \langle | \rangle^- & \langle | \rangle^0 \end{pmatrix} w.$$

The key idea is to evaluate the diagonal and first off diagonal blocks of the ∇^2 matrix and then to compute the other blocks above and below the tridiagonal through the transforms.

$\langle | \rangle^0$, $\langle | \rangle^+$, $\langle | \rangle^-$, $\langle | \rangle^{++}$, **and** $\langle | \rangle^{--}$

The definitions for the contributions in the decomposition proceed just as they did for the 1-level case.

$$\langle \iota_k v | \nabla^2 | \iota_k w \rangle^0 = \sum_l 2^{l(n-2)} \sum_{\theta_k(z)=\theta_k(y)=l} L_{z-y}^0 v(y)w(z)$$

$$\begin{aligned}
\langle \iota_k v | \nabla^2 | \iota_k w \rangle^+ &= \sum_l 2^{l(n-2)} \sum_{\theta_k(z)-1=\theta_k(y)=l} L_{z-y}^+ v(y) w(z) \\
\langle \iota_k v | \nabla^2 | \iota_k w \rangle^- &= \sum_l 2^{l(n-2)} \sum_{\theta_k(z)=\theta_k(y)-1=l} L_{z-y}^- v(y) w(z) \\
\langle \iota_k v | \nabla^2 | \iota_k w \rangle^{++} &= \sum_l 2^{l(n-2)} \sum_{\theta_k(z)-1>l=\theta_k(y)} L_{z-y}^{++} v(y) (J_{k,l+2} w)(z) \\
\langle \iota_k v | \nabla^2 | \iota_k w \rangle^{--} &= \sum_l 2^{l(n-2)} \sum_{\theta_k(z)=l<\theta_k(y)-1} L_{z-y}^{--} (J_{k,l+2} v)(y) w(z)
\end{aligned}$$

Implementations

input: $v, w \in \mathcal{F}_k(Z^n)$

output: $ans = \langle \iota_k v | \nabla^2 | \iota_k w \rangle \in R$

Let $wtmp = w$, let $vtmp = v$, let $ans = 0$

for $l = 0$ to k

$$ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=\theta_k(z)=l} L_{z-y}^0 v(y) w(z)$$

end for

for $l = 0$ to $k - 1$

$$ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=l, \theta_k(z)=l+1} L_{z-y}^+ v(y) w(z)$$

end for

for $l = 0$ to $k - 1$

$$ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=l+1, \theta_k(z)=l} L_{z-y}^- v(y) w(z)$$

end for

for $l = k - 2$ down-to 0

$$ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_k(y)=l, \theta_{l+2}(z)=l+2} L_{z-y}^{++} v(y) wtmp(z)$$

$$wtmp \leftarrow J_{l+1, l} wtmp$$

end for

for $l = k - 2$ down-to 0

$$ans \leftarrow ans + 2^{l(n-2)} \sum_{\theta_{l+2}(y)=l+2, \theta_k(z)=l} L_{z-y}^{--} vtmp(y) w(z)$$

$$vtmp \leftarrow J_{l+1, l} vtmp$$

end for

We adapt this algorithm to compute $u \in \mathcal{F}_k(Z^n)^*$ such that $u(v) = \langle \iota_k v | \nabla^2 | \iota_k w \rangle$.

input: $w \in \mathcal{F}_k(Z^n)$

output: u such that $u(v) = \langle \iota_k v | \nabla^2 | \iota_k w \rangle$

Let $wtmp = w$, let $u = 0 \in \mathcal{F}_k(Z^n)^*$, let $utmp = 0 \in \mathcal{F}_0(Z^n)^*$

for $l = 0$ to k

$$u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=\theta_k(z)=l} L_{z-y}^0 w(z)$$

end for

for $l = 0$ to $k - 1$

$$u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=l, \theta_k(z)=l+1} L_{z-y}^+ w(z)$$

end for

for $l = 0$ to $k - 1$

$$u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=l+1, \theta_k(z)=l} L_{z-y}^- w(z)$$

end for

for $l = k - 2$ down-to 0

$$u(y) \leftarrow u(y) + 2^{l(n-2)} \sum_{\theta_k(y)=l, \theta_{l+2}(z)=l+2} L_{z-y}^{++} wtmp(z)$$

$$wtmp \leftarrow J_{l+1, l} wtmp$$

end for

for $l = 0$ to $k - 2$

$$utmp \leftarrow J_{l+1, l}^t utmp$$

$$utmp(y) \leftarrow utmp(y) + 2^{l(n-2)} \sum_{\theta_{l+2}(y)=l+2, \theta_k(z)=l} L_{z-y}^{--} w(z)$$

end for

$$u \leftarrow u + utmp$$

5.6 Efficient implementation

We have produced a very successful 3D implementation of all of the above algorithms for the interpolant used as this chapter's example ($m = 3$). Implementation details are given in this section. The ideas used to make this implementation efficient for all of the above algorithms.

The purpose of this section is to give additional information to readers who wish to implement these algorithms themselves.

5.6.1 Data structures

The interpolet data and function samples are kept in a sequence of blocks at various levels. Each block at level k contains the points of a rectangular subset of C_{k-1} . Since $D_k = C_{k-1} - C_k$, we use the collection of blocks at level $k < p$ (p being the top level) to represent a rectangular subset O_k , ignoring the C_k points of each of these blocks. In our implementation, these extra C_k points hold the value 0 in between operations and take on useful intermediate values during operations. Since we are working in 3 dimensions, this multiplies the storage required by a factor of about $\frac{8}{7}$, which we found an acceptable cost for its advantages.

The coefficients for the transforms and operators are kept in various 3D arrays. Although it is possible to build the coefficients upon demand from a set of 1D arrays of coefficients, we have found that the arithmetic cost of doing this is much greater than the cost of storing them (about 10 flops are required for each ∇^2 coefficient, while the 3D arrays are still small enough to be stored in cache). We have (in Fortran notation) the filters `cs(0:3,0:3,0:3)`, `SAMELEVEL(0:5,0:5,0:5)`, `ONELEVEL(0:8,0:8,0:8)`, and (for 2-level algorithms) `TWOLEVEL(0:14,0:14,0:14)` (note: we have made use of the fact that our operators are symmetric to cut the size of these arrays by a factor of $\frac{1}{8}$ and use `ONELEVEL` and `TWOLEVEL` for both upward and downward inter-level communication).

5.6.2 Implementation overview

The blocks described in the previous section are used as the fundamental objects for manipulation. The computation proceeds by employing block-to-block subroutines for the various operations, having every block at a level send data to every block at the same level, one level up or down, or (for 2-level algorithms) two levels up or down.

The number of blocks at each level is not very large, and if a subroutine determines that the intersection of two blocks is empty (which it does by examining the bounding rectangles), then it returns immediately. Thus, while this algorithm is to be $O(B^2)$ where B is the number of blocks, it remains $O(N)$ where N is the number of actual points, because B is much smaller than N .

5.6.3 Block-to-block subroutines

The block-to-block subroutines are all designed to take two blocks (source and destination) and a set of filter coefficients and place the result of convolving the filter with the source block in the overlapping points of the destination block. There is a block-to-block subroutine for the interpolet transform, its transpose, its inverse, and its inverse transpose, as well as operator application routines for the same-level operator, up-one-level operator, down-one-level operator, up-two-level operator, and the down-two-level operator.

All of these routines precompute the bounding box for the points in the destination block which are in the range of influence of the source block and, for each point in this sub-block, the bounding box for the points in the source block in the domain of influence of the destination point. The result of this precomputation is that the only data values of the source (destination) which are accessed are the ones which are read (modified). This decreases the number of data accesses in our test problems by a factor of 7.

Additionally, blocking the computation generally increases the locality of access for the data. More data requests hit the cache than would occur in a more arbitrarily arranged construction.

5.7 Results and conclusions

With interpolets, it is possible to carry out $O(N)$ computations in truncated bases, where N is the number of elements retained in the truncation, without having to augment the grid of points associated with the functions maintained in the basis. Along with allowing one to compute common linear physical operations, interpolet algorithms also allow one to transfer between function values and multiscale expansion coefficients on grids of variable resolution recovering the same results as one would obtain working with data on a full grid of arbitrary resolution but without introducing additional grid points into the calculation. This allows local nonlinear couplings to be computed quickly without the introduction of additional sample points and without the introduction of additional approximations which must be controlled.

These algorithms have been implemented in Fortran90 and have subsequently been adopted for use in electronic structure computations as described in the introduction. Prior to this, we had been using very simple, naïve $O(N^2)$ algorithms which implement each transform and operator as multiplication by the multiscale representation of the corresponding matrix. These multiplies check all points for being within interaction range and then construct the appropriate matrix element as needed. This is required in the naïve approach because the variety of inter-scale matrix elements is too wide to store in table of reasonable size. This algorithm ultimately scales quadratically with the number of refinement levels for our application. This is because, as described in the introduction, basis functions are kept in the basis whenever they contain an atomic nucleus within their support. All functions in this subset of significant size of the basis functions associated with each atomic center therefore touch one another, and the multiscale matrices contain dense blocks connecting all of these elements of a given center with one another. Because the number of functions associated with a given center grows linearly with the number of refinement scales k , the number of operations required in the naïve approach of multiplying directly by these dense matrices scales quadratically with the number of functions in the basis. For reference, a typical number of refinement levels in electronic structure calculations of the lighter elements would be $k = 5$, as employed in the carbon atom [5] and the nitrogen molecule [3].

A comparison with the previous implementation in Fortran90 on the same processor (Superscalar SPARC Version 9, UltraSPARC) demonstrates the speed improvements and scaling which can be achieved with the new approach. The “time” axis is the CPU time taken by one application of the ∇^2 operator. The “k” axis represents the number of levels of refinement made in the basis and is proportional to the number of points in S .

Figure 5-7 compares the runtimes of ∇^2 in three dimensions on a 1-level touching good basis with 3rd order interpolets consisting of concentric cubes of size 15^3 centered about one atomic nucleus, as would be appropriate for the calculation of the electronic structure of a single atom. Although there is initially a significant $O(N)$ contribution, as a function of the number of refinement levels k , the times for the naïve approach show the constant increments in slope characteristic of a quadratic function. The new approach compares very favorably and is about 40 times faster for typical values of k . (Note the difference in vertical

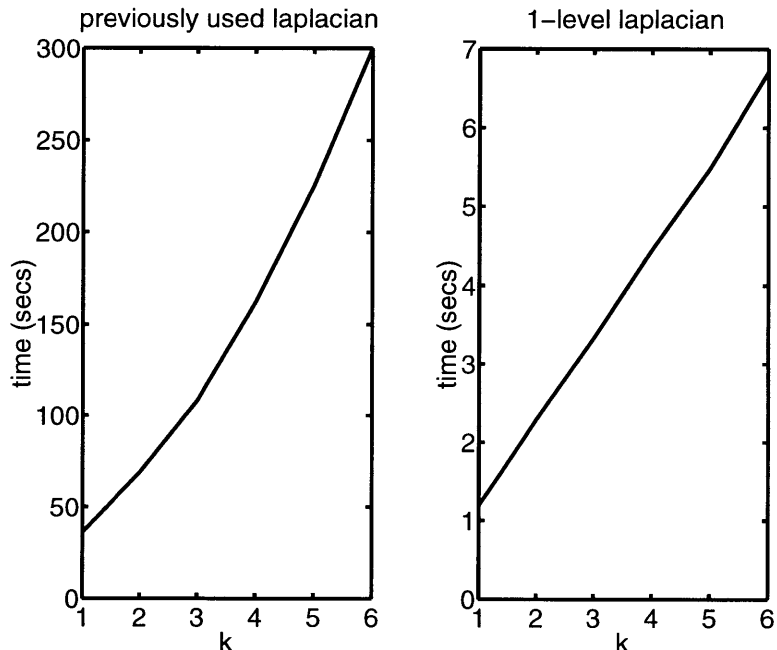


Figure 5-7: The previously used implementation is on the left, and the implementation employing a 1-level touching algorithm is on the right. (Note the difference in scale on the vertical axes.)

scale between the two figures.)

Although the comparison in Figure 5-7 is quite favorable for the new algorithm, one must bear in mind that given the typical decay in the interpolet expansion coefficients about an atom[5], [3], the functions which are appropriate to maintain in the expansions tend to have the 2-level touching property, not the 1-level touching property. Figure 5-8 compares the runtimes of ∇^2 in three dimensions on a 2-level touching good basis of concentric cubes of size 9^3 , where the speed up just as dramatic as before, now by approximately a factor of 30.

Figure 5-9 compares the runtimes of ∇^2 in three dimensions on a 2-level touching good basis of two refinement centers, with refinements now consisting of cubes of size 9^3 (similar to figure 6). This situation arises in the the calculation of the electronic ground state of the nitrogen molecule, N_2 . Note that with the introduction now of two atomic centers the times are again consistent with the scalings described above: The runs times only double in the multilevel algorithm but quadruple in the naïve algorithm.

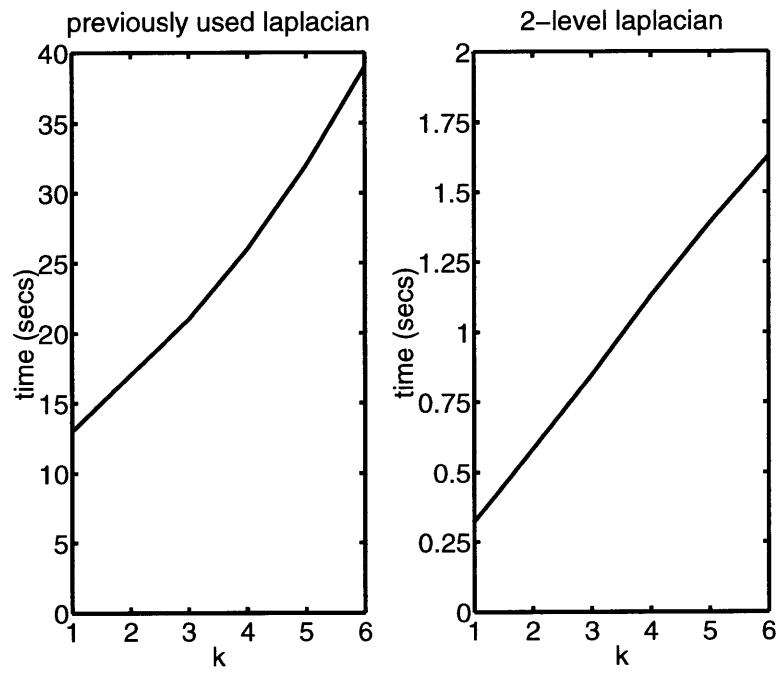


Figure 5-8: The previously used implementation is on the left, and the implementation employing a 2-level touching algorithm is on the right. (Note the difference in scale on the vertical axes.)

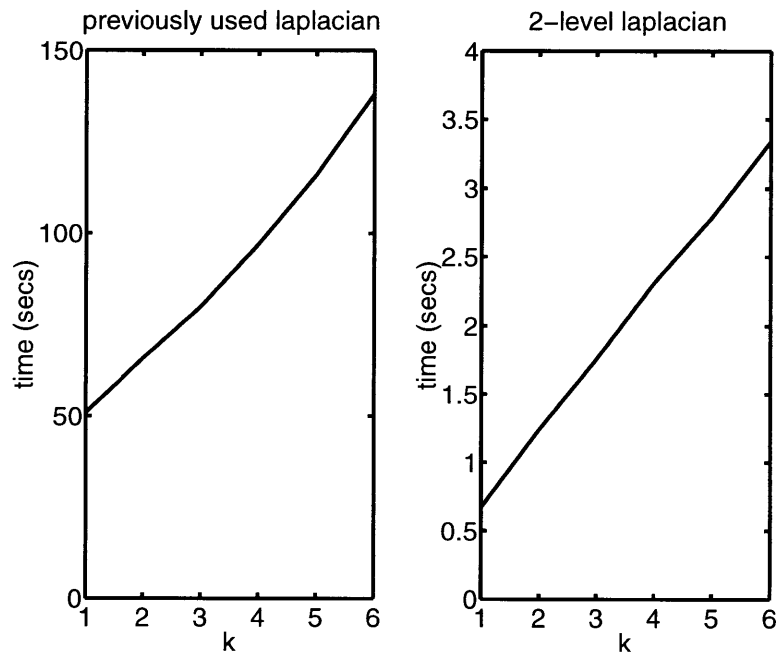


Figure 5-9: The previously used implementation is on the left, and the implementation employing a multilevel algorithm is on the right. (Note the difference in scale on the vertical axes.)

Having considered the efficiency of the algorithms, we next turn to the use of these algorithms in the solution of Poisson's equation to determine electrostatic fields, which was the rate limiting step in the calculations carried out in [5] and [3]. From those calculations, we were aware that the combination of conjugate gradients with the simple preconditioning consisting of just applying the inverse of the diagonal elements of the Laplacian matrix leads to an algorithm requiring very few iterations. It was the application of the operator within each conjugate gradient iteration which limited the efficiency of those earlier calculations.

Figures 5-10,5-11,5-12 illustrate results for varying levels of refinement in the two different systems. The first system consists of refinements centered about a single atomic center within a cubic super cell of side 15 Bohr radii with periodic boundary conditions. (One Bohr radius is approximately 0.529 Angstroms.) The second system contains two refinement centers separated at a distance of 2 Bohr radii, approximately the inter-nuclear separation in the nitrogen molecule. This latter system resides within a rectangular supercell of dimensions $(15 \text{ Bohr})^2 \times (17 \text{ Bohr})$. In both cases, the spacing of the grid at the coarsest scale is 1 Bohr, and the finest spacing is 2^{-k} Bohr. At $k = 22$, the greatest refinement considered in our numerical experiments, the finest grid spacing is approximately $0.24 \times 10^{-6} \text{ \AA}$. A full grid at this resolution would contain 2.8×10^{23} points. Our truncated basis contains only about 60,000 functions in this case.

Figure 5-10 compares, as a function of the number of refinement levels k , the condition number of the Laplacian represented in a truncated interpolet basis (the "stiffness matrix" for the basis) and in an untruncated orthogonal basis at the corresponding resolution. The figure also shows the effect on the condition number of the interpolet stiffness matrix of the simple diagonal preconditioner described above. The condition numbers for the truncated interpolet bases were determined numerically using the operators implemented as described above. The curves indicate results for the system with a single atomic center, and the symbols indicate results for the two atom system. Comparing the results for the one and two atom cases suggests that apart from some transient behavior for small k , the condition number is not sensitive to the number of atoms and depends primarily on the number of refinement levels k .

Although finite basis representations of the Laplacian constructed from orthogonal func-

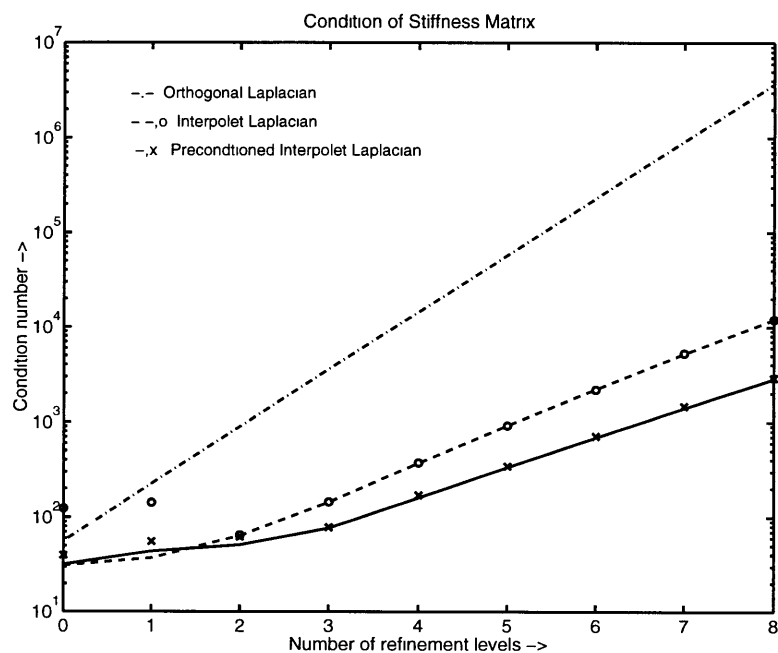


Figure 5-10: The condition number of the Laplacian operator represented in truncated multiresolution interpolet basis as a function of the number of refinement levels k with and without simple diagonal preconditioning and compared with the condition number in an orthogonal basis with the same resolution. Lines indicate results for bases with a single atomic center of refinement, and points represent results for two atomic centers corresponding to the nitrogen molecule.

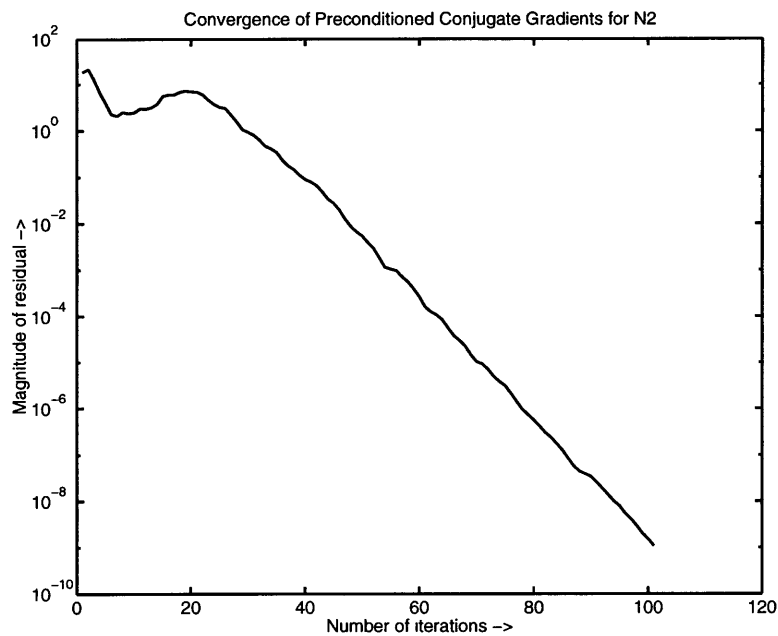


Figure 5-11: Convergence of the solution to Poisson's equation for the nuclear potential in a nitrogen molecule in an interpolet basis with $k = 8$ levels of refinement about each nucleus.

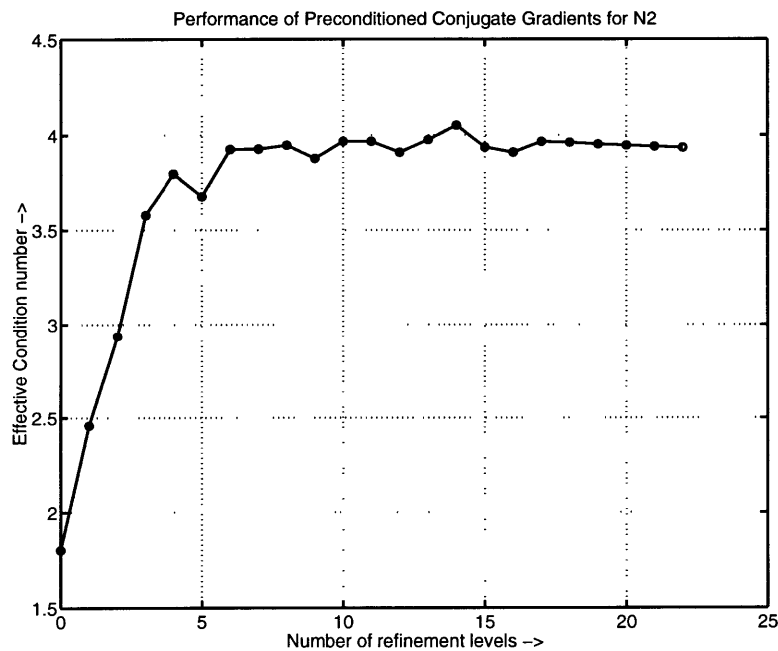


Figure 5-12: Effective condition number of Poisson's equation for the nuclear potential in a nitrogen molecule with simple diagonal preconditioning as a function of k , the number of levels of refinement about each nucleus.

tions at a given resolution should all have similar condition numbers, the fact that the interpolant basis is not orthogonal allows the condition numbers of multiscale interpolant operators to be quite different than their single-scale counterparts. Compared to an orthogonal basis, the condition number in the interpolant representation is already over two orders of magnitude superior at the typical $k = 5$ levels of refinement. This comparison continues to improve with increasing scale. The orthogonal condition number scales inversely as the square of the spacing on the grid of maximum resolution whereas the interpolant condition number scales inversely with approximately the $5/4$ power of the resolution, as determined from the slope in the figure. The interpolant basis itself therefore provides an intrinsic form of preconditioning. Figure 5-10 shows that our simple explicit, diagonal preconditioner improves the scaling of the condition number, which now scales merely as the inverse of the resolution. (Note the lower slope of the lower curve.) At $k = 5$ levels of refinement the improvement is only a factor of three but becomes more significant as the number of refinements increases.

Figure 5-11 shows the convergence of the preconditioned conjugate gradient algorithm in the solution of Poisson's equation. As a simple example, we solve for the electrostatic potential which arises from the two nuclei in a nitrogen molecule. In this calculation we use $k = 8$ levels of refinement, a somewhat higher level of resolution than would be employed in calculations of the N_2 molecule. For this calculation, the charge density of each nucleus is modeled as a three dimensional Gaussian of root mean square width σ along each direction equal to the spacing on the finest scale. After an initial phase of about twenty iterations, the convergence becomes nearly perfectly exponential. This procedure reduces the magnitude of the residual vector by ten orders of magnitude in one hundred iterations. This is very good performance for a system consisting of 14,000 degrees of freedom with a Laplacian operator with a nominal single-scale condition number of about 65,000 at this level of resolution. The slope of this exponential portion of the convergence curve corresponds to a reduction in error at each iteration by 25%. One would obtain the same error reduction in a simple weighted iterative Jacobi algorithm (with the inverse of the maximum eigenvalue as the weight) applied to an operator with condition number $c \approx 4$. The quantity c , the inverse of the fractional improvement in the magnitude of the residual, we define as the *effective condition number* for the algorithm.

Figure 5-12 shows this effective condition number c for the conjugate gradient algorithm with simple diagonal preconditioning as a function of the number of refinement levels k for the solution of Poisson's equation for the nuclei in the nitrogen molecule. In all cases the extent of the nuclei σ is again set to the spacing of the finest grid. We note that after about six refinements, the effective condition number is essentially constant. The example from Figure 5-11 is therefore representative of the typical rate of convergence attained. These results indicate that, regardless of the amount of refinement, a constant number of iterations will suffice to produce a result of a given accuracy, even as the nominal condition number for an orthogonal stiffness at the corresponding resolution approaches 1.8×10^{13} at $k = 22$. Because the computational work involved in each iteration is linear in the number of points in the basis, this approach appears to produce the solution to Poisson's equation in $O(N)$ time for these multiresolution bases.

Appendix A

Matlab source for Stiefel-Grassmann optimization

This appendix presents the source code for the `sg_min` routines that were discussed in chapter 4. We have included the sources for the basic routines followed by the sources for the various examples.

Sources from the top-level `sg_min` directory:

```
function [fn,Yn] = sg_min(Y0,varargin)
% SG_MIN      Stiefel/Grassmann minimization meant for template use.
%
%      [fn,Yn] = SG_MIN(Y0) minimizes F(Y) where F is
%      a matlab function defined in F.m (dF.m and ddF.m),
%      where Y satisfies Y'*Y=I.
%
%      [fn,Yn] = SG_MIN(Y0,rc,mode,metric,verbose,gradtol,ftol,partition)
%
%      Required Argument:  Y0 expected orthonormal, Y0'*Y0 = I
%      Optional Arguments: (may be specified in nearly any order)
```

10

```

%      rc={ 'real', 'complex' } specifies real vs complex computation.
%      mode={ 'frcg', 'newton' } picks search direction:
%          Fletcher–Reeves Nonlinear Conjugate Gradient
%          Newton's method (Linear CG Hessian inverter)
%      metric={ 'flat', 'euclidean', 'canonical' }
%      motion={ 'approximate', 'exact' }
%      verbose={ 'verbose', 'quiet' }
%      gradtol={ first of any scalar arguments } convergence tolerance
%      ftol  = { second of any scalar arguments } convergence tolerance
%      partition = cell array describing symmetries in F
%      Defaults:
%      rc: 'real' if isreal(Y0), 'complex' otherwise
%      partition: automatically determined
%      SG_MIN(Y0,rc,'newton','euclidean','approximate','verbose',1e-6, 1e-8, partition)
%      Output:
%      fn = function minimum
%      Yn = minimizing argument will satisfy  $Y_n^* Y_n = I$ .
%      role parses the arguments, sets the global parameters and calls the
%      minimizers

```

```

if ~ exist('F', 'file'), error('F.m must be in matlab''s path'), end
if ~ exist('dF', 'file'), error('dF.m must be in matlab''s path'), end
if ~ exist('ddF', 'file'), error('ddF.m must be in matlab''s path'), end

```

```

global SGParameters;
SGParameters = [];

[Y0,r] = qr(Y0,0);
SGParameters.verbose = 1;

```

```

nas = length(varargin);
metarg = 0; rcarg = 0; partarg = 0; ftolarg = 0; gradtolarg = 0;
mdarg = 0; motarg = 0;
for j=1:nas,
    if (ischar(varargin{j}))

```



```

if (strcmp(lower(varargin{j}), 'approximate'))
    SGParameters.motion = 0; motarg=1;
elseif (strcmp(lower(varargin{j}), 'exact'))
    SGParameters.motion = 1; motarg=1; 50
elseif (strcmp(lower(varargin{j}), 'flat'))
    SGParameters.metric = 0; metarg=1;
elseif (strcmp(lower(varargin{j}), 'euclidean'))
    SGParameters.metric = 1; metarg=1;
elseif (strcmp(lower(varargin{j}), 'canonical'))
    SGParameters.metric = 2; metarg=1;
elseif (strcmp(lower(varargin{j}), 'real'))
    SGParameters.complex = 0; rcarg=1;
elseif (strcmp(lower(varargin{j}), 'complex'))
    SGParameters.complex = 1; rcarg=1; 60
elseif (strcmp(lower(varargin{j}), 'quiet'))
    SGParameters.verbose = 0; verbarg=1;
elseif (strcmp(lower(varargin{j}), 'verbose'))
    SGParameters.verbose = 1; verbarg=1;
elseif (strcmp(lower(varargin{j}), 'frcg'))
    SGParameters.Mode = 1; mdarg=1;
elseif (strcmp(lower(varargin{j}), 'newton'))
    SGParameters.Mode = 0; mdarg=1;
end
elseif (iscell(varargin{j})) 70
    part = varargin{j}; partarg=1;
elseif (isnumeric(varargin{j}))
    if (gradtolarg)
        SGParameters.ftol = varargin{j}(1);
        ftolarg=1;
    else
        SGParameters.gradtol = varargin{j}(1);
        gradtolarg=1;
    end
end 80
end

```

```

% SGParameters.complex = 0 for real and 1 for complex.
    if (rcarg)
        if (~isreal(Y0)) & (1-SGParameters.complex)
            warning('Y0 has imaginary part, but real computation has been declared.
        end
    else
        SGParameters.complex = ~isreal(Y0);
    end
% SGParameters.metric = 0 for flat, 1 for euclidean, and 2 for canonical
    if (~metarg)
        SGParameters.metric = 1;
    end
% SGParameters.motion = 0 for approximate, 1 for exact
    if (~motarg)
        SGParameters.motion = 0;
    end
    if (~gradtolarg)
        SGParameters.gradtol = 1e-6;
    end
    if (~ftolarg)
        SGParameters.ftol = 1e-8;
    end
    if (~mdarg)
        SGParameters.Mode = 0;
    end
% Make a partition using a possible given one given one
    if (partarg)
        SGParameters.partition = part;
    else
        SGParameters.partition = partition(Y0);
    end

SGParameters.dimension = dimension(Y0);

if (SGParameters.Mode == 1)

```

90

100

110

```

        [fn, Yn] = sg_frcg(Y0);
    else
        [fn, Yn] = sg_newton(Y0);
    end

```

120

function [fn, Yn]= sg_newton(Y)

```

% SG_NEWTON(Y) Optimize the objective function, F(Y) over all
%     Y such that Y'*Y = I. Employs a local iterative search with
%     initial point Y and terminates if the magnitude of the gradient
%     falls to gradtol*(initial gradient magnitude) or if the relative
%     decrease in F after some iteration is less than ftol.
%
%     [fn, Yn]= SG_NEWTON(Y)
%     Y is expected to satisfy Y'*Y = I.
%     Yn will satisfy Yn'*Yn = I.

```

10

```

% role  high level algorithm, Newton's Methods

```

```

global SGParameters;
gradtol = SGParameters.gradtol;
ftol = SGParameters.ftol;

```

```

if (SGParameters.verbose)

```

```

    global SGdata;
    flps = flops;
    SGdata=[];

```

```

end

```

20

```

g = grad(Y); mag = sqrt(ip(Y,g,g)); oldmag = mag;

```

```

f = F(Y); oldf = 2*f; N = 0;

```

```

if (SGParameters.verbose)

```

```

    SGdata = [];
    disp(sprintf('%s\t%s\t\t%s\t\t%s\t\t%s', 'iter', 'grad', 'F(Y)', ' flops', 'step type'));
    SGdata(N+1,:) = [N mag f flops-flps];
    disp(sprintf('%d\t%e\t%e\t%9d\t%s', N, mag, f, flops-flps, 'none'));

```

```

end

```

```

while (mag>eps) & (mag/oldmag>gradtol) & (abs(oldf/f-1)>ftol)

```

```

    N= N+1;

```

30

```

sdir = -g;
gsdir = ip(Y,sdir,-g); sdir=sdir*sign(gsdir); gsdir=abs(gsdir);
sa = fmin('Fline',-abs(f/gsdir),abs(f/gsdir),[0 1e-4],Y,sdir);
fsa = F(move(Y,sdir,sa));
Hsdir = dgrad(Y,sdir); sdirHsdir = ip(Y,sdir,Hsdir);
sb = fmin('Fline',-abs(gsdir/sdirHsdir),abs(gsdir/sdirHsdir),[0 1e-4],Y,sdir);
fsb = F(move(Y,sdir,sb));

ndir = invdgrad(Y,-g,gradtol*oldmag);
40

gndir = ip(Y,ndir,-g); ndir = ndir*sign(gndir);
na = fmin('Fline',-abs(f/gndir),abs(f/gndir),[0 1e-4],Y,ndir);
fna = F(move(Y,ndir,na));
Hndir = dgrad(Y,ndir); ndirHndir = ip(Y,ndir,Hndir);
nb = fmin('Fline',-abs(gndir/ndirHndir),abs(gndir/ndirHndir),[0 1e-4],Y,ndir);
fnb = F(move(Y,ndir,nb));

if (fsa<fsb) st=sa; fst=fsa; else st=sb; fst=fsb; end
if (fna<fnb) nt=na; fnt=fna; else nt=nb; fnt=fnb; end
50
if (fst<fnt)
    if (SGParameters.verbose) steptype='steepest step'; end
    dir = sdir; t = st; newf = fst;
else
    if (SGParameters.verbose) steptype='newton step'; end
    dir = ndir; t = nt; newf = fnt;
end
if (newf>f)
    if (SGParameters.verbose) disp(' sg_newton: fmin overshoot!'), end
    fprime = ip(Y,g,dir);
60
    while (newf>f)
        t = t^2*fprime/(t*fprime+f-newf)/2;
        newf = F(move(Y,dir,t));
    end
end

```

```

Y = move(Y,dir,t); oldf= f; f = newf;
g=grad(Y); mag=sqrt(ip(Y,g,g));
if (SGParameters.verbose)
    SGdata(N+1,:) = [N mag f flops-flops];
    disp(sprintf('%d\t%e\t%e\t%9d\t%s',N,mag,f,flops-flops,steptype));
end
end
fn = f;
Yn = Y;

```

```

function [fn,Yn]= sg_frcg(Y)
% SG_FRCG(Y) Optimize the objective function, F(Y) over all
% Y such that Y'*Y = I. Fletcher-Reeves CG iterative search with
% initial point Y and terminates if the magnitude of the gradient
% falls to gradtol*(initial gradient magnitude) or if the relative
% decrease in F after some iteration is less than ftol.
%
% [fn,Yn]= SG_CG(Y)
% Y is expected to satisfy Y'*Y = I.
% Yn will satisfy Yn'*Yn = I.
% role high level algorithm, Fletcher-Reeves Method
global SGParameters;
gradtol = SGParameters.gradtol;
ftol = SGParameters.ftol;

if (SGParameters.verbose)
    global SGdata;
    flps = flops;
    SGdata=[];
end
g = grad(Y); mag = sqrt(ip(Y,g,g)); oldmag = mag;
f = F(Y); oldf = 2*f; N = 0;
olddir = 0;
if (SGParameters.verbose)
    SGdata = [];

```

```

disp(sprintf('%s\t%s\t\t%s\t\t%s', 'iter', 'grad', 'F(Y)', ' flops'));
SGdata(N+1,:) = [N mag f flops-flps];
disp(sprintf('%d\t%e\t%e\t%9d', N, mag, f, flops-flps));
end
while (mag>eps) & (mag/oldmag>gradtol) & (abs(oldf/f-1)>ftol) 30
    N= N+1;

    dir = -g;
    if (N>1)
        Hessolddir = dgrad(Y,olddir);
        alpha = ip(Y,dir,Hessolddir)/ip(Y,olddir,Hessolddir);
        dir = dir-alpha*olddir;
    end
    gdir = ip(Y,dir,-g); dir = dir*sign(gdir);
    cga = fmin('Fline',-abs(f/gdir),abs(f/gdir),[0 1e-4],Y,dir); 40
    fcga = F(move(Y,dir,cga));
    Hessdir = dgrad(Y,dir); dirHdir = ip(Y,dir,Hessdir);
    cgb = fmin('Fline',-abs(gdir/dirHdir),abs(gdir/dirHdir),[0 1e-4],Y,dir);
    fcgb = F(move(Y,dir,cgb));
    if (fcga<fcgb) t=cga; newf=fcga; else t=cgb; newf=fcgb; end
    if (newf>f)
        if (SGParameters.verbose) disp(' sg_frcg: fmin overshoot!'), end
        fprime = ip(Y,g,dir);
        while (newf>f)
            t = t^2*fprime/(t*fprime+f-newf)/2; 50
            newf = F(move(Y,dir,t));
        end
    end
    [Y,olddir] = move(Y,dir,t); oldf= f; f = newf;
    g=grad(Y); mag=sqrt(ip(Y,g,g));
    if (SGParameters.verbose)
        SGdata(N+1,:) = [N mag f flops-flps];
        disp(sprintf('%d\t%e\t%e\t%9d', N, mag, f, flops-flps));
    end
end 60

```

```
fn = f;
Yn = Y;
```

```
function G = grad(Y)
```

```
% GRAD computes the gradient of the energy F at the point Y.
```

```
%
```

```
% G = GRAD(Y)
```

```
% Y is expected to satisfy  $Y'Y=I$ .
```

```
% G will satisfy  $G = \text{tangent}(Y,G)$ 
```

```
%
```

```
% role geometrized objective function, this is the routine called to produce
```

```
% the geometric gradient of the unconstrained differential of F.
```

```
% The analog is  $G = A*Y-b$ .
```

10

```
G = tangent(Y,dF(Y));
```

```
function W = dgrad(Y,H)
```

```
% DGRAD Computes the tangent vector, W, which results from applying the
```

```
% geometrically correct hessian at the stiefel point, Y, to the
```

```
% tangent vector H.
```

```
%
```

```
% W = DGRAD(Y,H)
```

```
% Y is expected to satisfy  $Y'Y = I$ 
```

```
% H is expected to satisfy  $H = \text{tangent}(Y,H)$ 
```

```
% W will satisfy  $W = \text{tangent}(Y,W)$ 
```

```
%
```

10

```
% role geometrized objective function, the is the routine called to apply
```

```
% the covariant hessian of F to a tangent vector. The analog is
```

```
%  $W = A*H$ , here A is the hessian of F.
```

```
global SGParameters;
```

```
met = SGParameters.metric;
```

```
if (met==0)
```

```
    W=tangent(Y,ddF(Y,H));
```

```
else
```

```
    df = dF(Y); g = tangent(Y,df);
```

```

W = connection(Y,g,H)+...
    dtangent(Y,df,H)+...
    tangent(Y,ddF(Y,H));

```

20

```
end
```

```
% this line removes the grassmann degrees of freedom from Y
```

```
W = nosym(Y,W);
```

```
function H = invdgrad_cg(Y,W,tol)
```

```
% INVDGRAD_CG Inverts the operator dgrad. i.e. solves for H satisfying
```

```
%          dgrad(Y,H) = W. Uses a conjugate gradient algorithm with
```

```
%          a tolerance of gep*norm(W), or a tolerance of tol if given.
```

```
%
```

```
%          H = INVDGRAD_CG(Y,W)
```

```
%          Y is expected to satisfy Y'*Y=I
```

```
%          W is expected to satisfy W = tangent(Y,W)
```

```
%          H will satisfy H = tangent(Y,H)
```

```
%
```

10

```
% role  geometrized objective function, this is the function called when one
```

```
% wishes to invert the geometric hessian. The analog is H = Hess\W.
```

```
global SGParameters;
```

```
dim = SGParameters.dimension;
```

```
gep = SGParameters.gradtol;
```

```
b = W;
```

```
% p = reshape(dgrad(Y,reshape(q,n,k)),n*k,1) is the black box which
```

```
% applies the hessian to a column vector, q, to produce a column vector p.
```

20

```
% Any iterative inversion algorithm that applies the matrix as a black
```

```
% box can be used here.
```

```
oldd = 0;
```

```
x = 0*b;
```

```
r = -b;
```

```
rho = ip(Y,r,r);
```

```
if (nargin==3)
```



```

        gepr = tol;
    else
        gepr = sqrt(ip(Y,r,r))*gep;
    end
    posdef=1;
    cn = 0;
    while (posdef & sqrt(ip(Y,r,r))>gepr & cn<2*dim)
% application of the hessian.
        cn = cn+1;
        if (cn==1)
            d = -r;
        else
            beta = rho/oldrho;
            d = -r+beta*oldd;
        end
% It is important to make sure long term roundoff does not blow us
% off the tangent surface. So we reproject.
        d = tangent(Y,d);
        Ad = dgrad(Y,d);
        dAd = ip(Y,d,Ad);
        if (dAd<0)
            posdef=0;
        else
            dist = rho/dAd;
            x = x+dist*d;
            r = r+dist*Ad;
            oldd = d;
            oldrho = rho;
            rho = ip(Y,r,r);
        end
    end
    if (SGParameters.verbose & posdef==0)
        disp(' invdgrad: Hessian not positive definite, CG terminating early');
        if (cn==1) x = -b; end
    end
end

```

```

if (SGParameters.verbose & cn==2*dim)
    disp(' invdgrad: max iterations reached inverting the hessian by CG'),
    if (cn==1) x = -b; end
end
H = tangent(Y,x);

```

```

function H = tangent(Y,D)

```

```

% TANGENT produces tangent H from unconstrained differential D.

```

```

% H will be the unique tangent vector such that for any

```

```

% tangent W,  $\text{real}(\text{trace}(D'W)) = \text{ip}(Y, W, H)$ .

```

```

%

```

```

% H = TANGENT(Y,D)

```

```

% Y is expected to satisfy  $Y'Y=I$ 

```

```

% D is expected to be the same size as Y

```

```

% H will satisfy  $H = \text{tangent}(Y,H)$ 

```

```

%

```

10

```

% role geometric implementation, this function helps produce the gradient and

```

```

% covariant hessian.

```

```

global SGParameters;

```

```

met = SGParameters.metric;

```

```

vert = Y'*D;

```

```

verts = (vert+vert')/2;

```

```

if (met==0)

```

```

    H = D - Y*verts;

```

```

elseif (met==1)

```

```

    H = D - Y*verts;

```

20

```

elseif (met==2)

```

```

    H = (D - Y*vert')/2;

```

```

end

```

```

function T = dtangent(Y,H,dY)

```

```

% DTANGENT computes the differential of the tangent map. That is

```

```

%  $T = d/dt \text{tangent}(\text{params}, Y+t*dY, H)$ .

```

```

%

```

```

%      T = DTANGENT(Y,H,dY)
%      Y is expected to satisfy Y'*Y = I
%      H is expected to satisfy H = tangent(Y,H)
%      dY is expected to be the same size as Y
%      T will be the same size as Y
%
%
% role  geometric implementation, helps to produce a geometrically
%       correct covariant hessian.
global SGParameters;
met = SGParameters.metric;
vert = Y'*H; verts = (vert+vert')/2;
dvert = dY'*H; dverts = (dvert+dvert')/2;
if (met==0)
    T = -dY*verts-Y*dverts;
elseif (met==1)
    T = -dY*verts-Y*dverts;
elseif (met==2)
    T = (-dY*vert'-Y*dvert')/2;
end

```

```

function [Yo,Ho] = move(Yi,Hi,t)
% MOVE moves the point Yi and direction Hi along a geodesic of length t
% in the metric to a point Yo and a direction Ho.
%
% [Yo,Ho] = MOVE(Yi,Hi,t)
% Yi is expected to satisfy Yi'*Yi = I
% Hi is expected to satisfy Hi = tangent(Yi,Hi)
% Yo will satisfy Yo'*Yo = I
% Ho will satisfy Ho = tangent(Yo,Ho)
%
%
% role  geometric implementation, the analog to Yo = Yi+Hi*t, Ho = Hi.
global SGParameters;
met = SGParameters.metric;
mot = SGParameters.motion;
[n,k] = size(Yi);

```

```

    if (t==0)
        Yo = Yi;
        if (nargout==2)
            Ho = Hi;
        end
        return;
    end
    if (met==0)
        % Move by straight lines with a qr projection back to the manifold
        if (nargout==2) mag1 = sqrt(ip(Yi,Hi,Hi)); end
        [Yo,r] = qr(Yi+t*Hi,0);
        if (nargout==2)
            Ho = tangent(Yo,Hi);
            mag2 = sqrt(ip(Yo,Ho,Ho));
            Ho = Ho*mag1/mag2;
        end
    elseif (met==1)
        if (mot==0)
            % This section computes approximate euclidean geodesics
            % using polar decomposition
            if (nargout==2) mag1 = sqrt(ip(Yi,Hi,Hi)); end
            [U,S,V] = svd(Yi+t*Hi,0); Yo = U*V';
            if (nargout==2)
                Ho = tangent(Yo,Hi);
                mag2 = sqrt(ip(Yo,Ho,Ho));
                Ho = Ho*mag1/mag2;
            end
        else
            % This section computes exact euclidean geodesics
            if (nargout==2) mag1 = sqrt(ip(Yi,Hi,Hi)); end
            a = Yi'*Hi;
            [q,r] = qr(Hi-Yi*a,0);
            mn = expm(t*[2*a, -r'; r, zeros(k)]);
            nm = expm(-t*a);
            Yo = (Yi*mn(1:k,1:k) + q*mn(k+1:2*k,1:k))*nm;
        end
    end

```

```

[Yo,r] = qr(Yo,0);
if (nargout==2)
    q =(Yi*mn(1:k,k+1:2*k)+q*mn(k+1:2*k,k+1:2*k));
    Ho = Yi*a+q*r*nm;
    Ho = tangent(Yo,Hi);
    mag2 = sqrt(ip(Yo,Ho,Ho));
    Ho = Ho*mag1/mag2;
end
end
elseif (met==2)
    if (mot==0)
        % this section computes approximate canonical geodesics using approximate
        % matrix inverse.
        if (nargout==2) mag1 = sqrt(ip(Yi,Hi,Hi)); end
        a = Yi'*Hi;
        [q,r] = qr(Hi-Yi*a,0);
        geo = t*[a, -r'; r, zeros(k)];
        mn = (eye(2*k)+geo/2)/(eye(2*k)-geo/2);
        mn = mn(:,1:k);
        Yo = Yi*mn(1:k,:) + q*mn(k+1:2*k,:);
        if (nargout==2)
            Ho = Hi*mn(1:k,:) - Yi*(r'*mn(k+1:2*k,:));
            Ho = tangent(Yo,Hi);
            mag2 = sqrt(ip(Yo,Ho,Ho));
            Ho = Ho*mag1/mag2;
        end
    else
        % This section computes exact canonical geodesics
        if (nargout==2) mag1 = sqrt(ip(Yi,Hi,Hi)); end
        a = Yi'*Hi;
        [q,r] = qr(Hi-Yi*a,0);
        geo = t*[a, -r'; r, zeros(k)];
        mn = expm(geo);
        mn = mn(:,1:k);
        Yo = Yi*mn(1:k,:) + q*mn(k+1:2*k,:);

```

```

        if (nargout==2)
            Ho = Hi*mn(1:k,:) - Yi*(r'*mn(k+1:2*k,:));
            Ho = tangent(Yo,Hi);
            mag2 = sqrt(ip(Yo,Ho,Ho));
            Ho = Ho*mag1/mag2;
        end
    end
end

```

```

function f = Fline(t,Y,H)

```

```

% FLINE the energy,  $e$ , along the geodesic passing through  $Y$ 

```

```

%     in the direction  $H$  a distance  $t$ .

```

```

%

```

```

%      $f = \text{FLINE}(t, Y, H)$ 

```

```

%      $Y$  is expected to satisfy  $Y' * Y = I$ .

```

```

%      $H$  is expected to satisfy  $H = \text{tangent}(Y, H)$ 

```

```

%

```

```

% role  high level algorithm, basic window dressing for the fmin function

```

```

    f = F(move(Y,H,t));

```

10

```

function C = connection(Y,H1,H2)

```

```

% CONNECTION Produces the christoffel symbol,  $C$ , for either the canonical

```

```

%     or euclidean connections at the point  $Y$ .

```

```

%

```

```

%      $C = \text{CONNECTION}(Y, H1, H2)$ 

```

```

%      $Y$  is expected to satisfy  $Y' * Y = I$ 

```

```

%      $H1$  and  $H2$  satisfy  $H1 = \text{tangent}(Y, H1)$  and  $H2 = \dots$  (similarly)

```

```

%      $C$  will be a matrix of the same size as  $Y$ 

```

```

%

```

```

% role  geometric implementation, an important term in computing the hessian

```

10

```

%     and in defining the geodesics.

```

```

    global SGParameters;

```

```

    met = SGParameters.metric;

```

```

    if (met==0)

```

% the unconstrained connection

C = zeros(size(Y));

elseif (met==1)

% the euclidean connection for stiefel

C = Y*(H1'*H2+H2'*H1)/2;

elseif (met==2)

20

% the canonical connection for the stiefel

b = H1'*H2-H1'*Y*Y'*H2;

C = (H1'*H2'*Y+H2'*H1'*Y)/2+Y*(b+b')/2;

end

function i = ip(Y,H1,H2)

% IP computes the inner produce of H1,H2 which are tangents at the stiefel point Y.

%

% i = IP(Y,H1,H2)

*% Y is expected to satisfy Y'*Y=I*

% H1,y are expected to satisfy H1 = tangent(Y,H1), H2 = tangent(Y,H2)

%

*% role geometric implementation, the analog of real(H1'*H2)*

global SGParameters;

10

met = SGParameters.metric;

if (met==0)

% unconstrained metric

i = sum(sum(real(conj(H1).*H2)));

elseif (met==1)

% euclidean metric

i = sum(sum(real(conj(H1).*H2)));

elseif (met==2)

% canonical metric

i = sum(sum(real(conj(H1).*H2)))-sum(sum(real(conj(Y'*H1).*(Y'*H2)))/2;

20

end

function H = nosym(Y,H)

```

% NOSYM Orthogonal projection to remove the block diagonal components
%   of the tangent vector H corresponding to the block diagonal
%   right symmetries of F(Y).
%
%   H = NOSYM(Y,H)
%   Y is expected to satisfy Y'*Y=I
%   H is expected to and sill satisfy H = tangent(Y,H)
%
% role   geometrized objective function, necessary to reduce the number of
%         dimensions of the problem and to have a well conditioned hessian.
%         Somewhat analogous to projecting H to a feasible set of search
%         directions.
%
global SGParameters;
part = SGParameters.partition;
vert = Y'*H;
for j = 1:length(part),
    H(:,part{j})=H(:,part{j})-Y(:,part{j})*vert(part{j},part{j});
end

```

```

function part = partition(Y0)
% PARTITION makes a guess at the partition of the function F
%   based on the properties of dF at a randomly selected point Y.
%
%   part = partition(Y0);
%   Y0 is expected to satisfy Y0'*Y0 = I
%   P = size(Y0,2)
%
% role   geometrized objective function, determines the feasible subspace
%         of possible search directions for the function F.
%
global SGParameters;
cm = SGParameters.complex;
[N,P] = size(Y0);
[Y,r] = qr(randn(N,P)+i*cm*randn(N,P),0);
A = Y'*dF(Y);
As = (A+A')/2;

```



```

Aa = (A-A')/2;
M = (abs(Aa)<1e-7*abs(As));

scorebd = ones(1,P); np=0;
for j=1:P,
    if (scorebd(j))
        np=np+1;
        part{np} = find(M(j,:));
        scorebd(part{np}) = 0*part{np};
    end
end

Mcomp = 0*M;
for j=1:length(part),
    Mcomp(part{j},part{j}) = ones(length(part{j}));
end

goodpart = (sum(sum(abs(Mcomp-M)))==0);
if ~goodpart
    warning('unable to find consistent partition of F. ');
    part = num2cell(1:P);
end

```

```

function dim = dimension(Y0)
% DIMENSION correctly counts the dimension, dim (stiefel dimension minus
% the grassmann degrees of freedom).
%
% dim = DIMENSION
%
% role geometrized objective function, used to set the parameters describing
% the particular submanifold of the stiefel manifold on which the
% computation will occur.
global SGParameters;
part = SGParameters.partition;
cm = SGParameters.complex;

```

```

[N,P] = size(Y0);

np = length(part);
if (~cm)
    dim = N*P;
    dim = dim - P*(P+1)/2;
    for i=1:np,
        k = length(part{i});
        dim = dim - k*(k-1)/2;
    end
else
    dim = 2*N*P;
    dim = dim - P^2;
    for i=1:np,
        k = length(part{i});
        dim = dim - k^2;
    end
end

```

20

30

Sources from the `sg_min/finitediff` subdirectory:

```
function df = dF(Y)
% dF   Computes the differential of F, that is,
%      df satisfies  $\text{real}(\text{trace}(H'*df)) = d/dx (F(Y+x*H))$ .
%
%      df = DF(Y)
%      Y is expected to satisfy  $Y'*Y = I$ 
%      df is the same size as Y
%
% role  objective function, this is the routine called to compute the
%       differential of F. 10
[N,P] = size(Y);
ep = 1e-6;
for k=1:P, for j=1:N,
    Yp = Y; Yp(j,k) = Yp(j,k)+ep;
    Ym = Y; Ym(j,k) = Ym(j,k)-ep;
    df(j,k) = (F(Yp)-F(Ym))/ep/2;
end, end
```

```
function ddf = ddF(Y,H)
% ddF  Computes the second derivative of F, that is,
%      ddf =  $d/dx dF(Y+x*H)$ .
%
%      ddf = DDF(Y,H)
%      Y is expected to satisfy  $Y'*Y = I$ 
%      H is expected to be the same size as Y
%      ddf will be the same size as Y
%
% role  objective function, the function is called to apply the
%       unconstrained hessian of F to a vector. 10
ep = 1e-6;
ddf = (dF(Y+ep*H)-dF(Y-ep*H))/ep/2;
```

Sources from the `sg_min/examples/procrustes` subdirectory:

function `f = F(Y)`

```
% F    Computes the energy associated with the stiefel point Y.
%      In this case, by the equation  $f = \|AY - YB\|^2/2$ .
%
%       $f = F(Y)$ 
%      Y is expected to satisfy  $Y' * Y = I$ 
%
% role objective function, the routine called to compute the objective
%      function.
        global FParameters;
        A = FParameters.A;
        B = FParameters.B;
        q = A*Y - Y*B;
        f = sum(sum(real(conj(q).*(q))))/2;
```

10

function `df = dF(Y)`

```
% dF    Computes the differential of F, that is,
%       $df$  satisfies  $\text{real}(\text{trace}(H' * df)) = d/dx (F(Y + x * H))$ .
%
%       $df = DF(Y)$ 
%      Y is expected to satisfy  $Y' * Y = I$ 
%       $df$  is the same size as Y
%
% role objective function, this is the routine called to compute the
%      differential of F.
        global FParameters;
        A = FParameters.A;
        B = FParameters.B;
        q = A*Y - Y*B;
        df = A'*q - q*B';
```

10

function `ddf = ddF(Y,H)`

```

% ddF Computes the second derivative of F, that is,
% ddf = d/dx dF(Y+x*H).
%
% ddf = DDF(Y,H)
% Y is expected to satisfy Y'*Y = I
% H is expected to be the same size as Y
% ddf will be the same size as Y
%
% role objective function, the function is called to apply the 10
% unconstrained hessian of f to a vector.
global FParameters;
A = FParameters.A;
B = FParameters.B;
dq = A*H-H*B;
ddf = A'*dq-dq*B';

```

```

function Y = guess()
% GUESS provides the initial starting guess Y for energy minimization.
% In this case it makes a somewhat random guess.
%
% Y = GUESS
% Y will satisfy Y'*Y = I
%
% role objective function, produces an initial guess at a minimizer 10
% of F.
global FParameters;
inA = FParameters.A;
inB = FParameters.B;
[n,n] = size(inA);
[p,p] = size(inB);
Y = randn(n,p);
Y = (inA*Y)/inB;
[Y,r] = qr(Y,0);

```

```
function parameters(A,B)
% PARAMETERS initializes the parameters for an instance of a
%      minimization problem.
%
%      PARAMETERS(A,B)
%      A is expected to be a square matrix
%      B is expected to be a square matrix
%
% role sets up the global parameters used at all levels of computation.
```

10

```
global FParameters;
FParameters = [];
FParameters.A = A;
FParameters.B = B;
```

```
function [A,B] = randprob()
    A = randn(10);
    B = randn(4);
```

Sources from the `sg_min/examples/jordan` subdirectory:

```
function f = F(Y)
% F    Computes the objective value associated with the stiefel point Y.
%      In this case, by the equation  $f = \|AY - YB(Y)\|^2/2$ .
%
%       $f = F(Y)$ 
%      Y is expected to satisfy  $Y' * Y = I$ 
%
% role objective function, the routine called to compute the objective
%       function.
global FParameters;
A = FParameters.A;
Ab = Block(Y);
f = A * Y - Y * Ab; f = sum(sum(real(conj(f) .* f)))/2;
```

```
function df = dF(Y)
% dF    Computes the differential of F, that is,
%        $df$  satisfies  $\text{real}(\text{trace}(H' * df)) = d/dx (F(Y + x * H))$ .
%
%        $df = DF(Y)$ 
%       Y is expected to satisfy  $Y' * Y = I$ 
%        $df$  is the same size as Y
%
% role objective function, this is the routine called to compute the
%       differential of F.
global FParameters;
A = FParameters.A;
Ab = Block(Y);
f = A * Y - Y * Ab;
df = A' * f - f * Ab';
```

```
function ddf = ddF(Y,H)
% ddF    Computes the second derivative of F, that is,
```

```

%      ddf = d/dx dF(Y+x*H).
%
%      ddf = DDF(Y,H)
%      Y is expected to satisfy Y'*Y = I
%      H is expected to be the same size as Y
%      ddf will be the same size as Y
%
% role  objective function, the function is called to apply the          10
%       unconstrained hessian of F to a vector.
global FParameters;
A = FParameters.A;
Ab = Block(Y);

dAb = dBlock(Y,H);
f = A*Y - Y*Ab;
df = A*H - H*Ab - Y*dAb;
ddf = A'*df - df*Ab' - f*dAb';

```

```

function Y = guess()
% GUESS provides the initial starting guess Y for energy minimization.
%
%      Y = GUESS
%      Y will satisfy Y'*Y = I
%
% role  objective function, produces an initial guess at a minimizer
%       of F.
global FParameters;
inA = FParameters.A;
blocks = FParameters.blocks;
eigs = FParameters.eigs;

[n,k] = size(inA);
[ne,ns] = size(blocks);
off = 0;
Q = eye(n); A = inA;

```



```

for i=1:ne, for j=1:ns,
    if (blocks(i,j) ~= 0)
        [u,s,v] = svd(A(off+1:n,off+1:n)-eye(n-off)*eigs(i));
        v = v(:,n-off:-1:1);
        q = [eye(off) zeros(off,n-off); zeros(n-off,off) v];
        A = q'*A*q;
        Q = Q*q;
        s = diag(s); s = flipud(s);
        off = off + blocks(i,j);
    end
end, end
Y = Q(:,1:off);

```

```

function [B,eigvs] = Block(Y)
% BLOCK computes the matrix B appearing in the energy function
% F = ||AY-YB(Y)||^2 for jordan block problems.
%
% [B,eigvs] = Block(Y)
% Y is expected to be a stiefel point (Y'*Y= I)
% B will be an upper triangular matrix in staircase form the
% size of Y'*Y
% eigvs will be the eigenvalues along the diagonal of B. If
% FParameters.type == 'orbit' then eigvs == FParameters.eigs.
%
% role objective function, auxiliary routine used by F, dF, ddF
global FParameters;
A = FParameters.A;
eigvs = FParameters.eigs;
blocks = FParameters.blocks;
bundle = FParameters.bundle;

[n,k] = size(Y);
B = Y'*A*Y;
[ne,ns] = size(blocks);
off = 0;

```

```

for i=1:ne,
    if (bundle)
        eigvs(i)= trace(B(off+1:off+sum(blocks(i,:)),off+1:off+sum(blocks(i,:))))/sum(blocks(i,:))
    end
    for j=1:ns,
        if (blocks(i,j)>0)
            B(off+1:k,off+1:off+blocks(i,j))= eigvs(i)*eye(k-off,blocks(i,j));
            off = off + blocks(i,j);
        end
    end
end

```

30

```

function B = dBlock(Y,H)

```

```

% dBLOCK    This is an auxiliary procedure used by ddF which computes

```

```

%    B = d/dx Block(Y+x*H)

```

```

%

```

```

%    B = DBLOCK(Y,H)

```

```

%    Y is expected to be a stiefel point (Y'*Y= I)

```

```

%    H is expected to satisfy H = tangent(Y,H)

```

```

%    B will be a staircase matrix the size of Y'*Y

```

```

%

```

```

% role    objective function, auxiliary function for ddF

```

10

```

global FParameters;

```

```

A = FParameters.A;

```

```

eigvs = FParameters.eigs;

```

```

blocks = FParameters.blocks;

```

```

bundle = FParameters.bundle;

```

```

[n,k] = size(Y);

```

```

B =H'*A*Y+Y'*A*H;

```

```

[ne,ns] = size(blocks);

```

```

off = 0;

```

20

```

for i=1:ne,

```

```

    if (bundle)

```

```

        deigvs(i)= trace(B(off+1:off+sum(blocks(i,:)),off+1:off+sum(blocks(i,:))))/sum(blocks(i,

```

```

else
    deigvs = 0*eigvs;
end
for j=1:ns,
    if (blocks(i,j)>0)
        B(off+1:k,off+1:off+blocks(i,j))= deigvs(i)*eye(k-off,blocks(i,j));
        off = off + blocks(i,j);
    end
end
end
end

```

```

function parameters(A,eigvs,serge,type)

```

```

% PARAMETERS initializes the parameters for an instance of a
%      minimization problem. This functions must be executed
%      before any attempts to do any jordan structure minimizations.
%

```

```

% PARAMETERS(A,eigvs,serge,type)

```

```

% A is expected to be a square matrix
% eigvs is expected to be a 1xp matrix of eigenvalues, assumed to
% be distinct.

```

```

% serge is expected to be a pxq matrix of serge characteristics.
%      structures (zero padded).

```

```

% for example, if one wishes to describe a matrix that
% has a substructure of
%      J_4(1.0)xJ_2(1.0)xJ_2(1.0)xJ_3(2.0)xJ_3(2.0)

```

```

% then one has
%      eigvs = [1.0 2.0]

```

```

%      serge = [4 2 2; 3 3 0]

```

```

% type is expected to be the string 'bundle' or 'orbit'

```

```

%

```

```

% role sets up the global parameters used at all levels of computation.

```

```

global FParameters;

```

```

FParameters = [];

```

```

FParameters.A = A;

```

```

    FParameters.eigs = eigvs;
    serge = fix(max(serge,0*serge));
% convert the serge characteristics into weyr characteristics.
    blocks = [];
    for j=1:size(serge,1),
        for k=1:max(serge(j,:)),
            blocks(j,k) = sum(serge(j,:)>=k);
        end
    end
    FParameters.blocks = blocks;
    if (strcmp(lower(type), 'orbit'))
        FParameters.bundle = 0;
    elseif (strcmp(lower(type), 'bundle'))
        FParameters.bundle = 1;
    else
        'Either bundle or orbit must be specified.';
    end

```

```

function [M,eigvs] = post(Y)
% POST Computes the matrix which corresponds to the one nearest to
% FParameters.A with the jordan structure FParameters.blocks
% and eigenspace Y.
%
% [M,eigvs] = post(Y)
% Y is expected to be a stiefel point ( $Y' * Y = I$ )
% M will be the same size as FParameters.A with jordan structure
% encoded by FParameters.blocks, eigenvalues of eigvs, and eigenspace
% of Y.
% eigvs will be the vector of jordan eigenvalues of M.
global FParameters;

```

```

    A = FParameters.A;
    [B,eigvs] = Block(Y);
    M = A - (A*Y - Y*B)*Y';

```

Sources from the `sg_min/examples/tracemin` subdirectory:

```
function f = F(Y)
% F    Computes the energy associated with the stieffel point Y.
%      In this case, by the equation  $f = \text{trace}(Y'AY)/2$ .
%
%       $f = F(Y)$ 
%      Y is expected to satisfy  $Y'*Y=I$ 
%
% role  objective function, the routine called to compute the objective
%       function.
global FParameters;
A = FParameters.A;
f = sum(sum(real(conj(Y).*(A*Y))))/2;
```

```
function df = dF(Y)
% dF   Computes the differential of F, that is,
%       de satisfies  $\text{real}(\text{trace}(H'*df)) = d/dx (F(Y+x*H))$ .
%
%        $df = DF(Y)$ 
%       Y is expected to satisfy  $Y'*Y = I$ 
%       df is the same size as Y
%
% role  objective function, this is the routine called to compute the
%       differential of F.
global FParameters;
A = FParameters.A;
df = A*Y;
```

```
function ddf = ddF(Y,H)
% ddF  Computes the second derivative of F, that is,
%        $ddf = d/dx dF(Y+x*H)$ .
%
%        $ddf = DDF(Y,H)$ 
```

```

%      Y is expected to satisfy  $Y' * Y = I$ 
%      H is expected to be the same size as Y
%      ddf will be the same size as Y
%
% role  objective function, the function is called to apply the
%       unconstrained hessian of F to a vector.
global FParameters;
A = FParameters.A;

ddf = A * H;

```

```

function Y = guess(P)
% GUESS provides the initial starting guess Y for energy minimization.
%      Y will be  $n \times P$  where FParameters.A is an  $n \times n$  matrix. Uses a
%      random matrix.
%
%      Y = GUESS
%      Y will satisfy  $Y' * Y = I$ 
%
% role  objective function, produces an initial guess at a minimizer
%       of F.
global FParameters;
inA = FParameters.A;
[n,n] = size(inA);
[Y,r] = qr(randn(n,P),0);

```

```

function parameters(A)
% PARAMETERS initializes the parameters for an instance of a
%      minimization problem.
%
%      PARAMETERS(A)
%      A is expected to be a square matrix
%
% role  sets up the global parameters used at all levels of computation.

```

```
global FParameters;  
FParameters = [];  
FParameters.A = A;
```

10

```
function K = Kinetic(n)  
    K = 2*eye(n) - diag(ones(1,n-1),-1) - diag(ones(1,n-1),1);
```

Sources from the `sg_min/examples/ldatoy` subdirectory:

```
function f= F(Y)
% F(Y) computes the energy of a toy lda particle in a box configuration.
%  $F(Y) = 1/2*\text{trace}(Y'*A*Y)+c*1/4*\text{sum}(\text{sum}(Y.^2,2).^2,1)$ .
%
%  $f = F(Y)$ 
% Y is expected to satisfy  $Y'*Y=I$ 
%
% role objective function, the routine called to compute the objective
% function.
    global FParameters;
    A = FParameters.A;
    c = FParameters.c;
    f = trace(Y'*(A*Y))/2+c*sum(sum(Y.^2,2).^2,1)/4;
```

```
function df = dF(Y)
% dF Computes the differential of F, that is,
% df satisfies  $\text{real}(\text{trace}(H'*df)) = d/dx (F(Y+x*H))$ .
%
%  $df = DF(Y)$ 
% Y is expected to satisfy  $Y'*Y = I$ 
% df is the same size as Y
%
% role objective function, this is the routine called to compute the
% differential of F.
    global FParameters;
    A = FParameters.A;
    c = FParameters.c;
    df = (A*Y)+c*(sum(Y.^2,2)*ones(1,size(Y,2))).*Y;
```

```
function ddf = ddF(Y,H)
% ddf Computes the second derivative of F, that is,
%  $ddf = d/dx dF(Y+x*H)$ .
```



```

%
%  $ddf = DDF(Y,H)$ 
%  $Y$  is expected to satisfy  $Y' * Y = I$ 
%  $H$  is expected to be the same size as  $Y$ 
%  $ddf$  will be the same size as  $Y$ 
%
% role objective function, the function is called to apply the
% unconstrained hessian of  $F$  to a vector.
global FParameters;
A = FParameters.A;
c = FParameters.c;
ddf = (A * H);
ddf = ddf + c * ( (sum(Y.^2,2) * ones(1,size(Y,2))) * H );
ddf = ddf + c * ( (sum(2 * Y * H,2) * ones(1,size(Y,2))) * Y );

```

```

function Y = guess(p)
% GUESS provides the initial starting guess  $Y$  for energy minimization.
%
%  $Y = GUESS$ 
%  $Y$  will satisfy  $Y' * Y = I$ 
%
% role objective function, produces an initial guess at a minimizer
% of  $F$ .
global FParameters;
A = FParameters.A;
n = length(A);
[V,D] = eig(A); D = diag(D);
[D,I] = sort(D); V = V(:,I);
Y = V(:,1:p);

```

```

function parameters(A,c)
% PARAMETERS initializes the parameters for an instance of a
% toy lda problem with laplacian  $A$  and self coupling
% constant  $c$ .

```

```
%  
% PARAMETERS(A,c)  
% A is expected to be a square symmetric matrix  
%  
% role sets up the global parameters used at all levels of computation.
```

10

```
global FParameters;  
FParameters = [];  
FParameters.A = A;  
FParameters.c = c;
```

```
function K = Kinetic(n)
```

```
    K = 2*eye(n) - diag(ones(1,n-1),-1) - diag(ones(1,n-1),1);
```

Sources from the `sg_min/examples/simschur` subdirectory:

function `f = F(Q)`

% F Computes the objective value associated with the stiefel point Q.

% In this case, by the equation

% $f = \text{sum}(\text{sum}(qAql.^2))/2 + \text{sum}(\text{sum}(qBql.^2))/2;$

*% where $qAql = \text{masked out part of } Q'*A*Q \text{ and } qBql \text{ similarly.}$*

%

% $f = F(Q)$

*% Q is expected to satisfy $Q'*Q=I$*

%

% role objective function, the routine called to compute the objective

10

% function.

global FParameters;

A = FParameters.A;

B = FParameters.B;

mask = FParameters.Mask;

qAq = Q'*A*Q;

qBq = Q'*B*Q;

qAql = (1-mask).*qAq;

qBql = (1-mask).*qBq;

f = sum(sum(qAql.^2))/2+sum(sum(qBql.^2))/2;

20

function `df = dF(Q)`

% dF Computes the differential of F, that is,

*% df satisfies $\text{real}(\text{trace}(H'*df)) = d/dx (F(Q+x*H)).$*

%

% $df = DF(Q)$

*% Q is expected to satisfy $Q'*Q = I$*

% df is the same size as Q

%

% role objective function, this is the routine called to compute the

% differential of F.

10

global FParameters;

A = FParameters.A;

```

B = FParameters.B;
mask = FParameters.Mask;
qAq = Q'*A*Q;
qBq = Q'*B*Q;
qAql = (1-mask).*qAq;
qBql = (1-mask).*qBq;
df = Q*(qAq'*qAql+qAq*qAql'+ qBq'*qBql+qBq*qBql');

```

```

function ddf = ddF(Q,dQ)

```

```

% ddF  Computes the second derivative of F, that is,

```

```

%      ddf = d/dx dF(Q+x*dQ).

```

```

%

```

```

%      ddf = DDF(Q,dQ)

```

```

%      Q is expected to satisfy Q'*Q = I

```

```

%      dQ is expected to be the same size as Q

```

```

%      ddf will be the same size as Q

```

```

%

```

```

% role  objective function, the function is called to apply the

```

10

```

%      unconstrained hessian of F to a vector.

```

```

    global FParameters;

```

```

    A = FParameters.A;

```

```

    B = FParameters.B;

```

```

    mask = FParameters.Mask;

```

```

    qAq = Q'*A*Q;

```

```

    qBq = Q'*B*Q;

```

```

    qAql = (1-mask).*qAq;

```

```

    qBql = (1-mask).*qBq;

```

```

    dqAq = dQ'*A*Q+Q'*A*dQ;

```

20

```

    dqBq = dQ'*B*Q+Q'*B*dQ;

```

```

    dqAql = (1-mask).*dqAq;

```

```

    dqBql = (1-mask).*dqBq;

```

```

    ddf = dqAq'*qAql+dqAq*qAql'+ dqBq'*qBql+dqBq*qBql';

```

```

    ddf = ddf+qAq'*dqAql+qAq*dqAql'+ qBq'*dqBql+qBq*dqBql';

```

```

    ddf = Q*ddf;

```

```

    ddf = ddf+dQ*(qAq'*qAql+qAq*qAql'+ qBq'*qBql+qBq*qBql');

```

```

function Y = guess()
% GUESS provides the initial starting guess Y for energy minimization.
%
%   Y = GUESS
%   Y will satisfy  $Y'^*Y = I$ 
%
% role objective function, produces an initial guess at a minimizer
%   of F.
%       global FParameters;
%       A = FParameters.A;
%       B = FParameters.B;
% use average
%       [Y,T] = schur(A+B);
% using random guess
%       Y = randn(size(Y));
% use identity
%       Y = eye(size(A));

```

```

function parameters(A,B)
% PARAMETERS initializes the parameters for an instance of a
%       a Schilders problem.
%
%   PARAMETERS(A,B)
%   A,B are expected to be real matrices of the same size.
%
% role sets up the global parameters used at all levels of computation.
%       global FParameters;
%       FParameters.A = A;
%       FParameters.B = B;
%       n = length(A);
%       Mask = triu(ones(n));
%       for l=1:2:n-1,
%           Mask(l+1,l)=1;

```

end

FParameters.Mask = Mask;

Sources from the `sg_min/examples/indscal` subdirectory:

```
function f = F(Y)
% F    Computes the energy associated with the stiefel point Y.
%      In this case, by the equation  $f = \sum_i \|S_i - Y D_i Y\|^2/2$ .
%
%       $f = F(Y)$ 
%      Y is expected to satisfy  $Y' * Y = I$ 
%
% role objective function, the routine called to compute the objective
%      function.
global FParameters;
Ss = FParameters.Ss;
df = 0;
for k=1:size(Ss,3),
    S = Y' * Ss(:,k) * Y; D = max(diag(S),0);
    f = sum(sum( (S - diag(D)).^2 ))/2;
end
```

```
function df = dF(Y)
% dF    Computes the differential of F, that is,
%      df satisfies  $\text{real}(\text{trace}(H' * df)) = d/dx (F(Y + x * H))$ .
%
%       $df = DF(Y)$ 
%      Y is expected to satisfy  $Y' * Y = I$ 
%      df is the same size as Y
%
% role objective function, this is the routine called to compute the
%      differential of F.
global FParameters;
Ss = FParameters.Ss;
df = 0;
for k=1:size(Ss,3),
    S = Y' * Ss(:,k) * Y; D = max(diag(S),0);
    df = df + 2 * Y * S * (S - diag(D));
```

end

function ddf = ddF(Y,H)

% ddF Computes the second derivative of F, that is,

*% ddf = d/dx dF(Y+x*H).*

%

% ddf = DDF(Y,H)

*% Y is expected to satisfy Y'*Y = I*

% H is expected to be the same size as Y

% ddf will be the same size as Y

%

% role objective function, the function is called to apply the

10

% unconstrained hessian of F to a vector.

global FParameters;

Ss = FParameters.Ss;

ddf = 0;

for k=1:size(Ss,3),

S = Y'*Ss(:,k)*Y; D =max(diag(S),0);

dS = H'*Ss(:,k)*Y+Y'*Ss(:,k)*H;

dD = (D>0).*diag(dS);

ddf = ddf+2*H*S*(S - diag(D));

ddf = ddf+2*Y*dS*(S - diag(D));

20

ddf = ddf+2*Y*S*(dS - diag(dD));

end

function Y0 = guess()

% GUESS provides the initial starting guess Y for an F minimization.

%

% Y = GUESS

*% Y will satisfy Y'*Y = I*

%

% role objective function, produces an initial guess at a minimizer

% of F.

global FParameters;


```

        Ss = FParameters.Ss;
        Q = 0;
% take an average
%     av = Ss(:, :, 1);
%     for k=2:size(Ss,3)
%         av = av + Ss(:, :, k);
%     end
%     [Y0,D] = eig(av);
% using random guess
%     Y0 = randn(size(av));
% using the identity
        Y0 = eye(size(Ss(:, :, 1))) + 1e-1 * randn(size(Sz(:, :, 1)));

```

function parameters(varargin)

```

% PARAMETERS initializes the parameters for an instance of a
%     INDFCAL minimization problem.
%
%     PARAMETERS(S1,S2,...,Sn)
%     Si are expected to be symmetric matrices of equal size.
%
% role sets up the global parameters used at all levels of computation.

```

```

global FParameters;
FParameters = [];
FParameters.Ss = cat(3,varargin{:});

```

function [A,B] = exact()

```

    A = diag(1:10);
    B = diag((1:10).^2);

```

function [A,B] = noisy()

```

    [A,B] = exact;
    randn('state',0);

```

```
A = A + 1e-4*randn(size(A));
```

```
B = B + 1e-4*randn(size(B));
```

Bibliography

- [1] T.A. Arias, A. Edelman and S.T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, to appear.
- [2] I. Andersson. Experiments with the conjugate gradient algorithm for the determination of eigenvalues of symmetric matrices. Technical Report UMINF-4.71, University of Umeå, Sweden, 1971.
- [3] T. A. Arias. Multiresolution analysis of electronic structure: semicardinal and orthogonal wavelet bases. Submitted to *Reviews of Modern Physics*, 1997.
- [4] T. A. Arias, A. Edelman, and S. T. Smith. Curvature in conjugate gradient eigenvalue computation with applications to materials and chemistry calculations. In J. G. Lewis, editor, *Proceedings of the 1994 SIAM Applied Linear Algebra Conference*, pages 233–238, Philadelphia, 1994. SIAM.
- [5] T.A. Arias, K.J. Cho, P. Lam, and M.P. Teter. Wavelet transform representation of the electronic structure of materials. In Rajiv K. Kalia and Priya Vashishta, editors, *Proceedings of the '94 Mardi Gras Conference: Toward Teraflop Computing and New Grand Challenge Applications*, page 23, Commack, New York, 1995. Nova Science Publishers.
- [6] M. Arioli, I. S. Duff, and D. Ruiz. Stopping criteria for iterative solvers. Report RAL-91-057, Central Computing Center, Rutherford Appleton Laboratory, 1992.

- [7] S. Bertoluzza and G. Naldi. A wavelet collocation method for the numerical solution of partial differential equations. *Applied Computational and Harmonic Analysis*, 3:1–9, 1996.
- [8] G. Beylkin, R.R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms. *Commun. Pure and Appl. Math*, 44:141–183, 1991.
- [9] G. Beylkin and J. M. Keiser. On the adaptive numerical solution of nonlinear partial differential equations in wavelet bases. *PAM Report*, 262, 1995.
- [10] G. Beylkin and J. M. Keiser. On the adaptive numerical solution of nonlinear partial differential equations in wavelet bases. *Journal of Computational Physics*, 132:233–259, 1997.
- [11] G. Beylkin and N. Saito. Wavelets, their autocorrelation functions, and multiresolution representation of signals. *Expanded abstract in Proceedings ICASSP-92*, 4:381–384, 1992.
- [12] G. Beylkin and N. Saito. Multiresolution representations using the auto-correlation functions of compactly supported wavelets. *IEEE Transactions on Signal Processing*, 41:3584–3590, 1993.
- [13] J. W. Bruce and P. J. Giblin. *Curves and Singularities (Second Edition)*. Cambridge University Press, Cambridge, England, 1992.
- [14] I. Chavel. *Riemannian Geometry—A Modern Introduction*. The Cambridge University Press, 1993.
- [15] K. Cho, T.A. Arias, J.D. Joannopoulos, and Pui K. Lam. Wavelets in electronic structure calculations. *Physical Review Letters*, 71:1808, 1993.
- [16] S.-N. Chow, C. Li, and D. Wang. *Normal Forms and Bifurcation of Planar Vector Fields*. Cambridge, University Press, 1994.
- [17] A. Cohen and R. Danchin. Multiscale approximation of vortex patches. *preprint*, 1997.

- [18] S. Dahlke, W. Dahmen, R. Hochmuth, and R. Schneider. Stable multiscale bases and local error estimation for elliptic problems. *Applied Numerical Mathematics*, 23(1):21–47, 1997.
- [19] I. Daubechies. Orthonormal bases of compactly supported wavelets. *J. Comm. Pure Appl. Math.*, 41:909–996, Oct 1988.
- [20] I. Daubechies. *Ten Lectures on Wavelets*. Number 61 in CBMS/NSF Series in Applied Math. SIAM, 1992.
- [21] J. de Leeuw and W. Heiser. Theory of multidimensional scaling. In P. R. Krishnaiah and L. N. Kanal, editors, *Handbook of Statistics, Vol 2*, pages 285–316. North-Holland Publishing Co., 1982.
- [22] J. W. Demmel. *A Numerical Analyst’s Jordan Canonical Form*. PhD thesis, Univ. of California at Berkeley, Berkeley, CA, USA, 1983.
- [23] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.
- [24] G. Deslauriers and S. Dubuc. Symmetric iterative interpolation processes. *Constr. Approx.*, 5(1):49–68, 1989.
- [25] M. P. do Carmo. *Differential geometry of curves and surfaces*. Prentice–Hall, 1976.
- [26] D. L. Donoho. Interpolating wavelet tranforms. Preprint, Department of Statistics, Stanford University, 1992.
- [27] S. Dubuc. Interpolation through and iterative scheme. *J. Math. Anal. Appl.*, 114:185–204, 1986.
- [28] A. Edelman and E. Elmroth. Personal communication, January 1997.
- [29] A. Edelman and Y. Ma. Non-generic eigenvalue perturbations of jordan blocks. *Linear Algebra Appl.*, 273:45–63, 1998.
- [30] A. Edelman and Y. Ma. Staircase failures explains by orthogonal versal forms. *Submitted to SIAM J. Matrix Ana. and Appl.*, 1998.

- [31] A. Edelman and S. T. Smith. On conjugate gradient-like methods for eigen-like problems. *BIT*, to appear. See also *Proc. Linear and Nonlinear Conjugate Gradient-Related Methods*, eds. Loyce Adams and J. L. Nazareth. SIAM, Philadelphia, PA, 1996.
- [32] L. Eldén. Algorithms for the regularization of ill-conditioned least-squares problems. *BIT*, 17:134–145, 1977.
- [33] T. F. Fairgrieve. The application of singularity theory to the computation of Jordan Canonical Form. *Master Thesis at Computer Science in Univ. of Toronto*, 1986.
- [34] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, 2d edition, 1987.
- [35] J. Frohlich and K. Schneider. An adaptive wavelet galerkin algorithm for one- and two-dimensional flame. *Computations. Eur. J. Mech*, 13:439–471, 1994.
- [36] Z. Fu and E. M. Dowling. Conjugate gradient eigenstructure tracking for adaptive spectral estimation. *IEEE Trans. Signal Processing*, 43(5):1151–1160, 1995.
- [37] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, New York, 2d edition, 1981.
- [38] S. Goedecker and O.V. Ivanov. Linear scaling solution of the coulomb problem using wavelets. *Preprint (see <http://xxx.lanl.gov/abs/physics/9701024>)*, Submitted Jan 1997.
- [39] G. H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, 2d edition, 1989.
- [40] G. H. Golub and J. H. Wilkinson. Ill-conditioned eigensystems and the computation of the Jordan canonical form. *SIAM Review*, 18:578–619, 1976.
- [41] S. Helgason. *Differential Geometry, Lie Groups, and Symmetric Spaces*. Academic Press, New York, 1978.

- [42] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Physical Review*, 136:B864–B871, 1964.
- [43] T. Kato. *Perturbation Theory for Linear Operators*. Springer-Verlag, New York, NY, USA, 1966.
- [44] S. Kobayashi and K. Nomizu. *Foundations of Differential Geometry*, volume 1 and 2. Wiley, New York, 1969.
- [45] W. Kohn and L.J. Sham. Self-consistent equations including exchange and correlation effects. *Physical Review*, 140:A1133, 1965.
- [46] J. Liandrat, V. Pierrier, and P. Tchamitchian. Numerical resolution of the regularized Burgers equation using the wavelet transform. *Tech. Report CPT-89/P. 2320 - Center of Theoretical Physics, Marseille, France*, 1989.
- [47] J. Liandrat and P. Tchamitchian. Resolution of the 1d regularized burgers equation using a spatial wavelet approximation. report, Technical Report 90-83, NASA ICASE, December 1990.
- [48] C. F. Van Loan. On estimating the condition of eigenvalues and eigenvectors. *Linear Algebra Appl.*, 88/89:715–732, 1987.
- [49] A. N. Malyshev. On wilkinson’s problem. Technical report, Report 140, Dept. of Informatics, University of Bergen, Norway, November 1997.
- [50] S. G. Nash and Ariela Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, New York, 1995.
- [51] M.C. Payne, M.P. Teter, D.C. Allan, T.A. Arias, and J.D. Joannopoulos. Iterative minimization techniques for *ab initio* total energy calculations: molecular dynamics and conjugate gradients. *Reviews of Modern Physics*, 64:1045, 1992.
- [52] E. Polak. *Computational Methods in Optimization*. Academic Press, New York, 1971.

- [53] N. Saito and G. Beylkin. Multiresolution representations. *Applied and Computational Harmonic Analysis*, 41:3584, 1993.
- [54] A. H. Sameh and J. A. Wisniewski. A trace minimization algorithm for the generalized eigenvalue problem. *SIAM J. Numerical Analysis*, 19:1243–1259, 1982.
- [55] W. H. A. Schilders. Personal communication, September 1997.
- [56] S. T. Smith. Optimization techniques on Riemannian manifolds. *Fields Institute Communications*, 3:113–146, 1994.
- [57] I. Štich, R. Car, M. Parrinello, and S. Baroni. Conjugate gradient minimization of the energy functional: A new method for electronic structure calculation. *Phys. Rev. B.*, 39:4997–5004, 1989.
- [58] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, Cambridge, Massachusetts, 1996.
- [59] S. H. Strogatz. *Nonlinear dynamics and Chaos: with applications to physics, biology, chemistry, and engineering*. Addison-Wesley, 1994.
- [60] J.-G. Sun. A note on simple non-zero singular values. *Journal of Computational Mathematics*, 6:258–266, 1988.
- [61] J.-G. Sun. Sensitivity analysis of multiple eigenvalues. I. *Journal of Computational Mathematics*, 6:28–38, 1988. Cited in [62].
- [62] J.-G. Sun. A note on local behavior of multiple eigenvalues. *SIAM Journal on Matrix Analysis and Applications*, 10:533–541, 1989.
- [63] L. N. Trefethen. Pseudospectra of matrices. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1991*. Longman Scientific and Technical, Harlow, Essex, 1992.
- [64] C.J. Tymczak and X.-Q. Wang. Orthonormal wavelet bases for quantum molecular dynamics. *Physical Review Letters*, 78:3654, 1997.

- [65] S. Wei and M.Y Chou. Wavelets in self-consistent electronic structure calculations. *Physical Review Letters*, 76:2650, 1996.
- [66] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, England, 1965.
- [67] J. H. Wilkinson. Sensitivity of eigenvalues. *Util. Math.*, 25:5–76, 1984.
- [68] J. H. Wilkinson. Sensitivity of eigenvalues II. *Util. Math.*, 30:243–286, 1986.
- [69] Y.-C. Wong. Differential geometry of Grassmann manifolds. *Proc. Natl. Acad. Sci. USA*, 57:589–594, 1967.
- [70] D. Yeşiltepe and T. A. Arias. Synthesis of wavelet theory with the multigrid algorithm. Condensed Matter Physics, MIT, 1996.