INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Rational approximation of transfer functions in the hyperion software

José Grimm

No ????

Septembre 2000

THÈME 4

Rapport
de recherche

N 0249-6399

Rational approximation of transfer functions in the hyperion software

José Grimm [*]

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet MIAOU

Rapport de recherche no ???? — Septembre 2000 — 241 pages

Abstract: The objective of this paper is to explain how rational approximation is performed within the HYPERION software. This work is divided into three parts. In the first part, we explain the theory underlying the algorithm: given some Fourier coefficients, find a rational transfer function of McMillan degree n of best approximation in the l^2-sense to the corresponding Fourier series. This gives a certain number of equations for stationary points and we detail in the second part how they can be efficiently solved numerically, using techniques of automatic differentiation. In the last part, we further discuss the complexity of the implementation thus obtained.

Key-words: System theory, stable linear systems, transfer function, McMillan degree, Schur parameters, automatic differentiation, code generation, complexity.

The author wishes to thank all members of the Safir and Miaou teams.

[*] This work was mainly done while the author was in the Safir team.

Approximation rationelle de fonctions de transfert dans le logiciel hyperion

Résumé : Dans cet article, nous supposons donné un certain nombre de coefficients de Fourier, qui est censé représenter la fonction de transfert d'un système linéaire stable, rationnel de degré de McMillan n. Nous présentons un algorithme qui permet de retrouver ce système. L'article contient essentiellement trois parties : d'une part, une étude théorique du problème qui nous donne un ensemble d'équations, puis une implémentation effective de ces équations, et de leur dérivées via des techniques de différentiation automatique, et finalement, une discussion sur la complexité de cette implémentation dans le logiciel HYPERION.

Mots-clés : Théorie des systèmes, systèmes linéaires stables, fonction de transfert, degré de McMillan, paramètres de Schur, différentiation automatique, génération automatique de code, complexité.

Chapter 1

Introduction

The aim of this paper is the following: given the Fourier coefficient of a stable, rational, transfer function, find the function as the quotient of two polynomials. This is the heart of the HYPERION software. We shall distinguish the scalar and the non-scalar case, because there is a fundamental difference in the topological structure of the set of functions we are looking for, see for instance [3]

Let $\mathcal{H}$ be the transfer function, this is a $p \times m$ matrix. The scalar case is when $m = 1$ or $p = 1$, the non-scalar case is when $m \neq 1$ and $p \neq 1$. In the scalar case, we can assume $m = p = 1$. The problem is then: find two polynomials p and q such that $\mathcal{H} = p/q$, and $\deg(q) = n$, $\deg(p) < n$. In the non-scalar case, we write $\mathcal{H} = PQ^{-1}$, where P and Q are matrices. There are different kinds of factorisations of this form. We shall chose Q inner, of McMillan degree n, and $P \in H^\infty$. This is called the Douglas-Shapiro-Shields factorisation.

In this chapter we shall give some definitions, some well-known theorems, and some general properties needed later on. We shall explain what an inner matrix is, how to define the McMillan degree of a matrix, etc. In the scalar case, a necessary condition for p/q to be of degree n is that p and q are coprime. The same is also required in the non-scalar case, but the definition of coprime is more complex.

It happens, in general, that the Fourier coefficients of $\mathcal{H}$ are not precise, that only a finite number of coefficients are given, that the physical device that gives $\mathcal{H}$ is only an approximation of a linear system, etc. For all these reasons, it is impossible to have an equality $\mathcal{H} - PQ^{-1} = 0$. Hence, we try to minimise the norm of the difference. If the norm well-chosen, then the solution is defined by $P = L(Q)$, where Q minimises the function ψ. The aim of the chapter 2 is to define the functions L and ψ, and study their properties.

In Chapter 3, we study in great detail the set of inner matrices of given McMillan degree. This set is a manifold, and we shall define charts, associated to the Schur algorithm. Hence, given u, almost every inner function Q can be written as $Q = f(u, y)$, for some function f, where y lies in open set of $\mathbb{R}^k$. Minimising $\psi(Q)$ is then the same as minimising $\psi_1(y)$.

In Chapter 4, we explain how the formulas given in the previous chapters are implemented in HYPERION in an efficient way. In the last chapter, we compute the complexity of the implementation, and of equivalent formulas.

1.1 Definitions

Throughout this paper, if z is a complex number, we shall denote the complex conjugate of z by $\overline{z}$. The notations $\Re z$ and $\Im z$ stand for the real and imaginary parts of z.

If A is a matrix, A^t will be the transpose of A, and A^* the transpose conjugate of A. If q is a polynomial, then $\tilde{q}$ will be $z^n \overline{q}(1/z)$. Unless stated otherwise, n is the degree of q, or is clear from the context; in particular, if p is the remainder of the Euclidean division of A by B, then $\tilde{p} = z^n \overline{p}(1/z)$ where n is the degree of B minus one. If $q = \sum_{k=0}^n q_k z^k$, then $\tilde{q} = \sum_{k=0}^n \overline{q_k} z^{n-k}$. We have $\widetilde{AB} = \tilde{A}\tilde{B}$, so that, if $q = \prod(z - \alpha_i)$, then $\tilde{q} = \prod(1 - \overline{\alpha}_i z)$. Note that $\tilde{q}$ has degree exactly n if and only if $q(0) \neq 0$. If A is a matrix of polynomials of degree $\leq n$, then $\tilde{A}$ is the matrix of all $\tilde{A}_{ji}$, it is also $z^n A^*(1/z)$.

If A and B are two matrices, two polynomials, or two matrices of polynomials, we define

$$\langle A \,|\, B \rangle = \frac{1}{2\pi} \int_0^{2\pi} \mathrm{Tr}[\overline{A(e^{i\theta})}^t B(e^{i\theta})] d\theta, \tag{1.1}$$

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

Here Tr stands for the trace of the matrix. If the entries of A and B are A_{ij} and B_{ij}, then the trace of $A^t B$ is $\sum_{ij} A_{ij} B_{ij}$, so that $\langle A \,|\, B \rangle = \sum \langle A_{ij} \,|\, B_{ij} \rangle$. Note that if A and B are vectors, then $\mathrm{Tr}(A^* B)$ is just $A^* B$.

If A and B are polynomials, say $A = \sum a_k z^k$ and $B = \sum b_k z^k$, then

$$\langle A \,|\, B \rangle = \frac{1}{2\pi} \int_0^{2\pi} \sum_{kl} \overline{a_k} b_l e^{i\theta(l-k)} d\theta = \sum_k \overline{a_k} b_k.$$

Finally, if A is a matrix of polynomials with entries $A_{ij} = \sum_k a_{ijk} z^k$, then $\langle A \,|\, A \rangle = \sum_{ijk} |a_{ijk}|^2$, and $\langle A \,|\, B \rangle$ is the Hermitian form associated to this quadratic form.

A consequence of equation (1.1) that will be used throughout this paper is

$$\langle AB \,|\, C \rangle = \langle A z^n \,|\, \tilde{B} C \rangle \qquad (\tilde{B} = z^n \overline{B}(1/z)).$$

1.2 H^p spaces

Let $\mathbb{T}$ be the set of all complex numbers z with $|z| = 1$ (the unit circle in the complex plane), $\mathbb{U}$ be the set of all complex numbers z with $|z| < 1$ (the open unit disk) and $\overline{\mathbb{U}}$ the set of all complex numbers with $|z| \leq 1$ (the closed unit disk).

If f is analytic in $\mathbb{U}$, we define

$$M_p(f, r) = \left\{ \frac{1}{2\pi} \int_0^{2\pi} \left| f(re^{i\theta}) \right|^p d\theta \right\}^{1/p} \qquad (0 < p < \infty),$$

$$M_\infty(f, r) = \sup_\theta |f(re^{i\theta})|.$$

Let $\|f\|_p$ be the limit (if it exists) of $M_p(f, r)$ as $r \to 1$, and H^p the set of all functions analytic in $\mathbb{U}$ for which this quantity is finite. It can be shown, if $p \geq 1$, that $M_p(f, r)$ is a monotonically increasing function of r, so that $\|f\|_p$ exists, and H^p is a Banach space. If $1 \leq s \leq p < \infty$ then

$$H^\infty \subset H^p \subset H^s.$$

One can show (for instance [17, Th 17.12]) that if $f \in H^1$ (hence for $f \in H^p$, $p \geq 1$), that

$$f^*(e^{i\theta}) = \lim_{r \to 1} f(re^{i\theta})$$

exists at almost every point in $\mathbb{T}$, and

$$\lim_{r \to 1} \frac{1}{2\pi} \int_0^{2\pi} |f^*(e^{i\theta}) - f(re^{i\theta})| d\theta = 0.$$

Moreover, f is the Poisson integral, and the Cauchy integral of f^*. This can be sharpened in the case of of H^2 (cf [17, Th 17.10])

- If f is analytic in $\mathbb{U}$, $f(z) = \sum_{k=0}^{\infty} a_k z^k$, then $f \in H^2$ if and only if $\sum |a_k|^2 < \infty$. We have $\|f\|_2^2 = \sum |a_k|^2$.
- If $f \in H^2$, the function f^* is in $L^2(\mathbb{T})$, the n-th Fourier coefficient of f is a_n for $n \geq 0$, zero otherwise, and the mapping $f \to f^*$ is an isometry from H^2 onto the subset of $L^2(\mathbb{T})$ formed of those functions for which Fourier coefficients with negative index vanish.
- If $f(z) = \sum a_k z^k$, $g(z) = \sum b_k z^k$ then

  $$\frac{1}{2\pi} \int_0^{2\pi} \overline{f^*(e^{i\theta})} g^*(e^{i\theta}) d\theta = \sum_k \overline{a_k} b_k$$

  is a scalar product in H^2 that makes it a Hilbert space. Note that this is $\langle f^* \,|\, g^* \rangle$, so that H^2 is nothing else than the completion of the set of polynomials for the inner product defined by (1.1).
- If $P_r(\theta) = \dfrac{1 - r^2}{1 - 2r\cos\theta + r^2}$ is the Poisson kernel, then f is the Poisson integral of f^*, i.e.

  $$f(re^{i\theta}) = \frac{1}{2\pi} \int_0^{2\pi} P_r(\theta - t) f^*(e^{it}) dt.$$
- f is the Cauchy integral of f^*, that is

  $$f(z) = \frac{1}{2i\pi} \int_\Gamma \frac{f^*(\xi)}{\xi - z} d\xi$$

  where Γ is the positively oriented unit circle. If $|\alpha| < 1$, this is

  $$f(\alpha) = \langle \frac{1}{1 - z\overline{\alpha}} \,|\, f^* \rangle.$$

Recall that $L^p(\mathbb{T})$ is the set of functions f defined on $\mathbb{T}$ such that

$$\|f\|_p = \left\{ \frac{1}{2\pi} \int_0^{2\pi} \left| f(e^{i\theta}) \right|^p d\theta \right\}^{1/p} < \infty$$

and $\|f\|_p$ makes this set a Banach space for $p \geq 1$. If $1/p + 1/q = 1$, $f \in L^p$ and $g \in L^q$, then $fg \in L^1$, so that the integral of $|fg|$ is defined. The n-th Fourier coefficient of f is defined as

$$c_n = \frac{1}{2\pi} \int_0^{2\pi} f(e^{i\theta}) e^{-in\theta} d\theta.$$

Assume $c_n = 0$ for $n < 0$. Then $F(z) = \sum c_n z^n$ is analytic in $\mathbb{U}$. If $p > 1$ and $f \in L^p$, then $F \in H^p$. Hence, if $1/p + 1/q = 1$, $f \in H^p$ and $g \in H^q$, the product fg is in H^1.

Lemma 1
Assume $p > 1$ and $g \in L^p(\mathbb{T})$ and

$$h(\omega) = \frac{1}{2\pi} \int_0^{2\pi} g(e^{i\theta}) \frac{d\theta}{e^{i\theta} - \omega}$$

vanishes on Ω, which is a subset of $\mathbb{U}$ that has an accumulation point in $\mathbb{U}$. Then there exists a function $g_1 \in H^p$, such that $g_1^*(e^{i\theta}) = g(e^{-i\theta})$ almost everywhere on $\mathbb{T}$.

Proof. We have

$$h(\omega) = \sum_{k=0}^{\infty} \omega^k \frac{1}{2\pi} \int_0^{2\pi} g(e^{i\theta}) e^{-i(k+1)\theta} d\theta.$$

Hence h is analytic in $\mathbb{U}$. The assumption says that h is identically zero, so that Fourier coefficients of g with positive index are zero. If we replace g by $g(e^{-i\theta})$, we get a function in H^p. □

If f is a function of z, we shall define

$$\check{f}(z) = (1/z)\overline{f}(1/z).$$

If f is analytic in $\mathbb{U}$, then $\check{f}$ is analytic outside $\mathbb{U}$. The set H_2^- will be the set of all $\check{f}$ for f in H^2. A function f is in H_2^- if and only if it has the form $\sum_{k\geq 0} a_k/z^{k+1}$ where $\sum |a_k|^2 < \infty$. We can define f^* as previously ($f^*(e^{i\theta}) = \lim_{r\to 1} f(re^{i\theta})$, $r > 1$), and the mapping $f \to f^*$ is a isometry from H_2^- into a subspace of $L^2(\mathbb{T})$, which is the orthogonal of the image of H^2 by the mapping $f \to f^*$, $f \in H^2$.

We define $\overline{H}_2$ to be the set of all $f(1/z)$ where f is in H^2. In what follows, we shall identify f and f^*, and reserve the notation x^* for the transpose conjugate of x. We shall sometimes write H_2 instead of H^2. In particular, H_2^p and $H_2^{n\times m}$ will be the set of vectors of size p, or matrices of size $n \times m$ with components in H^2. If M is a $n \times m$ matrix, we sometimes say $M \in H^2$ instead of $M \in H_2^{n\times m}$.

Note that if f is a rational function, $f = p/q$, p and q coprime, then f is in H^2 if and only if the roots of q lie outside $\overline{\mathbb{U}}$. Then f is also in H^∞.

If q is of degree n, then $\tilde{q}(z) = z^n \overline{q}(1/z)$. If the roots of q are in $\mathbb{U}$, then q is called *stable*. Hence $f = p/q$ is in H^2 if and only if $\tilde{q}$ is stable (if p and q are coprime). If $\deg(p) < \deg(q)$, then f is called *strictly proper*. If q is stable, then f is called *stable*. Hence a rational function f is in H_2^- if and only if f is strictly proper and stable.

Lemma 2
If f is analytic in Ω, q a polynomial with roots in Ω, there exists a unique polynomial r, $\deg(r) < \deg(q)$ such that $(f - r)/q$ is analytic in Ω. If $\Omega = \mathbb{U}$, if $f \in H^2$, then $(f - r)/q \in H^2$.

Writing $f = aq + r$ will be called the Euclidean division of f by q.

Proof. Unicity of r is obvious, since r/q can be analytic only if its has no poles, hence $r = 0$ because of the condition on degrees.

We shall prove existence by induction on the degree of q. We may hence assume $q = z - \alpha$. Take $r = f(\alpha)$. Define $g(z) = (f(z) - r)/q(z)$ for $z \neq \alpha$ and $g(\alpha) = f'(\alpha)$. This defines a function which is analytic in Ω, it will be denoted by $R_\alpha(f)$ later on.

Assume now $\alpha \in \mathbb{U}$, $f \in H^2$. We must show $g \in H^2$. Write $f_\alpha(z) = f(z) - f(\alpha)$. Let now r be a real number, and

$$M_r = \frac{1}{2\pi} \int_0^{2\pi} \left| \frac{f_\alpha(re^{i\theta})}{re^{i\theta} - \alpha} \right|^2 d\theta.$$

By definition, if this has a limit as $r \to 1$, then g is in H^2 and the limit is the square of the norm of g. Assume $r > |\alpha|$. The quantity $|rz - \alpha|$ is minimal on $\mathbb{T}$ for $z = \alpha/|\alpha|$, and the minimum is $r - |\alpha|$. Hence

$$M_r \leq \frac{1}{2\pi} \int_0^{2\pi} |f_\alpha(re^{i\theta})|^2 d\theta / (r - |\alpha|)^2.$$

The integral is an increasing function of r, so that $M_r \leq \|f_\alpha\|^2/(r - |\alpha|)^2$. Hence, the limit of M_r is bounded by $\|f_\alpha\|^2/(1 - |\alpha|)^2$. □

Note: assume $q = z - \alpha$, where $|\alpha| > 1$. If $f \in H^2$, then $(f - r)/q \in H^2$ for every constant r.

Lemma 3
Assume that q is a stable polynomial. There exists a constant C such that, for every $x \in H^2$, $\|x\| \leq C\|qx\|$.

Proof. Let $x \in H^2$, $\alpha \in \mathbb{C}$. Define $A = \langle zx \mid x\rangle / \|x\|^2$. Then

$$\|(z-\alpha)x\|^2 = \|x\|^2(1 + |\alpha|^2 - 2\Re\alpha A).$$

Taking $\alpha = 1/A$ shows $|A| \le 1$, hence $\Re\alpha A \le |\alpha|$. Thus

$$\|(z-\alpha)x\|^2 \ge \|x\|^2(1-|\alpha|)^2.$$

The proof is now obvious: take for $1/C$ the product of the leading coefficient of q and the $1-|\alpha_i|$, where α_i is a root of q. □

1.3 Form of Smith McMillan

We consider here a ring R and its quotient field K.

If M is a square matrix in R and has an inverse in R, then M is called *unimodular*. M is unimodular if and only if its determinant is an invertible element of R. If $R = A[z]$, where A is a field, then M is unimodular if and only if its determinant is a non-zero constant.

Theorem 1
Let R be a Euclidean ring, M a matrix with entries in R. There exist two unimodular matrices A and B in R, such that AMB has the form $\begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}$, where D is a $k \times k$ diagonal matrix, diagonal entries d_i of D are non-zero, and d_i divides d_{i+1}. Moreover, if $f_i = \prod_{j \le i} d_j$, for $i \le k$, and $f_i = 0$ for $i > k$, then f_i is the gcd of all minors of order i of M.

If K is the quotient field of R, M a matrix with entries in K, there are two unimodular matrices A and B such that AMB has the form above, with $d_i = a_i/b_i$, a_i and b_i are coprime. Moreover a_i divides a_{i+1} and b_{i+1} divides b_i. These quantities are unique up to an invertible factor in R.

Definition 1
If M is a matrix of polynomials, the quantities f_i are called the invariants of M. If M is a matrix of polynomials or rational functions, the matrix AMB is called the form of Smith of M. The maximum of the degrees of $\prod a_i$ and $\prod b_i$ is called the McMillan degree *of M.*

Proof. 1. Let E_{ij} be the matrix whose only non-zero element is one at position (i,j) and $F_{ij}(\lambda) = I + \lambda E_{ij}$. Let also G_{ij} be the matrix, which is like the identity matrix, but entries at location (i,i) and (j,j) are zero, while entries at locations (i,j) and (j,i) are one. It is obvious that if $i \ne j$, $F_{ij}(\lambda)$ and G_{ij} are unimodular matrices. Moreover, if D is a diagonal matrix, whose diagonal elements are invertible, then D is unimodular. The proof will show that every unimodular matrix is the product of matrices of the form D, $F_{ij}(\lambda)$ and G_{ij}. Note that, if A is a matrix, AG_{ij} is A with columns i and j swapped, while $AF_{ij}(\lambda)$ is A with column j replaced by its sum with λ times column i. Replacing A by AG_{ij} or $AF_{ij}(\lambda)$ is called an elementary operation.

2. Let a and b be elements of R. The extended gcd algorithm gives quantities u, v, x and y such that

$$\begin{pmatrix} u & v \\ x & y \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} c \\ 0 \end{pmatrix} \tag{1.2}$$

where c is the gcd of a and b. The matrix in this equation is unimodular (its determinant is 1), and is the product of elementary operations. In the same fashion, there exists elementary operations that replace elements of column j, rows i_1 and i_2 by the gcd and 0, and only rows i_1 and i_2 are modified. Operating on columns instead of rows, we can replace two elements of a row i, columns j_1 and j_2 by 0 and the gcd, modifying only columns j_1 and j_2.

3. If the matrix M is identically zero, there is nothing to do. Otherwise, swapping columns, we may assume that the first column is not identically zero. Swapping rows, we may assume that $M_{i1} = 0$ if $i > k$

and $M_{i1} \neq 0$ for $i \leq k$. Consider operations (1.2) with $a = M_{11}$ and $b = M_{i1}$, $i = k, k-1, \ldots, 2$. After that, we have $M_{i1} = 0$ for each i. We may do the same for the first row. However, this will modify the first column. Apply these transformation as many times as needed, alternatively to the first row, and to the first column. Let x_k be the value of M_{11} after the k-th transformation. If the transformation is not trivial (i.e. if there is a non-zero element on the row or the column, other than element at $(1, 1)$), then x_{k+1} is a strict divisor of x_k. Since x_1 has only a finite number of divisors (up to an invertible factor), this show that the process must end. We get

$$M' = \begin{pmatrix} M_{11} & 0 \\ 0 & M'' \end{pmatrix}$$

and conclude by induction that there are matrices A and B such that

$$AMB = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} \tag{1.3}$$

where D is diagonal, with non zero entries on the diagonal.

4. A consequence of (1.2) is

$$\begin{pmatrix} u & v \\ x & y \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \begin{pmatrix} 1 & -vb/c \\ 1 & ua/c \end{pmatrix} = \begin{pmatrix} c & 0 \\ 0 & d \end{pmatrix} \tag{1.4}$$

where d is the lcm of a and b. Note that the third matrix in this equation is the product of elementary matrices, namely $F_{21}(1)F_{12}(-vb/c)$. This means that there are elementary operations that replace d_i and d_j by their gcd and lcm. If we apply these operations for each i and j, we obtain the relation: d_i divides d_{i+1}.

5. Assume now M unimodular. There exist matrices A and B, that are product of elementary operations such that (1.3) holds. Note that the right hand side is unimodular, so that $AMB = D$, $M = A^{-1}DB^{-1}$. This means that any unimodular matrix is the product of elementary matrices, and unimodular diagonal matrices.

6. We show now unicity of the quantities d_i. The previous remark tells us that it suffices to show that MA and M have the same invariants if A is of the form $F_{ij}(\lambda)$, or G_{ij}, or diagonal unimodular. This is obvious for the last two cases. Let's show that $MF_{ij}(\lambda)$ and M have the same invariants. Consider a minor of size k. Let U be the minor in M, V the same in $MF_{ij}(\lambda)$. Let W be the minor obtained by replacing column i with column j (this is zero in case column i does not appear in U). Then $V = U + \lambda W$. If f_k is the k-th invariant of M, it divides U and W, hence V. Thus, if f'_k is the k-th invariant of $M' = MF_{ij}(\lambda)$, f_k divides f'_k. The converse is true, because $M = M'F_{ij}(-\lambda)$.

7. Finally, if M is a matrix over K, there exists u in R such that uM has entries in R. Hence, there are unimodular matrices A and B such that

$$AMB = \begin{pmatrix} D' & 0 \\ 0 & 0 \end{pmatrix}, \qquad D' = D/u.$$

We have $d'_i = d_i/u = a_i/b_i$. If we take a_i and b_i coprime, then the relations "d_i divides d_{i+1}" imply a_i divides a_{i+1} and b_{i+1} divides b_i. These quantities are unique, because, if we have another such form with a'_i/b'_i on the diagonal, and multiply everything by $\prod b_i \prod b'_i$, we get, by unicity in R,

$$\frac{a_i}{b_i \prod b_j \prod b'_j} = \frac{a'_i}{b'_i \prod b_j \prod b'_j}$$

hence $a_i/a'_i = b_i/b'_i$. $\square$

1.4 Inner matrices

If A is a square constant matrix, $A^*A = I$, then A is called *unitary*. If A is real, it is called orthogonal. Unitary matrices satisfy

$$\langle Ax \mid Ay \rangle = \langle x \mid y \rangle.$$

If A is a matrix of functions with entries in H^∞, x and y are vectors with entries in H^2, the previous relation is true (using the scalar product in H^2) if $A(z)$ is unitary for almost all z in $\mathbb{T}$.

Definition 2
A matrix $M(z)$ is called inner if it is square, its entries are in H^∞, and the inverse of $M(z)$ is $M(z)^*$ almost everywhere on $\mathbb{T}$.

The following obvious theorem will be used everywhere.

Theorem 2
Let A and B be inner matrices. The product AB is inner. The quotients AB^{-1} and $B^{-1}A$ are inner provided they are in H^∞. The transpose and the conjugate of A are inner.

Let $z = re^{i\theta}$ and $\alpha = se^{i\phi}$ be two complex numbers. Assume $|\alpha| < 1$, i.e. $s < 1$. Let $x = (z-\alpha)/(1-z\overline{\alpha})$. We have

$$\frac{1}{|x|^2} - 1 = \frac{(1-r^2)(1-s^2)}{r^2 - 2rs\cos(\theta-\phi) + s^2}. \tag{1.5}$$

Thus $z \in \mathbb{T}$ is equivalent to $x \in \mathbb{T}$, and $z \in \mathbb{U}$ is equivalent to $x \in \mathbb{U}$. Hence, if c is a complex number, with modulus one, α_i are elements of $\mathbb{U}$, the quantity

$$c \prod \frac{z - \alpha_i}{1 - z\overline{\alpha}_i}$$

is inner. This is called a Blaschke product. See [17, Theorem 15.21] for how to construct infinite Blaschke products. According to [17, Theorem 17.15], any inner function is the product of a Blaschke product and a factor of the form

$$\exp\left\{-\int_0^{2\pi} \frac{e^{it}+z}{e^{it}-z} d\mu(t)\right\}$$

where μ is a finite positive measure which is singular with respect to the Lebesgue measure. For instance $\exp((z+1)/(z-1))$ is inner. At the end of the second chapter, we shall consider functions that are analytic in the half plane (continuous time systems). The typical example of a non-rational inner function is then the function $\exp(s)$.

Consider a Blaschke product as above. We can write it as $cq/\tilde{q}$, where q is defined by its roots and its leading coefficient. If we change the leading coefficient, we get another value for c. In general, we shall assume that q is monic, but in some other cases, we shall assume $q(1) = 1$ (recall that $q(1)$ cannot be zero). Hence, a Blaschke product will be a product of terms of the form

$$\beta_\omega(z) = \frac{(z-\omega)(1-\overline{\omega})}{(1-z\overline{\omega})(1-\omega)}. \tag{1.6}$$

The notation β_ω will be used constantly in this paper.

Theorem 3
A rational function f is inner if and only if it is a Blaschke product.

Proof. Let f be an inner rational function. Since f is rational, there exists a polynomial p, with zeroes in $\mathbb{U}$, such that $f = pg$, g is analytic in $\mathbb{U}$, and does not vanish in $\mathbb{U}$. Let $h = \tilde{p}g$, so that $f = hp/\tilde{p}$. Note that $p/\tilde{p}$ is a Blaschke product, so that h is inner. We have to show that h is constant.

Since h is inner, we have

$$h(1/z) = 1/\overline{h}(z) \tag{$*$}$$

for every $z \in \mathbb{T}$. But since h is rational, this is true whenever both terms are defined. Since h does not vanish on $\mathbb{U}$, $h(1/z)$ is defined on $\mathbb{U}$. Since h has no poles, even at infinity, it must be constant. □

Note: if h is inner, the maximum modulus principle says that h is bounded by 1 in $\mathbb{U}$. Now $1/h$ is analytic in $\mathbb{U}$. If it is bounded (for instance if h is rational), then h must be constant. In the case of $h = \exp((z+1)/(z-1))$, it happens that h is not zero on $\mathbb{U}$, but the minimum of $|h|$ is zero.

Theorem 4
A rational matrix A is inner if and only if there exists a polynomial matrix D, whose entries are of degree at most n, a stable polynomial q of degree n, an element c in $\mathbb{T}$, such that, if $\tilde{q} = z^n \overline{q}(1/z)$ and $\tilde{D} = z^n \overline{D}^t(1/z)$, then

$$A = \frac{D}{\tilde{q}}, \qquad A^{-1} = \frac{\tilde{D}}{q}, \qquad \det A = c\frac{q}{\tilde{q}}. \tag{1.7}$$

The integer n is the McMillan degree of A.

Proof. By definition, A is inner if and only if

$$A^{-1}(z) = \overline{A}(1/z)^t \tag{$**$}$$

for almost every $z \in \mathbb{T}$. Since A is rational, this equation is true for every z for which both members are defined. Since A is in H^∞, it is defined on $\mathbb{T}$, so that $(**)$ is true for each $z \in \mathbb{T}$.

Let d be the determinant of A. We have

$$\frac{1}{d(z)} = \overline{d}(\frac{1}{z}),$$

so that d is inner, and $d = cq/\tilde{q}$ for some stable polynomial q of degree n, and some constant c. Let C be the matrix of cofactors of A, so that $A^{-1} = C/d$. We have

$$cqA^{-1} = C\tilde{q}.$$

The rational function $E = qA^{-1}$ is defined for all $|z| \leq 1$, being the right hand side of this equation. But the left-hand side is defined for $|z| \geq 1$ according to $(**)$. Thus E is a polynomial. Its degree is at most n: E/z^n has a limit as $z \to \infty$, this limit being $A^{-1}(\infty) = \overline{A}(0)^t$ (modulo the leading coefficient of q). This means that there exists a polynomial matrix D such that $E = \tilde{D}$. The relation $A = D/\tilde{q}$ is now obvious from $(**)$.

We have now to show that n is the McMillan degree of A. Consider the form of Smith of A. If the diagonal elements are a_i/b_i, we have

$$c\frac{q}{\tilde{q}} = \frac{\prod a_i}{\prod b_i}.$$

We pretend that $\prod a_i$ and $\prod b_i$ are coprime, so that the max of their degrees is the max of the degrees of q and $\tilde{q}$, which is n.

If $\prod a_i$ and $\prod b_i$ were not coprime, there would exist a complex number α which is a root of a_i and b_j, with $i \neq j$. For this α neither $A(z)$ nor $A(1/z)$ is defined. But equation $(**)$ says that, for any α, $A(z)$ or $A(1/z)$ is defined. □

Note: the McMillan degree n is uniquely defined by the determinant of Q. Thus, the McMillan degree of the product of two inner matrices is the sum of the degrees.

Corollary 1
$\tilde{q}^{i-1}$ divides every minor of size i of D.

Proof. This is because the i-th invariant of D is

$$\tilde{q}^i \prod_{j\leq i}(a_j/b_j) = c\tilde{q}^{i-1} \prod_{j\leq i} a_j \prod_{j>i} b_j.$$

□

The corollary will be used with $i = 2$: $\tilde{q}$ divides any minor of order 2.

In the case $i = 1$, we get the following: the first invariant is $f_1 = a_1\tilde{q}/b_1$. If A has positive McMillan degree, then $b_1 \neq 1$ (since b_i divides b_1). Hence $\tilde{q}$ does not divide f_1. In other terms, $\tilde{q}$ does not divide every entry of D. It can happen that $D(z) = 0$ whenever $\tilde{q}(z) = 0$. A typical case is: take $b = (z - a)/(1 - \overline{a}z)$, and $A = bI_n$. Then $q = (z - a)^n$, $D = (z - a)(1 - z\overline{a})^{n-1}I$.

It is well known that a polynomial can be written as the product of polynomials of degree one. We shall show that the same holds for inner matrices.

Consider

$$Q_0 = I - (1 - \beta_\omega)uu^*, \tag{1.8.a}$$

$$Q_1 = [I - (1 - \beta_\omega)uu^*][I - (1 - \beta_{\overline{\omega}})u'u'^*]. \tag{1.8.b}$$

In the case where u has components $(1, 0, \dots, 0)$, then Q_0 has the form

$$Q_\omega = \operatorname{diag}(\beta_\omega, 1, \dots, 1)$$

and if u is of unit norm, there exists a unitary matrix P such that $Q_0 = PQ_\omega P^{-1}$.

Lemma 4
If $\omega \in \mathbb{U}$, and $\|u\| = 1$, then Q_0 is inner, of McMillan degree one. Its determinant is β_ω, its value at $z = 1$ is the identity matrix. If $\|u'\| = 1$, then Q_1 is inner of McMillan degree two.

Assume that ω is not real. Let Q_0' be the second factor of Q_1. Then Q_1 is real if and only if u' is proportional to $Q_0(\overline{\omega})^*\overline{u}$, this condition is equivalent to u proportional to $Q_0'(\omega)\overline{u}'$. It is also

$$u' = \lambda(1 + \frac{\omega - \overline{\omega}}{1 - |\omega|^2} - uu^*)\overline{u}. \tag{1.8.c}$$

Assume Q inner. Then $Q_0^{-1}Q$ is inner if and only if $u^*Q(\omega) = 0$. If Q is real, then $Q_1^{-1}Q$ is inner if and only if this condition is true, together with condition (1.8.c).

Proof. The first claim is obvious. We have

$$Q_0^{-1} = I - (1 - 1/\beta_\omega)uu^*.$$

Hence $Q_0^{-1}Q$ is inner if and only if it has no singularity at $z = \omega$, which is $u^*Q(\omega) = 0$. The result in the real case will be shown in chapter 3 (theorem 26). □

Corollary 2
Assume that Q is inner, $\det Q = cq/\tilde{q}$, *and $q(\omega) = 0$. There exists an inner matrix Q_0 of the form* (1.8.a) *such that $Q_0^{-1}Q$ is inner. If Q is real, and ω is real, we can assume Q_0 real. If Q is real, and ω is not real, there exists a real matrix Q_1 of the form* (1.8.b) *such that $Q_1^{-1}Q$ is inner.*

Proof. The assumption is that $\det Q$ vanishes at $z = \omega$, and $\omega \in \mathbb{U}$. Hence, there exists a non-zero vector u such that $u^*Q(\omega) = 0$. We may assume $\|u\| = 1$. If Q and ω are real, we can also assume that u is real. □

Theorem 5 (Potapov)
Assume that Q *is an inner rational matrix,* $\det Q = cq/\tilde{q}$ *and* $q = (z-\omega_1)(z-\omega_2)\cdots(z-\omega_n)$*. We can write*

$$Q = Q_1 Q_2 \cdots Q_n A \tag{1.9}$$

where each Q_i *has the form* (1.8.a)*, with determinant* β_{ω_i}*:*

$$Q_i = I - (1-\beta_{\omega_i})u_i u_i^*. \tag{1.10}$$

If Q *is real, and* $q = q_1 q_2 \dots q_n$*, where each* q_i *is irreducible, then* Q_i *has the form* (1.8.a) *in case* q_i *is of degree one, and the form* (1.8.b) *if* q_i *is of degree two. In any case, the matrix* A *is constant and unitary.*

Proof. Denote by $q(Q)$ the polynomial q such that $\det Q = cq/\tilde{q}$. Then $q(Q_1Q_2) = q(Q_1)q(Q_2)$. We proceed by induction on the number of factors of q, hence obtain (1.9) for some matrix A, which is inner, and has constant determinant. Thus A must be constant. Note that $Q_i(1)$ is the identity matrix, so that $A = Q(1)$. $\square$

Note: We shall give later (theorem 26) a more elaborate version of this theorem. If we impose the condition $Q(1) = I$, then Q lies in a manifold of dimensuion np, where n is the degree and p the size of Q. The decomposition above has exactly np free parameters, but we cannot use it to parameterise the manifold. In fact, if q has n distint roots, then u_i is uniquely defined (modulo a phase factor), once an ordering for the roots is given. The next corollary implies that the span of the vectors u_i depends only on Q. We can also restate the theorem as: any inner matrix the product of some matrices Q_ω and some unitary matrices, but in this decomposition, there are much more free parameters.

Corollary 3
If $\omega \in \mathbb{U}$*, and* Q *is inner, rational, then* $\|Q(\omega)^*u\| \le \|u\|$*. We have equality if and only if* u *is orthogonal to every* u_i*, for a factorisation like* (1.9), (1.10).

Proof. The proof is by induction on the McMillan degree of Q. There is nothing to show if Q is constant. Assume $Q = Q_1 B$. Let $\lambda = 1 - \overline{\beta_{\omega_1}(\omega)}$ and $w = Q_1(\omega)^*u = (I - \lambda u_1 u_1^*)u$. We have

$$\|w\|^2 = \|u\|^2 - (1 - |\beta_{\omega_1}(\omega)|^2)|u_1^*u|^2$$

so that $\|w\| \le \|u\|$, and $\|w\| = \|u\|$ if and only if $u_1^*u = 0$, case where $w = u$ (recall that $|\beta(\omega)| < 1$).

By induction,

$$\|Q(\omega)^*u\| = \|B(\omega)^*w\| \le \|w\| \le \|u\|.$$

If $\|Q(\omega)^*u\| = \|u\|$, then $u = w$, and $\|B(\omega)^*u\| = \|u\|$, hence by induction, u is orthogonal to every u_i. $\square$

1.5 Shift invariant spaces

Assume that we have a discrete time system, which is defined by a time-invariant input-output relation. Given a sequence $(u_i)_i$, which is the input, we associate a series $U = \sum u_i z^{-i}$. Given an input U, the system defines an output Y. The origin of time is $i = 0$, the past is defined by $i < 0$, and the future by $i > 0$. With our conventions, a polynomial U defines a sequence for which only a finite number of values are not zero, none of them are in the future. The system is time-invariant if the output of z^iU is z^iY whenever the output of U is Y. The system is said causal in case y_i depends only on u_j for $j \le i$ (it is called strictly causal if y_i does not depend on u_i). For details, see [19].

Under these assumptions, it is possible to compute the output associated to a given input, provided that we can compute the output associated to an input with $u_i = 0$ for $i > 0$. We shall say that the system is stable in case $\sum_{i\le 0}\|u_i\|^2$ gives an output where $\sum_{i<j}\|y_i\|^2$ is finite, whatever j. For instance,

if $\mathcal{H}$ is an element of H_2^- or $\overline{H_2}$, the system defined by $Y = \mathcal{H}U$, is time-invariant, stable and causal (or strictly causal).

We can identify two sequences u_i and u'_i (defined for $i < 0$) if, whatever v_i ($i \geq 0$), the outputs Y and Y' corresponding to the concatenations of u and v, or of u' and v, are identical for $i \geq 0$. The state of the system is then defined by the set of all past inputs, modulo this identification. If the system is stable, we identify U with zero provided that Y is in H^2.

If the system is not stable, we shall assume that, at some time t_0 in the past, the state is zero, and we consider only inputs u_i such that $u_i = 0$ if $i < t_0$. We assume that this implies $y_i = 0$ for $i < 0$. Such a system can be defined by $Y = \mathcal{H}U$, where $\mathcal{H}$ is proper (or strictly proper), this gives a causal (or stricly causal) time-invariant system. Here the state is defined by all past inputs (here polynomials U), modulo identification: U is identified to zero if it gives an output Y which is a polynomial.

In any case, the state space $\mathcal{X}$ is the quotient of past inputs, modulo a space $\mathcal{V}$. If the system is stable, both methods fo defined the state space give the same result. This property will be used in order to show that Q is rational in the factorisation $\mathcal{H} = Q^{-1}C$. Since the system is time-invariant, the set $\mathcal{V}$ is shift invariant.

In what follows, a *shift invariant* vector space will be a set E of functions, such that, if $f \in E$, then the function $z \to zf(z)$ is in E. Since E is a vector space, it implies that $pf \in E$, whenever p is a polynomial and f an element of E.

The easy case is when E is a set of polynomials over a field K. Then E is an ideal of $K[z]$, hence is of the form $rK[z]$, for some polynomial r. In particular, if p and q are two polynomials, E is the set of all $up + vq$, where u and v are polynomials, then the polynomial r is the gcd of p and q. The polynomials p and q are coprime if $E = K[z]$.

We shall consider two extensions of this easy case: the case where E is a set of vectors of polynomials, and the case where E is a set of analytic functions. In this last case, we have to make some assumptions. For instance, if $1 \in E$, then every polynomial is in E. However, there are many shift invariant spaces that satisfy this condition. We shall restrict our attention to functions analytic in $\mathbb{U}$, in fact, E will be a subset of H^2. Since H^2 is a metric space, we may add the condition that E is closed. Now, the theorem of Beurling says that $E = rH^2$ for some inner function r. An extension of this theorem in the non-scalar case will be given here.

Assume that E is formed of vectors of size p. Then E is called *of full rank* if there are p elements $x_1, x_2, \ldots, x_p$ such that for some evaluation point z_0, the vectors $x_i(z_0)$ are linearly independent.

Theorem 6
Let E be a subset of $\mathbb{C}^p[z]$, which is shift invariant, of full rank. There exists a matrix $Q \in \mathbb{C}^{p\times p}[z]$, with non-zero determinant, unique up to a unimodular factor, such that E is the set of all Qx, where $x \in \mathbb{C}^p[z]$.

Theorem 7 (Beurling-Lax)
Let E be a subset of H_2^p, which is shift invariant, closed, of full rank. There exists an inner matrix Q, of size p, unique up to a constant unitary factor, such that E is the set of all Qx, where $x \in H_2^p$.

Let's first consider unicity. Assume that we have two matrices Q and Q' that satisfy the conditions. Hence, we get two matrices R and S such that $Q' = QR$ and $Q = Q'S$. Since Q and Q' have non-zero determinant, the matrix S is the inverse of R. In the polynomial case, it follows that R is polynomial, with polynomial inverse, hence is unimodular. In the H^2 case, S is in H^2, with inverse in H^2. Since $Q = Q'S$, and Q and Q' are inner, S is inner. We pretend that S is constant.

In the scalar case, the proof of [17, Th 17.21] is the following. Let $h = s + 1/s$. Now h is in H^2, and its imaginary part is zero on $\mathbb{T}$ (on $\mathbb{T}$, we have $s\overline{s} = 1$, hence $h = 2\Re s$). But h is the Poisson integral of its value on $\mathbb{T}$, hence h is real, thus h is constant. This implies that s is constant (take square roots). In the matrix case, we cannot take square roots. We know for instance that $S^t + S^{-1}$ is constant, and $d + 1/d$ is constant, where d is the determinant of S. But this is not enough: for instance $S = \begin{pmatrix} 1+\cos z & \sin z - 1 \\ 1 + \sin z & 1 - \cos z \end{pmatrix}$ satisfies $\det S = 1$ and $S^t + S^{-1}$ constant.

In the general case, we apply the following result.

Lemma 5
Assume that S is an inner $p\times p$ matrix, and S^{-1} has entries in H^2. Then S is a constant unitary matrix.

Proof. Let x and y be constant vectors, $t = S^{-1}y$. This is some element of H^2. We have

$$\langle z^n Sx \,|\, y\rangle = \langle z^n Sx \,|\, St\rangle = \langle z^n x \,|\, t\rangle.$$

Since y is constant, it is orthogonal to $z^n Sx$ for $n > 0$, so that t is orthogonal to $z^n x$ for $n > 0$. Since x is arbitrary, it implies that t is constant. Hence S is constant. □

Assume that we have $E = Q\mathbb{C}^k[x]$ or $E = QH_2^k$. We know that there are p elements x_i in E, of the form $x_i = Qy_i$ such that, evaluated at z_0, this gives p linearly independent vectors. This implies $k \geq p$.

Let's show the existence in the polynomial case. Define E_i to be the set of all elements of E for which components of index $< i$ are zero. For $0 < i \leq n$, we consider an element p_i in E_i for which component i has smallest degree. It may happen that all elements p in E_i have 0 as i-the component, said otherwise $E_i = E_{i+1}$. In this case, we chose $p_i = 0$.

Let x be an element of E. We consider some polynomials y_i, such that $x_i = x - \sum_{j\leq i} y_j p_j$ is in E_i. These numbers are found by induction on i. We define y_i to be zero in case the i-th component of x_i is zero. Otherwise, y_i is the remainder of the i-the component of x_i by the i-th component of p_i. This shows that any x is a linear combinations (with coefficients in $K[z]$) of the p_i. If Q is the matrix whose columns are the non-zero elements p_i, we have $x = Qy$ for some y. Clearly, the rank of Q is the number of columns, and the remark above says that it must br p, so that Q is square and has non-zero determinant.

Consider now the H^2 case. Let zE be the set of all zx, $x \in E$. Let L be its orthogonal complement. It is easy to show that zE is closed (recall that E is closed in H^2). This implies that zE is the orthogonal of L. In other words

$$\forall x \in E, \quad \exists t \in E, \exists y \in L, \qquad x = y + zt. \tag{1.11}$$

Consider n elements x_i in L, of unit norm, which are orthogonal. Then

$$\langle x_i z^j \,|\, x_k z^l\rangle = \delta_{ik}\delta_{jl}. \tag{1.12}$$

This is because the scalar product is $\langle x_i z^{j-l} \,|\, x_k\rangle$ or $\langle x_i \,|\, x_k z^{l-j}\rangle$, depending on the sign of $j - l$. Since x_i and x_k are in L, the product is zero if $j \neq l$; it is $\langle x_i \,|\, x_k\rangle$ otherwise. Let now $v_{ij}(z) = \sum_k \overline{x_{ik}(z)} x_{jk}(z)$. This is in $L^1(\mathbb{T})$, and (1.12) says

$$\frac{1}{2\pi}\int_0^{2\pi} e^{in\theta} v_{jk}(e^{i\theta})d\theta = 0$$

for each integer $n \in \mathbb{Z}$ (if $n = 0$, the relation is true only if $j \neq k$, if $j = k$, the integral is one). Thus, v_{ij} is constant, it is δ_{ij}. This means that the set of vectors $x_i(z)$ is orthonormal in $\mathbb{C}^p$, for almost every $z \in \mathbb{T}$. In particular, L has finite dimension, at most p.

Let $(\Phi_\alpha)_\alpha$ be an orthonormal basis of L. Let Q be the matrix whose columns are the Φ_α. Then $Q^*Q(z)$ is the identity matrix if $z \in \mathbb{T}$. Equation (1.12) implies that $\sum_{i,\alpha} \lambda_{i,\alpha} z^i \Phi_\alpha$ is in H^2 provided that $\sum |\lambda_{i,\alpha}|^2 < \infty$. Since E is closed, this expression is in E, and $Qx \in E$ whenever x has components in H^2. On the other hand, let $u \in E$, $\lambda_{i,\alpha} = \langle z^i \Phi_\alpha \,|\, u\rangle$, and $x_0 = u - \sum_{i,\alpha} \lambda_{i,\alpha} z^i \Phi_\alpha$. This element is in H^2. It is in E, and orthogonal to each $z^i\Phi_\alpha$. Write $x_0 = z^i x$, with i as large as possible. Then x is orthogonal to each Φ_α. Write $x = y + zt$ like in (1.11). Then $y = 0$, because $(\Phi_\alpha)_\alpha$ is a basis of L. This contradicts the fact that i is maximal. It implies $x = 0$, and $u = Qy$ for some y.

The same argument as above shows that Q must have at least p columns. The relation $Q^*Q(z) = I$ on $\mathbb{T}$ says now that Q is inner.

1.6 The state space

Let H be a $m \times p$ matrix. We shall consider the following sets

$$\mathcal{V}(H) = \{x \in \mathbb{C}^p[z], Hx \in \mathbb{C}^m[z]\}$$

$$\mathcal{V}'(H) = \{x \in H_2^p, Hx \in H_2^m\}.$$

Lemma 6
Assume H rational. Then $\mathcal{V}(H)$ is shift invariant, of full rank. Assume moreover $H \in \overline{H_2}$. Then $\mathcal{V}'(H)$ is shift invariant, of full rank, and closed.

Proof. We can always assume $H = P/q$, where P is a matrix of polynomials, and q a polynomial. If q is of minimal degree, then H is defined if and only if $1/q$ is defined, so that, if $H \in \overline{H_2}$, we may assume that the roots of q are in $\mathbb{U}$.

If e_i is a basis of $\mathbb{C}^p$, then qe_i is a set of p vectors in $\mathcal{V}$ and $\mathcal{V}'$ which are linearly independent, whenever evaluated at z_0 which is not a zero of q. Thus $\mathcal{V}$ and $\mathcal{V}'$ are shift invariant, of full rank.

Let's show that $\mathcal{V}'$ is closed. Consider a Cauchy sequence x_i. Let $y_i = Hx_i$. Then $qy_i = Px_i$. Obviously, x_i converges in H^2 to some x, and Px_i converges to Px. Lemma 3 says that the sequence y_i is Cauchy, hence converges to some y. Obviously $qy = Px$. Thus $y = Hx$ is in H^2 and x is in $\mathcal{V}'(H)$. □

Theorem 8
Let $\mathcal{H}$ be a $m \times p$ matrix of rational functions. There exists a $p \times p$ polynomial matrix D, a $m \times p$ polynomial matrix N, two polynomial matrices X and Y such that

$$\mathcal{H} = ND^{-1}, \qquad XN + YD = I. \tag{1.13}$$

If $\mathcal{H} = N_1 D_1^{-1}$, then $D_1 = DU$ for some polynomial matrix U. If moreover $X_1N_1 + Y_1D_1 = I$ then U is unimodular.

Assume moreover that $\mathcal{H} \in \overline{H_2}$. There exists an inner matrix Q, a matrix $C \in H^\infty$, two matrices X' and Y' in H^2 such that

$$\mathcal{H} = CQ^{-1}, \qquad X'C + Y'Q = I. \tag{1.14}$$

If $\mathcal{H} = C_1Q_1^{-1}$, then $Q_1 = QU$ for some matrix U in H^2. If moreover $X_1'C_1 + Y_1'Q_1 = I$, then U is constant.

The matrix Q is rational. If its determinant is $q/\tilde{q}$, then q is the determinant of D. If $\mathcal{H}$ is real, all matrices can be chosen real.

In the same fashion, we have a factorisation $\mathcal{H} = D^{-1}N$ and $\mathcal{H} = Q^{-1}C$.
Proof. 1. We know that there exists a matrix D such that

$$x \in \mathcal{V}(\mathcal{H}) \Longleftrightarrow D^{-1}x \text{ is a polynomial.} \tag{1.15}$$

From this, we deduce first that $N = \mathcal{H}D$ is a polynomial. Let E be the set of all $N^tx + D^ty$, where x and y are polynomials. This space is shift invariant, of full rank, so that this is the set of all R^tu for some polynomial matrix R^t; there are matrices X_1, Y_1, D_1 and N_1 such that

$$R^t = N^tX_1^t + D^tY_1^t, \quad D^t = R^tD_1^t, \quad N^t = R^tN_1^t.$$

If we transpose, we get

$$R = X_1N + Y_1D, \quad D = D_1R, \quad N = N_1R.$$

Note that $(I - X_1N_1 - Y_1D_1)R = 0$. Since R is of full rank, we get

$$X_1N_1 + Y_1D_1 = I.$$

Since $\mathcal{H} = ND^{-1} = N_1D_1^{-1}$, $D_1x \in \mathcal{V}$ whenever x is a polynomial. Using (1.15) a second time, we see that $D^{-1}D_1$ is a matrix of polynomials. Since it is R^{-1}, R is unimodular. So

$$I = R^{-1}X_1N + R^{-1}Y_1D,$$

and we get (1.13) with $X = R^{-1}X_1$, $Y = R^{-1}Y_1$.

2. Suppose now $\mathcal{H} = N_1D_1^{-1}$. Then

$$XN_1 + YD_1 = D^{-1}D_1.$$

This implies that $U = D^{-1}D_1$ is a matrix of polynomials. If moreover $X_1N_1 + Y_1D_1 = I$, its inverse is polynomial, hence unimodular. Note that, if $\mathcal{H}$ is real, we may chose D real. It follows that N is real, and $XN + YD = I$ implies $\Re(X)N + \Re(Y)D = I$.

3. The second claim can be proved in the same fashion. We know that $\mathcal{V}'(\mathcal{H})$ is closed, so that there exists Q such that $\mathcal{V}'(\mathcal{H}) = QH^2$. Let $C = \mathcal{H}Q$. This is in H^2, but also in L^∞, since $\mathcal{H}$ and Q are in L^∞. Hence $C \in H^\infty$. We can consider the space of all $C^tx + Q^ty$, where x and y are in H^2. This is a full rank, shift invariant closed subspace of H^2.

Using lemma 5, we see that the matrices R and U are constant, wherever they were unimodular in the previous point. This shows the second claim of the theorem. We shall see in the next point that Q is rational. This implies that $Q(1)$ is defined and invertible, so that we can multiply Q by $Q(1)^{-1}$, hence assume $Q(1) = I$. If $\mathcal{H}$ is real, we have $\mathcal{H} = CQ^{-1} = \overline{C}\overline{Q^{-1}}$, so that $Q^{-1}\overline{Q}$ is constant. Since this is the identity matrix for $z = 1$, it follows that Q is real, and C is real also. As above, we can then assume that X' and Y' are real.

4. A consequence of the unicity relations is that $Q^{-1}D$ is in H^2, with inverse in H^2. Let e_k be a basis of $\mathbb{C}^p$. We have

$$\langle e_kz^i \mid Q^{-1}De_j\rangle = \langle Qe_kz^i \mid De_j\rangle.$$

In the case where i is greater than the degree of D, the last scalar product is zero. Hence $Q^{-1}D$ is a polynomial, and Q is rational.

We know that the determinant of Q has the form $q/\tilde{q}$. Let r be the determinant of D. Since $Q^{-1}D$ is in H^2, with inverse in H^2, the same is true for the determinant $r\tilde{q}/q$. This means that q divides r, and the stable part of r divides q. Since $X\mathcal{H} + Y = D^{-1}$, all zeroes of r are in $\mathbb{U}$, thus r/q is constant. □

Let's introduce the spaces $\mathcal{X}(H)$, the quotient of the $\mathbb{C}^p[z]$ by $\mathcal{V}(H)$, and $\mathcal{X}'(H)$, the quotient of H_2^p by $\mathcal{V}'(H)$.

Lemma 7
Let A be a rational matrix. Assume that $A = UBV$, where U and V are unimodular, $B = \begin{pmatrix} C & 0 \\ 0 & 0 \end{pmatrix}$. Then $\mathcal{X}(A) = \mathcal{X}(C)$. If C is diagonal, with terms c_i on the diagonal, then $\mathcal{X}(A)$ is the direct sum of $\mathcal{X}(c_i)$. In the same fashion, $\mathcal{X}'(A)$ is the sum of $\mathcal{X}'(c_i)$.

If $c = a/b$, a and b coprime, then $\mathcal{X}(c)$ is isomorphic to the set of polynomials of degree $< \deg(b)$. The same is true of $\mathcal{X}'(c)$, provided that b is stable.

Proof. It is obvious that $\mathcal{X}(B) = \mathcal{X}(C) = \sum_i \mathcal{X}(c_i)$, and $\mathcal{X}'(B) = \mathcal{X}'(C) = \sum_i \mathcal{X}'(c_i)$,

Note that $UBVx$ is a polynomial (resp. in H^2) if and only BVx is, because U and U^{-1} are polynomial matrices (hence are in H^∞). On the other hand, the multiplication by V leaves the set of polynomial vectors invariant, it leaves also H^2 invariant. Hence, we can replace A by B.

Finally, any x can be written as $x = bq + r$, with $\deg(r) < \deg(b)$. If x is a polynomial, then q is a polynomial, if x is in H^2, then q is in H^2. Now $xa/b = aq + ar/b$. This is a polynomial if and only if b divides ar. Since b is coprime to a, it is a polynomial if and only if $r = 0$. In the same fashion, ar/b is in H^2 if and only if $r = 0$, since b is stable. □

Theorem 9
Under the assumptions of the previous theorem, we have

$$\mathcal{V}'(\mathcal{H}) = \mathcal{V}'(Q^{-1}) = \mathcal{V}'(D^{-1}) \qquad \mathcal{V}(\mathcal{H}) = \mathcal{V}(D^{-1}), \qquad \mathcal{X}(\mathcal{H}) = \mathcal{X}'(\mathcal{H}).$$

Proof. The relations $\mathcal{V}'(\mathcal{H}) = \mathcal{V}'(Q^{-1})$ and $\mathcal{V}(\mathcal{H}) = \mathcal{V}(D^{-1})$ are obvious, equivalent to (1.13) and (1.14). The relation $\mathcal{V}'(\mathcal{H}) = \mathcal{V}'(D^{-1})$ is clear also: it is a consequence of (1.13) and the fact that any polynomial is in H^2.

The last claim is $\mathcal{X}(D^{-1}) = \mathcal{X}'(D^{-1})$. According to the lemma, we can replace D by its form of Smith. But the form of Smith is $D^{-1} = UCV$, $C = \text{diag}(1/c_i)$, and $\det D = \prod c_i$. Hence c_i is stable. $\square$

Assume M square and invertible. Let the form of Smith of M be $AMB = \text{diag}(a_i/b_i)$. Then $B^{-1}M^{-1}A^{-1} = \text{diag}(b_i/a_i)$. There are permutation matrices that replace this by $\text{diag}(b_{n-i}/a_{n-i})$. This is the form of Smith of M^{-1}. Hence M and M^{-1} have the same McMillan degree.

Lemma 8
If M is rational and proper, the McMillan degree of M is the dimension of $\mathcal{X}(M)$. Thus, if A is constant, M and $M + A$ have same McMillan degree.

Proof. Let q be a polynomial of degree n, such that entries of Mq are polynomials. By assumption, these entries are of degree $\leq n$. Let $f_1, f_2, \ldots, f_k$ be the invariant factors of Mq. Take a minor x of size $k \times k$ of Mq. Then x has degree $\leq kn$. Since f_k is the gcd of these quantities, the degree of f_k satisfies this condition. Hence f_k/q^k is proper. But f_k/q^k is $\prod a_i/b_i$, where a_i/b_i are the diagonal elements of the form of Smith of M. Thus $\deg \prod a_i \leq \deg \prod b_i$. This implies that the McMillan degree of M is $\deg \prod b_i$, which is the dimension of $\mathcal{X}(M)$. $\square$

Corollary 4
The matrices $\mathcal{H}$, D and Q have the same McMillan degree.

Lemma 9
If Q is inner rational, $Q = D/\tilde{q}$, $Q^{-1} = \tilde{D}/q$, then $Q^{-1}C$ is strictly proper, stable, rational of McMillan degree $\leq n$ and C is in H^∞ if and only if $\tilde{q}C$ and $\tilde{D}C$ are polynomials of degree $< n$.

Proof. Assume $Q^{-1}C$ strictly proper, stable, rational. Then $Q^{-1}C = N/s$, where N is a polynomial matrix, and s a stable polynomial. We have $C = DN/(\tilde{q}s)$. Since $C \in H^\infty$, s divides DN, $DN = Ts$, hence $q\tilde{q}N = \tilde{D}Ts$. Thus s divides $q\tilde{q}N$. But s is stable, $\tilde{q}$ is unstable, and we may assume s coprime to N (i.e. s coprime to the gcd of the entries of N). Hence s divides q, $q = ks$. Hence $\tilde{D}C = kN$, $\tilde{q}C = T$. Since $Q^{-1}C$ is strictly proper, kN and T are of degree less than n.

On the other hand, if $\tilde{D}C = N$, then $Q^{-1}C = N/q$, and $Q^{-1}C$ is rational, strictly proper and stable. If moreover $C = T/\tilde{q}$, then $C \in H^\infty$. Since $\mathcal{X}(CQ^{-1}) \subset \mathcal{X}(Q^{-1})$, CQ^{-1} (hence $Q^{-1}C$) cannot have greater McMillan degree than Q^{-1}. $\square$

1.7 Left shift

Assume that f is analytic in Ω, $\alpha \in \Omega$. We define $g = R_\alpha(f)$ by $g(z) = (f(z) - f(\alpha))/(z - \alpha)$ and $g(\alpha) = f'(\alpha)$. Then g is analytic in Ω. We also know that if $f \in H^2$, then $g \in H^2$. The case of interest will be the case $\alpha = 0$, because the orthogonal of QH^2 is R_0-invariant, and finite dimensional. The operator R_α is called the left shift, or backwards shift.

An easy remark is the following: $R_0^k(f) = (f(z) - P)/z^k$, where P is some polynomial. Thus, if $\sum \lambda_i R_0^i(f) = 0$, then $f = p/q$ where p and q are polynomials. Thus, if X is a finite dimensional R_0-invariant space, every element of X is a rational function. Moreover, it is obvious that there exists a polynomial q of degree $\leq n$, such that fq is a polynomial of degree $\leq n$, for every f in X. The objective of this section is to prove the more precise following result.

Theorem 10
Assume that X *is a set of functions, analytic in* Ω*, with range in* $\mathbb{C}^p$*. Assume that* X *is* R_α*-invariant, and is a vector space of finite dimension* n*. There exists an observable pair* (A, C)*,* C *of size* n*, such that*

$$X = A[I - (z-\alpha)C]^{-1}\mathbb{C}^n. \tag{1.16}$$

The space X *is* R_β*-invariant for every* $\beta \in \Omega$*, in fact, for every* β *such that* $I - (\beta - \alpha)C$ *is invertible. Let* $B = I + \alpha C$*,* P *an invertible constant matrix, and*

$$K(z, \omega) = A(B - zC)^{-1}P^{-1}(B^* - \overline{\omega}C^*)^{-1}A^*. \tag{1.17}$$

Then X *is the linear span of the vectors* $K(z, \omega_i)c_i$*,* $\omega_i \in \Omega_0$*,* $c_i \in \mathbb{C}^p$*, provided that the subset* Ω_0 *of* Ω *has an infinite number of elements.*

The condition (A, C) observable says that the space defined by (1.16) has dimension n. By definition, (A, C) is observable if it satisfies the conditions (1.18). In the next chapter, we shall give an alternate proof of the equivalence of these conditions.

Lemma 10
Assume that A *is a* $p \times n$ *matrix,* C *is a square* $n \times n$ *matrix. The following conditions are equivalent.*

$$\forall x, \quad \forall \lambda \in \mathbb{C}, \quad Ax = 0, Cx = \lambda x \textit{ implies } x = 0. \tag{1.18.a}$$

$$\forall x, \quad AC^i x = 0 \quad (0 \le i < n) \textit{ implies } x = 0. \tag{1.18.b}$$

Proof. Clearly, if $Ax = 0$ and $Cx = \lambda x$, then $AC^i x = \lambda^i Ax = 0$, so that (1.18.$b$) implies (1.18.$a$).

Define some numbers a_i by $P = \det(zI - C) = z^n - \sum a_i z^i$. The Cayley-Hamilton theorem says

$$C^n = a_0 + a_1 C + \cdots + a_{n-1}C^{n-1}. \tag{1.19}$$

Write $P = \prod_i (z - \lambda_i)^{\alpha_i}$, with $\lambda_i \neq \lambda_j$ for $i \neq j$. There are some polynomials u_i such that

$$\sum_i u_i(z) \prod_{j \neq i} (z - \lambda_j)^{\alpha_j} = 1.$$

Fix some vector x, and define $a_i = u_i(C)\prod_{j\neq i}(C - \lambda_j)^{\alpha_j}x$. Then $x = \sum a_i$, and $(C - \lambda_i)^{\alpha_i}a_i = 0$.

Define $y_m = (C - \lambda_i)^m a_i$. Assume now $AC^i x = 0$. Then $Ap(C)x = 0$, for every polynomial p, so that $Ay_m = 0$. Assume moreover (1.18.a) true. Then, if $y_m = 0$, we have $Ay_{m-1} = 0$, and $Cy_{m-1} = \lambda_i y_{m-1}$ so that $y_{m-1} = 0$. Since $y_m = 0$ for $m = \alpha_i$, by induction, we have $y_m = 0$ for each m, thus $a_i = 0$ and $x = 0$. □

Lemma 11
Assume (A, C) *observable,* C *of size* n*,* $\omega_i \in \mathbb{C}$ *for* $1 \le i \le n$*,* $\omega_i \neq \omega_j$ *for* $i \neq j$*, and* $I - \omega_i C$ *invertible.*

Then, if $A(I - \omega_i C)^{-1}x = 0$ *for each* i*, we have* $x = 0$*. Moreover, if* $v \in \mathbb{C}^n$*, there are some vectors* c_i *in* $\mathbb{C}^p$ *such that*

$$v = \sum_i (I - \overline{\omega}_i C^*)^{-1}A^* c_i. \tag{1.20}$$

Proof. 1. The two claims are clearly equivalent. If f is any analytic function, defined on the eigenvalues of C, then $f(C)$ is a polynomial function of C, because of (1.19). Take $f(C) = (I - \omega C)^{-1}$. The proof consists of finding an explicit formula for the coefficients of this polynomial.

2. Define some numbers x_i by

$$x_i = \omega^i \frac{1 - a_{n-1}\omega - a_{n-2}\omega^2 - \cdots - a_{i+1}\omega^{n-i-1}}{1 - a_{n-1}\omega - a_{n-2}\omega^2 - \cdots - a_0\omega^n}. \tag{1.21}$$

We consider $a_n = -1$, and a_i is defined by (1.19). Thus, the denominator Δ is just the determinant of $I - \omega C$. Let $x'_i = \Delta x_i$. This is a multiple of ω^i, so that there exists some numbers b_{ik}, depending only on a_i, such that

$$\omega^i = x'_i + \sum_{k>i} b_{ik} x'_k. \tag{1.22}$$

For each ω_j, we define x_{ij} and x'_{ij}. By assumption, the determinant of ω^i_j is not zero, so that the determinant of x'_{ij} is not zero (moreover, the denominator of x_{ij} is not zero).

3. We have

$$\sum_{i=0}^{n-1} x'_i C^i = -\sum_{j=0}^{n} a_j \omega^{n-j} \sum_{i=0}^{j-1} \omega^i C^i$$

$$(I - \omega C)\sum_{i=0}^{n-1} x'_i C^i = -\sum_{j=0}^{n} a_j \omega^{n-j} \sum_{i=0}^{j-1} \omega^i C^i (I - \omega C) = -\sum_{j=0}^{n} a_j \omega^{n-j} + \omega^n \sum_{j=0}^{n} a_j C^j.$$

If we apply (1.19) we get

$$(I - \omega C)^{-1} = \sum_{i=0}^{n-1} x_i C^i. \tag{1.23}$$

4. Assume $A(I - \omega_j C)^{-1} y = 0$. Then, for each j, $\sum x'_{ij} A C^i y = 0$. Since the determinant of x'_{ij} is not zero, it follows that $AC^i y = 0$, hence $y = 0$, since (A, C) is observable. ☐

Corollary 5
The space X defined by (1.16) *is of dimension n, and the linear span of $K(z, \omega_i)c_i$ is X.*

Lemma 12
Assume $R_\alpha(f) = \lambda f$ for some constant λ, and f not identically zero. Then $f(\alpha) \neq 0$, and if $\beta = \alpha + 1/\lambda$, then $\beta \notin \Omega$.

Proof. From

$$f(z) = \frac{f(\alpha)}{1 - \lambda(z - \alpha)} = \frac{f(\alpha)/\lambda}{\beta - z}$$

it is obvious that, if $f(\alpha) = 0$, then f is identically zero, otherwise f is not analytic at β. ☐

From the relation

$$[I - (z - \alpha)C]^{-1} - [I - (\beta - \alpha)C]^{-1} = [I - (z - \alpha)C]^{-1} C (z - \beta)[I - (\beta - \alpha)C]^{-1}$$

we get

$$R_\beta(A[I - (z - \alpha)C]^{-1} u) = A[I - (z - \alpha)C]^{-1} C (z - \beta)[I - (\beta - \alpha)C]^{-1} u.$$

Define

$$M = A[I - (z - \alpha)C]^{-1}.$$

Then

$$R_\beta(Mu) = MC[I - (\beta - \alpha)C]^{-1} u. \tag{1.24}$$

This shows that X is R_β-invariant, whenever $I - (\beta - \alpha)C$ is invertible. It also shows that X is R_α-invariant, and R_α is defined by the matrix C: $R_\alpha(Mu) = MCu$.

Assume now that X is R_α-invariant. Define two operators A with range in $\mathbb{C}^p$ by $Af = f(\alpha)$ and C with range in X by $Cf = R_\alpha(f)$. By definition, we have

$$f(\alpha) = f(z) + \alpha R_\alpha(f) - z R_\alpha(f). \tag{1.25}$$

If $B = I + \alpha C$, this is

$$Af = (B - zC)f. \tag{1.26}$$

Replace in (1.23) ω by $z-\alpha$. Then $(B-zC)^{-1} = \sum x_i C^i$, where x_i is some rational function. Then we get $f = \sum x_i AC^i f$, so that f is a rational function, the denominator being the determinant of $I - (z - \alpha)C$. If we want a more precise result, we have to use a basis.

Let $(e_1, \ldots, e_n)$ be a basis of X, $(\epsilon_1, \ldots, \epsilon_p)$ the canonical basis of $\mathbb{C}^p$, and M_{ji} the i-component of e_i. Then $e_i = \sum M_{ji}\epsilon_j$. Define C_{ji} by $Ce_i = \sum C_{ji}e_j$, $B_{ji} = \delta_{ij} + \alpha C_{ji}$ and $A_{ji} = M_{ji}(\alpha)$. Then (1.25) is

$$A = M(B - zC). \tag{1.27}$$

Note that this is $A = MB - zMC$, where MB and MC are matrices, product of the matrices M_{ji}, B_{ji} and C_{ji}. It gives $M = A(B - zC)^{-1}$.

Thus, X has the form (1.16). Assume now $(B - \beta C)u = 0$, $u \neq 0$. If $\lambda = 1/(\beta - \alpha)$, this is the same as $Cu = \lambda u$. Let $f = Mu$. Then $R_\alpha(f) = \lambda f$. We know that this implies $\beta \notin \Omega$. This means that M is analytic in Ω. It also implies $f(\alpha) \neq 0$. But $f(\alpha) = Au$, so that $Au \neq 0$. In other words, (A, C) is observable, and this concludes the proof of the theorem.

Chapter 2

System Theory

We know that a strictly proper stable rational matrix $\mathcal{H}$ can be factored as $\mathcal{H} = CQ^{-1}$, where Q is inner. We show here that the minimum of $\|F - CQ^{-1}\|$ has the form $\psi_F(Q)$, for fixed Q. Some properties of this function ψ will be studied.

2.1 Realization

Definition 3
Let K be a field ($\mathbb{C}$ or $\mathbb{R}$), $\mathcal{H}$ a $m \times p$ matrix of functions over K. A state-space realization *of $\mathcal{H}$ is given by a vector space $\mathcal{X}$ (called the* state-space*) and three linear mappings H, F and G, where G is from K^p to $\mathcal{X}$, F from $\mathcal{X}$ to $\mathcal{X}$, and H from $\mathcal{X}$ to K^m, such that*

$$\mathcal{H} = H(zI - F)^{-1}G. \tag{2.1}$$

The realization is called minimal *if the space $\mathcal{X}$ has minimal dimension. (H, F) is* observable *if for every complex λ, the matrix $\begin{pmatrix} \lambda I - F \\ H \end{pmatrix}$ is of full rank (injective); (F, G) is* reachable *if (G^t, F^t) is observable, i.e. if $\begin{pmatrix} \lambda I - F & G \end{pmatrix}$ is of full rank. The realization is called* canonical *if (H, F) is observable and (F, G) is reachable.*

Note that if $\mathcal{X}$ is of finite dimension, then $\mathcal{H}$ is rational, strictly proper. If F is stable, then $\mathcal{H}$ is stable. We shall prove the converse: if $\mathcal{H}$ is strictly proper, rational, it has a finite-dimensional realization, and if $\mathcal{H}$ is stable, then F is stable for every minimal realization.

Unless stated otherwise, all results are valid in the real and in the complex case.

Theorem 11
We assume that $\mathcal{X}$ is of finite dimension n. A triple (H, F, G) is minimal if and only if it is canonical. For every triple (H, F, G), there exists a minimal triple (H_1, F_1, G_1) such that

$$H(zI - F)^{-1}G = H_1(zI - F_1)^{-1}G_1, \tag{2.2}$$

and if both triples are minimal, there exists a isomorphism U from $\mathcal{X}_1$ to $\mathcal{X}$ such that

$$G = UG_1 \qquad F_1 = U^{-1}FU \qquad H_1 = HU, \tag{2.3}$$

in other words, the following diagram commutes

$$\begin{tikzcd}
 & \mathcal{X} \arrow[r, "F"] & \mathcal{X} \arrow[dr, "H"] & \\
K^p \arrow[ur, "G"] \arrow[dr, "G_1"'] & & & K^m \\
 & \mathcal{X}_1 \arrow[uu, "U"] \arrow[r, "F_1"'] & \mathcal{X}_1 \arrow[uu, "U"'] \arrow[ur, "H_1"'] &
\end{tikzcd}$$

Proof. 1. Consider the matrix $A_q = \begin{pmatrix} H \\ HF \\ \vdots \\ HF^{q-1} \end{pmatrix}$. The matrix A_n is called the observability matrix of (H, F). The Cayley-Hamilton theorem says that $F^n = \sum_{k=0}^{n-1} \lambda_k F^k$, where $z^n - \sum \lambda_k z^k$ is the characteristic polynomial of F. Hence A_q has the same rank as A_n if $q \geq n$. The matrix

$$(G\ FG\ F^2G\ \dots\ F^{n-1}G)$$

is called the reachability matrix of the pair (F, G). We shall show that (H, F, G) is canonical if and only if the reachability and observability matrices are of full rank.

2. Assume A_n of full rank. If $(\lambda I - F)x = 0$, $Hx = 0$ then $HF^i x = \lambda^i Hx = 0$, hence $A_n x = 0$, so that $x = 0$.

Assume now that A_n is not of full rank. There exists a regular matrix U such that $AU = (\, B \quad 0 \,)$, B is of full rank. Write

$$HU = (\, H_1 \quad H_2 \,) \qquad U^{-1}FU = \begin{pmatrix} F_1 & F_2 \\ F_3 & F_4 \end{pmatrix} \qquad U^{-1}G = \begin{pmatrix} G_1 \\ G_2 \end{pmatrix}. \tag{2.4}$$

Since $HF^iU = HU(U^{-1}FU)^i$, and the last columns of this matrix are zero, we get $H_2 = 0$ and $H_1F_1^iF_2 = 0$ $(0 \leq i \leq n-2)$. Moreover B is the matrix whose rows are $H_1F_1^i$. Hence $BF_2 = 0$, $F_2 = 0$.

Let y be any eigenvector of F_4 associated to an eigenvalue λ, and $x = U \begin{pmatrix} 0 \\ y \end{pmatrix}$. Then $(\lambda I - F)x = 0$, $Hx = 0$, so that (H, F) is not observable. In the same fashion (F, G) is reachable if and only if the reachability matrix is of full rank.

3. If (H, F, G) is minimal, then it is canonical. This is because, if the system is not observable, equation (2.4) together with $F_2 = 0$, $H_2 = 0$ gives $H(zI - F)^{-1}G = H_1(zI - F_1)^{-1}G_1$, so that the system is not minimal.

4. Assume that (H, F, G) and (H_1, F_1, G_1) are canonical and $H(zI - F)^{-1}G = H_1(zI - F_1)^{-1}G_1$.

Let $x \in \mathcal{X}_1$. Since (F_1, G_1) is reachable, there are vectors u_i such that $x = \sum_{i=0}^{n-1} F_1^i G_1 u_i$. This representation is of course not unique, so consider also $x = \sum F_1^i G_1 u_i'$. Let $y = \sum F^i G u_i$ and $y' = \sum F^i G u_i'$. The relation $H(zI - F)^{-1}G = H_1(zI - F_1)^{-1}G_1$ says

$$HF^i y = HF^i y' = H_1 F_1^i x \quad (i \geq 0) \tag{2.5}$$

and the observability of (H, F) says $y = y'$. We can define a mapping U from $\mathcal{X}_1$ to $\mathcal{X}$, by defining $Ux = y$. In the same fashion, since (F, G) is reachable and (H_1, F_1) observable, we can define a mapping V from $\mathcal{X}$ to $\mathcal{X}_1$. These mappings U and V are obviously linear, and V is the inverse of U.

By definition $G = UG_1$ (take $x = G_1 u$). On the other hand, (2.5) says

$$HF^iU = H_1F_1^i.$$

For $i = 0$, this gives $H_1 = HU$, hence $HF^i = H(UF_1U^{-1})^i$. Write $UF_1U^{-1} = F + X$. Then $HF^iX = 0$ for every i, $X = 0$, and $UF_1U^{-1} = F$.

5. Assume (H,F,G) is canonical. There exists a minimal realization (H_1,F_1,G_1) such that $H(zI-F)^{-1}G = H_1(zI-F_1)^{-1}G_1$. Since (H_1,F_1,G_1) is canonical, the existence of U with $UF_1U^{-1}=F$ shows that F and F_1 have the same size. □

The converse is the following.

Theorem 12
Let $\mathcal{H}$ be a strictly proper rational matrix of McMillan degree n. Then $\mathcal{H}$ has a minimal realization, the dimension of the state space being n.

Proof. 1. Let $\mathcal{X}=\mathcal{X}(\mathcal{H})$ be the space defined in the previous chapter. Recall that this is the space of polynomial vectors, and $x=y$ in $\mathcal{X}$ if and only if $\mathcal{H}(x-y)$ is a polynomial. The dimension of the space is the McMillan degree of $\mathcal{H}$.

2. If u is a constant vector, we can consider it as a polynomial vector of degree zero, hence as an element Gu of $\mathcal{X}$. Obviously, this defines a linear function G. Now, if $x=y$ in $\mathcal{X}$, then $zx=zy$, so that the multiplication by z is defined in $\mathcal{X}$. Call it Fx. Then F is a linear mapping, and clearly every element of $\mathcal{X}$ can it be written as $\sum F^iGu_i$, so that (F,G) is reachable.

3. For every polynomial x, we can uniquely write $\mathcal{H}x=\sum_{k>K}\alpha_k/z^{k+1}$. If $\mathcal{H}y=\sum_{k>K}\beta_k/z^{k+1}$, then $x=y$ in $\mathcal{X}$ is $\alpha_k=\beta_k$ for $k\geq 0$. Hence, the mapping $x\rightarrow\alpha_0$ is well defined. If we denote it by H, we have $HF^ix=\alpha_i$, so that $HF^ix=0$ for every i implies that $\mathcal{H}x$ is a polynomial, hence $x=0$ in $\mathcal{X}$. Thus (H,F) is observable.

6. Finally, if $H=\sum_{k\geq 0}H_k/z^{k+1}$ (recall that $\mathcal{H}$ is strictly proper), then $HF^iG=H_i$, hence $\mathcal{H}=H(zI-F)^{-1}G$. □

In the remainder of this section, we explain how to get the matrices H, F and G. According to the theorem, it suffices to find a basis (e_i) of $\mathcal{X}$ and compute the Taylor expansion of $\mathcal{H}e_i$ (this gives H). We get F and G if we know how constants and ze_i are expressed in this basis.

Assume first $\mathcal{H}=CQ^{-1}$, where C and Q are coprime. Then $\mathcal{X}=\mathcal{X}'(Q^{-1})$. Define

$$K_Q(z,\omega)=\frac{I-Q(z)Q(\omega)^*}{1-z\overline{\omega}}.$$

One of the results of the next chapter is that we can find $e_i=K_Q(z,\omega_i)c_i$, such that every element $x\in H^2$ can be uniquely written as

$$x=Qa+\sum\lambda_ie_i,$$

where $a\in H^2$ and λ_i is complex. Hence e_i can be chose as a basis. Moreover,

$$\langle K_Q(z,\omega)c\,|\,x\rangle=\sum\lambda_ic^*e_i(\omega),$$

so that finding F and G becomes easy: it suffices to compute some scalar products, evaluate $e_i(\omega)$ and invert a matrix. We do not explain here how e_i is to be chosen. We just consider a simple example, the case where Q is of McMillan degree one. It can be written as $Q=I-(1-\beta_\omega)uu^*$, so that

$$K_Q(z,\mu)=\frac{A}{1-z\overline{\omega}},\quad A=\frac{1-|\omega|^2}{1-\overline{\mu}\omega}uu^*.$$

Taking $e_1=K_Q(z,\omega)u$ gives $A=uu^*$. Because u is of unit norm, we get

$$\lambda=\langle\frac{1-|\omega|^2}{1-z\overline{\omega}}u\,|\,x\rangle$$

Hence $\lambda=(1-|\omega|^2)u^*x(\omega)$. The matrix G is trivial: evaluate this expression for a constant x. The matrix F is even more trivial: if we evaluate at $x=ze_i$, we get $F=\omega$ (since ω is the single pole of $\mathcal{H}$, F has to be equal to ω).

In what follows, we consider the decomposition $\mathcal{H} = ND^{-1}$. Since this decomposition has more free parameters, it is possible to chose one for which the matrices F and G are easy to compute.

Let A be a matrix of polynomials. Denote by d_{ij} the degree of entry A_{ij} and by d_j the maximum of all d_{ij}. This will be called the degree of column j. The column j can be written as $B_j z^{d_j} + C_j$, where C_j is of degree less then d_j, and B_j is constant. In case the matrix whose columns are B_j is of full rank, the matrix A is called *column reduced.*

Lemma 13
Let D be a square matrix, with non-zero determinant of degree n. There exists a unimodular matrix U such that DU is column reduced. The sum of the degrees of the columns is n.

Proof. Assume D not column reduced. There exist some λ_i such that $\sum \lambda_i B_i = 0$. Chose k such that $\lambda_k \neq 0$ and d_k is maximal. Subtract form column k column i multiplied by $z^{d_k - d_i}\lambda_i/\lambda_k$. Then the degree of column k decreases. Note that no column of D can be identically zero, and the degree of the determinant is at most $d = \sum d_i$. If the matrix is column reduced, the coefficient of z^d in the determinant is the determinant of the B_j, hence is not zero, so that $n = \sum d_i$. $\square$

Lemma 14
Let D be a square matrices of polynomials, with non-zero determinant. There exists a permutation matrix V, a unimodular matrix U such that $A = VDU$ satisfies the following conditions, where d_{ij} is the degree of A_{ij}:

1. *Diagonal elements have degree $d_{ii} = d_i$, they are monic, and $d_i \leq d_j$ for $i \leq j$.*
2. *$d_{ij} < d_i$ for $j > i$;*
3. *$d_{ij} \leq d_j$ for $j < i$;*
4. *$d_{ij} < d_i$ for $j < i$.*

Conditions (1), (2) and (4) imply that row i is of degree d_i. In fact, there is only one term on the row which is of degree d_i, this element is on the diagonal. In particular, the matrix A is row reduced (its transpose is column reduced). Since diagonal terms are monic, for the matrix $B = VDU - \mathrm{diag}(z^{d_i})$, row i is of degree $< d_i$. Thus column j is a linear combination of vectors of the form $e_k z^i$, with $i < d_k$, where e_k is the canonical basis of $\mathbb{C}^p$.

Conditions (1), (2) and (3) imply $d_{ij} \leq d_j$, (and $d_{ij} < d_j$ if $i < j$). Thus d_j is the degree of column j, and A is column reduced, because the matrix that must be of full rank is triangular.

Proof. Let's start with D, and apply the following operations. We may assume that D is column reduced, with degree d_j on column j. We may also assume that the sequence d_j is non-decreasing, by re-ordering the columns.

We assume that for every i, every $j < k$, d_{ij} satisfies conditions (2) and (3). Moreover, if $j < i < k$, condition (4) is true. These relations are trivially true for $k = 1$, because they are empty. We apply some operations to the matrix so that they become true for $k + 1$.

Let a_i be the term A_{ik}, and b_i its degree. Consider index $i < k$ such that $b_i - d_i = t$ is maximal. If $t < 0$, then relation (2) is true. Otherwise, subtract from column k the column i, multiplied by $\alpha_i z^t$, where α_i is the leading coefficient of a_i.

Consider now what happens on row j. First, all elements of column i are of degree at most d_i, so that we add terms of degree at most $d_i + t$. This is $b_i \leq d_k$. Thus, the degree of row k will remain at most k. Assume $j \neq i$. Then $d_{ji} < d_j$. This means that we add a term of degree less than $d_i + t$. On the other hand, if $i = j$, we add a term of degree $b_i = d_i + t$, and by construction, the leading coefficient vanishes. After this operation, the number of rows i such that $t = b_i - d_i$ has diminished. If all rows satisfy $b_i + d_i < t$, we can chose a smaller t.

Assume now that condition (2) is satisfied. For $i \geq k$, we have $d_{ik} \leq d_k$, which is condition (3). But equality must hold somewhere (otherwise the degree of the determinant would be smaller that $\sum d_i$). By permuting rows, we may assume that element of row k has degree d_k (this is the only operations on rows that will be performed). By multiplication of the column by a scalar, we may assume that this element is monic.

We must now modify our matrix so that condition (4) holds, namely $d_{kj} < d_k$, for $j < k$. We know $d_{kj} < d_j$, so that there is nothing to do if $d_j < d_k$. Otherwise, it is of degree $\leq d_k$, and equality can happen. We handle columns $k-1$, $k-2$, etc., in order. If (4) is true, there is nothing to do; otherwise, let α be the leading coefficient of A_{kj}.

Subtract from column j the column k multiplied by α. On row i, the following conditions must be true: if $i < j$, the degree must be less than d_i. This is true before the operation on columns j and k, and remains true after subtraction. If $i > k$, the degree must be at most d_j and d_k. This is true before the subtraction, and remains true after it, because $d_j = d_k$. If $j \leq i < k$, the degree on column k is less than d_k. The degree on column j is at most d_j (equal if $i = j$, different otherwise, because these columns are already handled). Finally, if $i = k$, we add the quantity that makes the degree decrease. $\square$

The proof is a bit complicated, but the algorithm is easy to implement. Finding U such that DU is column reduced is numerically harder, because we have to compute determinants.

Lemma 15

Let D be a square matrix of polynomials, $A = VDU$ as in the previous lemma. If x is any polynomial vector, either $x = 0$, or $y = VDx$ has at least one entry y_i of degree $\geq d_i$.

Proof. Let $t = U^{-1}x$. Then $y = At$, and $t = A^{-1}y$. Let δ be the determinant of A and B the matrix of cofactors. We have $\delta t = By$. If n is the degree of δ, then B is column reduced, the degree of the column i being $n - d_i$ (this is because A is row reduced). Thus entries of By are of degree $< n$ if entries of y_i are of degree $< d_i$. This implies $t = 0$. $\square$

Lemma 16

Let D be a square matrix of polynomials, U and V as above. Let $A = VD$. Let ϵ_j be the set of vectors $e_i z^j$, with $j < d_i$.

For every polynomial vector y, there exists a vector polynomial x, and constants λ_i, uniquely defined, such that

$$y = Ax + \sum \lambda_i \epsilon_i. \tag{2.6}$$

Proof. Unicity comes from the previous lemma. Of course, elements $z^j e_i$ with $j < d_i$ can be written in this form, with $x = 0$. We want to show that this is true for every j.

Every y can be written as a sum of elements of the form $z^j e_i$, each term having one of the following forms: either $j < e_i$, and $z^j e_i$ is some ϵ_k, or $d_i \neq 0$, and $z^j e_i = z^k \epsilon_l$, or $d_i = 0$. In the last case, we know that $e_i - AUe_i$ is a linear combination of the ϵ_i. Thus, for some x

$$z^j e_i = Axz^j + \sum \lambda_k \epsilon_k z^j.$$

Thus, this last case reduces to the other ones.

We also know that $z^{d_i} e_i$ can be written in this form (with $x = Ue_i$). This means that $z\epsilon_i$ can be written in this form.

$$z\epsilon_i = Ax_i + \sum \lambda_{ij} \epsilon_j. \tag{2.7}$$

From this relation, it is obvious that we can write $z^k \epsilon_i$ in this form. $\square$

This lemma says that ϵ_i is a basis of $\mathcal{X}(D^{-1})$, the state-space of $\mathcal{H}$, and the matrices of F and G in this basis are easy to obtain from A.

2.2 Study of the approximation problem

Let

$$F = \sum_{k=0}^{\infty} \frac{F_k}{z^{k+1}}, \qquad G = \sum_{k=0}^{\infty} F_k^* z^k.$$

We assume that $F \in H_2^-$, that is, $G \in H^2$, or $\sum \|F_k\|^2 < \infty$. We want to find $\mathcal{H}$, strictly proper, stable, of McMillan degree n such that

$$\|F - \mathcal{H}\| \text{ is minimal.}$$

We shall assume in what follows that G (or F) has only a finite number of coefficients, hence $G = \sum_{k=0}^{M} G_k z^k$ is a polynomial. In principle, the theory works with an infinite number of coefficients, but the program can handle only a finite number of inputs. Moreover, unless stated otherwise, we consider the L^2 norm.

2.2.1 Scalar case

Statement of the problem In this subsection, we use the factorisation $\mathcal{H} = ND^{-1}$. We know that $\mathcal{H}$ is of degree n if D and N are coprime, and $\det D$ is of degree n. If F is a $m \times p$ matrix, then D is a square $p \times p$ matrix, but there is no simple bound on the degree of the entries of D. If however $p = 1$, we know that D is a scalar, whose degree is n. In the case $m = 1$, we transpose F, apply the theory, and transpose the result.

Hence, we may assume $\mathcal{H} = P/q$, where P is a vector of polynomials and q is a polynomial. The condition "P and q coprime" is that there exists a polynomial vector X, a polynomial y such that $X^*P + qy = 1$. This means that q is coprime to the gcd of the entries P_i of P. The problem is hence:

Find P and q such that

$$\|F - P/q\| \tag{2.9}$$

is minimal, under the conditions: 1) q is stable, of degree n, 2) entries of P have degree $< n$, and 3) P and q are coprime.

This problem has been studied extensively in [2]. Before solving this problem, let's discuss the conditions introduced before. It is obvious that we can take q of the form $z^n + \sum_{i<n} q_i z^i$. Then q will be of degree n. The set of stable polynomials q is bounded (with the usual metric), but is not compact. This will introduce some difficulties (in particular, there is no guaranty that (2.9) has a minimum). What we shall do is establish formulas for the stable case, and discuss what happens otherwise. In particular, we shall see that minimising (2.9) is equivalent to minimising $\psi(q)$ on the set of all polynomials with roots in $\overline{\mathbb{U}}$, which is a compact.

We can trivially relax the condition that P is of degree $< n$, for if $P = Aq + B$ (Euclidean division), then $\|F - P/q\|^2 = \|F - B/q\|^2 + \|A\|^2$, and this can be minimal only if $A = 0$. On the other hand, we can add the condition $\deg P < n - k$, see section 2.3.7.

We shall ignore the last condition and, later on, check if it is satisfied. The trouble is essentially that the set of polynomials q, with roots in $\overline{\mathbb{U}}$, which are coprime to $L(q)$, is not compact ($L(q)$ is the unique P which minimises (2.9) when q is fixed). In fact, if $F = P_0/q_0$ is rational and has McMillan degree $< n$, if q_1 is any stable polynomial, which is coprime to q_0, and whose degree is such that $\deg(q_0 q_1) = n$, then $P/q = P_0/q_0 + \epsilon/q_1$ has McMillan degree n for non-zero ϵ, and the norm of the distance to F is (a constant times) ϵ. Thus $\min \|F - P/q\|$ is zero, and the minimum is *not* of degree n.

We shall prove that the algorithm we propose is a good one, namely that it gives P and q which are coprime, whenever this is possible, in fact, in all cases where F is not a rational function of McMillan degree $< n$.

The equation for ψ For every polynomial S we have

$$\|F - \frac{P+S}{q}\|^2 = \|F - \frac{P}{q}\|^2 + \|\frac{S}{q}\|^2 - 2\Re\langle F - \frac{P}{q} \mid \frac{S}{q}\rangle.$$

A necessary and sufficient condition for P/q to be minimal is

$$\langle F - \frac{P}{q} \mid \frac{S}{q} \rangle = 0 \tag{2.10}$$

for every polynomial S of degree less than n. This means that P/q is the orthogonal projection of F on the space of all functions S/q. Note: this is true whenever the norm used in (2.9) is associated to a scalar product. In case we use the H^2 norm, more can be said, in fact, we have a formula for the projection.

Let's consider

$$G\tilde{q} = Vq + R \tag{2.11.a}$$

the Euclidean division of $G\tilde{q}$ by q. Here R is a polynomial of degree $< n$, and V an element of H^2 (if G is a polynomial, it is a polynomial with the same degree as G). Define

$$P = \tilde{R} = z^{n-1}\overline{R}^t(1/z). \tag{2.11.b}$$

Recall that $\check{V} = (1/z)\overline{V}^t(1/z)$ (this is in H_2^-) and $\tilde{q} = z^n\overline{q}(1/z)$. Then Equations (2.11) are

$$F - \frac{P}{q} = \check{V}\frac{\tilde{q}}{q}. \tag{2.12}$$

If X is the RHS of this equation, then $\|X\| = \|\check{X}\|$ and $\check{X} = Vq/\tilde{q}$. Since q is stable, $q/\tilde{q}$ is inner and $\|Vq/\tilde{q}\| = \|V\|$. Thus

$$\|F - \frac{P}{q}\|^2 = \|V\|^2. \tag{2.13}$$

Note: this relation is only true if we use the L^2 norm.

Let S be any polynomial of degree less than n. We have

$$\langle F - \frac{P}{q} \mid \frac{S}{q} \rangle = \langle \check{V}\frac{\tilde{q}}{q} \mid \frac{S}{q} \rangle = \langle V\frac{q}{\tilde{q}} \mid \frac{\tilde{S}}{\tilde{q}} \rangle.$$

(the last equation is obtained by replacing x by $\check{x}$). The last scalar product is also $\langle V \mid \tilde{S}/q \rangle$. This is zero because $p/q \in H_2^-$ if p is a polynomial, $\deg p < \deg q$.

Define $L_F(q) = P$ and $\psi_F(q) = \|V\|^2$. We shall sometimes write L_G and ψ_G instead of L_F and ψ_F. Equations (2.10) and (2.13) say now: $\|F - P/q\|$ is minimal if and only if $P = L_F(q)$ and $\psi_F(q)$ is minimal. Taking $S = P$ in (2.10) gives

$$\|F - \frac{P}{q}\|^2 = \|F\|^2 - \langle F \mid \frac{P}{q} \rangle. \tag{2.14}$$

Note. Let $F' = \lambda F$, where λ is a positive, real number. Then $G' = \lambda G$, $L_{F'}(q) = \lambda L_F(q)$ and $\psi_{F'}(q) = \lambda^2\psi_F(q)$. In the case $\lambda = 1/\|F\|$, we have $\|F'\| = 1$, this simplifies a bit (2.14). In fact, we have

$$0 \le \psi_{F'}(q) \le 1.$$

The first inequality is an equality if and only if $F = P/q$, the second is an equality if and only if $P = 0$.

2.2.2 Matrix case

We consider here the factorisation $\mathcal{H} = Q^{-1}C$, where Q is inner, of McMillan degree n, and C is in H^∞. According to lemma 9, $Q^{-1}C$ is strictly proper stable, of McMillan degree $\le n$, if and only if $\tilde{q}C$ and $\tilde{D}C$ are polynomials with entries of degree $< n$, assuming

$$Q = \frac{D}{\tilde{q}} \qquad Q^{-1} = \frac{\tilde{D}}{q}.$$

Condition (2.10) becomes

$$\langle F - Q^{-1}C \,|\, Q^{-1}C' \rangle = 0. \tag{2.15}$$

The equivalent of (2.11.a) is

$$G\tilde{D} = Vq + R. \tag*{(2.16.a)}$$

Let $S = RD/q$. The relation $D\tilde{D} = q\tilde{q}$ says

$$G\tilde{q} = VD + S. \tag*{(2.16.b)}$$

Let $C = \tilde{R}/\tilde{q}$, then $\tilde{q}C = \tilde{R}$ is a polynomial, and $\tilde{D}C = \tilde{S}$; note that $S = G\tilde{q} - VD$ is in H^2, but $S = RD/q$ says that S is rational and stable, so that S is a polynomial. Obviously R and S have degree $< n$. Now equations (2.16) give a function $\mathcal{H} = Q^{-1}C = \tilde{S}/q$, which is rational, stable, strictly proper, of McMillan degree $\leq n$. It minimises $\|F - Q^{-1}C\|$: in fact, equation (2.16.a) gives

$$F - Q^{-1}C = Q^{-1}\check{V} \tag{2.17}$$

so that the scalar product in (2.15) is

$$\langle F - Q^{-1}C \,|\, Q^{-1}C' \rangle = \langle Q^{-1}\check{V} \,|\, Q^{-1}C' \rangle = \langle VQ \,|\, \check{C}'Q \rangle = \langle V \,|\, \check{C}' \rangle.$$

The last quantity is zero since $V \in H^2$ and $\check{C}_2 \in H_2^-$.

The equivalent of (2.13), (2.14) is now, with $P = \tilde{S}$, $Q^{-1}C = P/q$:

$$\|F - Q^{-1}C\|^2 = \|F - P/q\|^2 = \|V\|^2 = \|F\|^2 - \langle F \,|\, Q^{-1}C \rangle = \|F\|^2 - \langle F \,|\, P/q \rangle.$$

Note: Equation (2.16.a) says that $GQ^{-1} = V + R/q$. But V is in H^2 and R/q is in H_2^- so that these terms are orthogonal, hence

$$\|F\|^2 = \|V\|^2 + \|R/q\|^2. \tag{2.18}$$

The quantity $\|V\|^2$ will be denoted by $\psi_F(Q)$, and the quantity C by $L_F(Q)$.

2.3 Properties of ψ

2.3.1 Optimisation methods

The problem now is to find the minimum of ψ. In the scalar case, ψ is a function of q, where q is a stable polynomial. Let S be the set of stable polynomials. We shall see that S is an open subset of $\mathbb{C}^n$, which has compact closure. Thus, there exists q, in the closure of S, that minimises ψ. This q has the form $q = q_1q_2$, where q_1 is stable and q_2 has all its roots in $\mathbb{T}$. We shall show that ψ is defined for these polynomials, and that $\psi(q) = \psi(q_1)$. If q is a minimum of ψ, then q_1 is also a minimum of ψ (at smaller order). We shall show that, in this case, the gradient of ψ points outwards (more about this will be explained later), and this gradient can only be zero if $\psi(q_1) = 0$. Said otherwise, either $F = P/q$, where P/q has McMillan degree $< n$, or ψ has a minimum in S.

In the matrix case, we have to parameterise all inner functions Q of McMillan degree n. We shall give formulas of the form $Q = f(u, y)$. Here y is constraint to be in a set S_u. When we use the Schur formulas, it happens that S_u is independent of u, and it is much easier to check $y \in S_u$ than to check $q \in S$ in the scalar case. The set S_u has compact closure. The question is: what happens if $y = \lim y_k$ is in the boundary of S_u. It can happen that Q has a limit, and this limit can be of McMillan degree n, or of smaller degree; sometimes the limit does not exist. In the first case, there exists u' such that $f(u, y_k) = f(u', y'_k)$, and y'_k has a limit in $S_{u'}$. In the second case this is false. The conditioning of the formulas are in general bad in case y is near the boundary of S_u. For this reason, we try to use a "better" u whenever possible.

As in the scalar case, there exists Q that minimises ψ, and there exist infinitely many quantities u such that $Q = f(u, y)$ (in fact, almost every u can be chosen). However, for fixed u, it is not true that

$\psi(f(u,y))$ has a minimum in S_u (consider for instance $f(u,y) = e^{i(u-y)}$, with the restriction $|y| < \pi$. Then the set of all $f(u,y)$ is $\mathbb{T}$, with $-e^{iu}$ removed). Assume that our algorithm finds a sequence Q_k such that $\psi(Q_k)$ is decreasing. It is possible, if u is fixed, that y_k has a limit, which is on the boundary of S_u, and that this limit is such that $Q = f(u,y)$ is not a minimum of ψ (consider the example above, with $\psi(Q) = |Q+1|^2$, the starting point is $-e^{i\epsilon}$, and the unreachable point on the boundary is $-e^{i\epsilon/2}$). This means that we must use an algorithm that changes the value of u from time to time. We use an algorithm that, in the complex case, always finds a "good" value of u. As a consequence, we cannot assert that the limit of the sequence Q_k is a critical point of ψ.

One can design an algorithm that finds a minimum of ψ in the following way. We construct a sequence of points q_k, such that $q_{k+1} = q_k + t_k d_k$. We first define a direction d_k such that the function $f(t) = \psi(q_k + td_k)$ is decreasing at $t = 0$. This means that the derivative at $t = 0$ is negative. If $\nabla\psi$ is the gradient of ψ, it means that the scalar product of $\nabla\psi$ and d_k is negative. This is true if for instance $d_k = -W\nabla\psi$, where W is a symmetric, definite positive matrix. There is at least one choice for W: the identity matrix. In the case where the Hessian of ψ is positive definite, a good choice is to take for W the inverse of the Hessian. In this case, we can take $t_k = 1$: If the initial condition q_0 is near enough to a local minimum of ψ, then this will converge to the local minimum. There are other methods, like BFGS, which construct a matrix W that has good properties.

Once the direction d_k is found, we are looking for t such that $f(t) = \psi(q_k + td_k)$ is minimal. It can be shown that, in certain cases, it is not necessary to find the absolute minimum of f, a good approximation is enough. In general, we do not know under which conditions these algorithms work.

Consider what happens in the scalar case. If $q = \sum q_i z^i$, then $\nabla\psi$ will be the polynomial $\sum_{i=0}^{n-1} z^i \partial\psi/\partial q_i$. In the complex case, each q_k has the form $u_k + iv_k$. Then $\partial\psi/\partial q_k = \partial\psi/\partial u_k + i\partial\psi/\partial v_k$. This is a well defined polynomial. The function $f(t)$ is defined, provided that $q_k + td_k$ is a stable polynomial. Note that the set of stable polynomials is not convex, so that the set of all t for which $f(t)$ is defined is not an interval. However, there exists $t_0 > 0$ such that the polynomial is stable for $0 < t < t_0$, and such that, for $t_1 > t_0$, there exists t_2, for which the polynomial is unstable and $t_0 < t_2 < t_1$. We have to find the minimum of f on $[0, t_0]$. This minimum may be $t = t_0$. We shall explain later what can be done in this case. Note that finding t_0 is not obvious. Hence, we just look for an approximation of it. Another way to proceed is the following. It is possible to define ψ for all polynomials. Hence $f(t)$ is defined for all t. Moreover, the limit of f is $+\infty$ for $t = +\infty$. This means that f has a minimum, and we can compute it (what we do, in fact, is to find a local minimum). If the polynomial we find is stable, there is no problem. Otherwise, we can check if f has another local minimum. We can also pretend that the direction d_k is wrong. In this case, we can try $-\nabla\psi$ as a better direction.

The situation is much different in the matrix case. We must have $y_k + td_k \in S_u$, and this condition is easy to check. This means that computing t_0 is not a problem. However, on the boundary, the conditioning of ψ and its derivatives is very bad. For this reason, we always minimise f on an interval of the form $[0, (1-\epsilon)t_0]$, for some small ϵ. This has as side effect that q_k never reaches the boundary (and that we cannot be sure that the algorithm converges).

In practice this method works well: if we chose a random initial condition, then, in some cases, we reach the boundary, and the algorithm stops. But this is fast. Otherwise, we stay in the domain, and find a minimum. This may take a longer time. Hence the strategy: chose random initial conditions, until a local minimum is found. Instead of choosing random initial conditions, we can chose a condition for which ψ is small. If $\psi(Q)$ is smaller than $\psi(Q')$, for every Q' of degree less than n, then we can never reach the boundary of the manifold (as in the scalar case, if we are on the boundary of the manifold, we have $\psi(Q) = \psi(Q')$, where Q' is of degree $< n$). Note however that this does not mean that we cannot reach the boundary of the chart.

We shall also see that, if we take as starting point a point on the boundary, constructed from Q' such that $\psi(Q) = \psi(Q')$, then $-\nabla\psi$ points into S_u (in a degenerate case, it might be tangent to the boundary), provided that Q' is a local minimum of ψ at order $n' < n$. This will be explained in the next chapter.

There is another method of finding a local minimum of ψ: we integrate the differential equation

$$\frac{dq}{dt} = -\nabla\psi(q(t)). \tag{2.19}$$

The function $\psi(q(t))$ is decreasing. If q remains in a compact, then $q(t)$ is defined for every t, and $q(\infty)$ exists and is a critical point of ψ. This will be a local minimum, because other critical points are numerically unstable.

A general method for solving the equation $dy/dt = f(y,t)$ is the following. Let $t_n = t_0 + nh$, where t_0 is the initial value of t, and h is the step. We define $y_r = s_r(h)$ for $0 \le r < k$, for some functions s_i (obviously $s_0 = y_0$, the initial condition for y). For each n, we write

$$\sum_{i=0}^{k} \alpha_i y_{n+i}/h = \phi_f(t_n; y_{n+k}, y_{n+k-1}, \ldots, y_n; h) \tag{2.20}$$

(for details, see [11]; we use a Gear method as described in [10]). The method is explicit in case ϕ_f does not depend on y_{n+k}, it is implicit otherwise. Implicit methods are more stable, but we have to solve a non-linear system. The simplest method is the Euler method, $y_{n+1} = y_n + hf(t_n, y_n)$. A well-known method is RK4 (Runge-Kutta of order four), it requires four evaluations of f at each step. In methods of type Adams/Gear, we consider the Lagrange interpolation polynomial of degree k whose value at t_i is y_i $(n \le i \le n+k)$ in order to find α_i and ϕ_f.

The local error is the difference $y_{n+k} - y(t_{n+k})$, assuming $y_i = y(t_i)$ for $i < n+k$. In general, this is small if h is small. However, the global error (the difference between the exact value of $y(t)$ and the computed value) is not necessarily small if h is small (essentially because of rounding errors, their limit is not 0 for small h). What happens when $t \to \infty$ and $h \to 0$ is in general unknown. What we know is just that the limit of the execution time is ∞. What we desire is to compute the solution as fast as possible.

We do not explain here how the integrator computes the step size h. On figure 2.1, we plotted the distance between $y_{100(k+1)}$ and y_{100k} in a case where the integrator fails to find a minimum.

The idea is to take it as large as possible, keeping the local error small (say 10^{-5}), if we are far from the solution, and much smaller otherwise. The main idea here is that $q(t)$ will converge to q_0. For small t, we do not need much precision, what we need is just to go in the right direction. However, if we are near q_0, more precision is needed, for otherwise, we would circle around q_0.

In order to explain what happens near q_0, consider the following example. Assume that $q = (x,y)$, and $\psi(q) = (ax^2 + by^2)/2$. Now, ψ has a minimum at the origin. If we use a method of order one, we get

$$q(t_0 + h) = (x_0 - ax_0h, y_0 - by_0h).$$

We must have $h < 1/a$, and $h < 1/b$. This gives a condition on h that is independent of the current point. Note that, in general, if we are far from the minimum, h should be large when $\nabla\psi$ is small (because if $h\nabla\psi$ is small, then $q(t+h) - q(t)$ is small, if it is too small, we have $q(t+h) = q(t)$ numerically). Now, if a is much smaller then b, it means that the y-part of q will converge, while the x-part will not converge.

This is a typical flaw in this method. In case q is far from the optimum, then the method works well, if q is near, it does not work. For this reason, we use a mixed method. Let $I_K(q)$ be the result of the integration scheme, with q as initial condition, after K steps. Then $I_K(q)$ is (an approximation to) $q(t)$, the solution of the differential equation, for some t. Let $N_0(q) = q - W^{-1}\nabla\psi(q)$, where W is the Hessian of ψ. Let $N(q)$ be defined as follows: we start with q, and replace it by $N_0(q)$, as many times as needed. If W is not definite positive, we return nothing. If the process does not converge after some iterations, we return nothing. If q is too far from the initial q, we return nothing. Otherwise, we return q.

Now the method is the following: replace q by $I_K(q)$, and consider $N(q)$. If $N(q)$ is defined, this is the end of the algorithm. Otherwise, we restart. This works, because $N(q)$ is defined when q is near a local minimum q_0, and the result is q_0. We shall discuss convergence of the Newton method later.

In chapter 4, we shall give the code that computes ψ, its first derivative and the second derivative. These computations are done in double and quadruple precision. Quadruple precision is much more time consuming than double precision (by a factor of ten or so), but is needed for the following reasons.

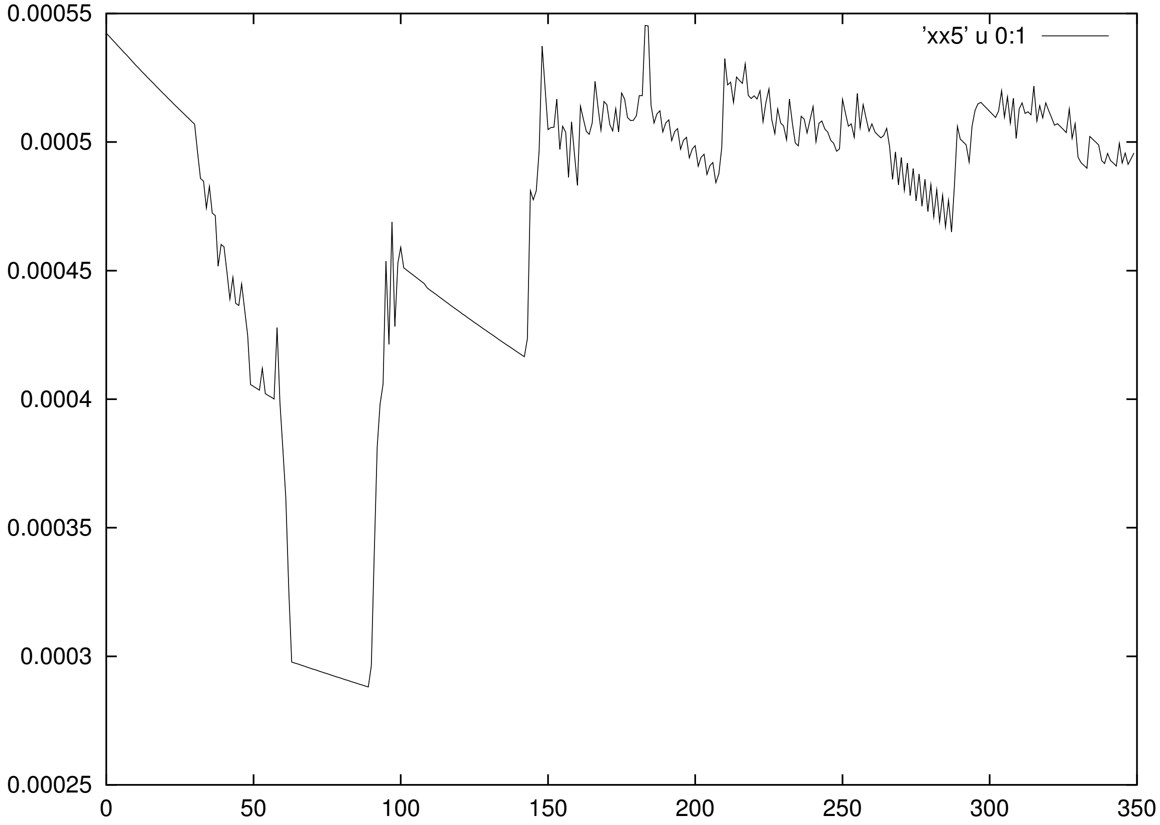

Figure 2.1: In this example, the integrator has some difficulties, and uses a order-one method, $y_{k+1} = h_k.f(y_k)$. The function f is nearly constant, and can see the value of the step h_k, by plotting the difference between $y_{100(k+1)}$ and y_{100k} for some values of k.

First, as shown in the example, it can happen that the integrator uses something like $q(t+h) = q(t) + h\delta q$, with small h. If we are near a local minimum, we can ignore this, because Newton will help us. There are cases where we cannot apply the Newton method. This means that, from time to time, we use quadruple precision for this expression, so that every component of q will vary. There are other cases where $\nabla\psi$ is not precise in double precision. We can find, in some cases, two ways to compute $\nabla\psi$, a slow and precise one, and a fast, but less precise one. In some cases, the second method is faster using quadruple precision than the first one using double precision. Note that the integrator needs the Hessian of ψ, but there is in general no problem of precision (the documentation of the lsode program says that, if computing the Hessian costs too much, it suffices to compute the diagonal elements).

The situation is quite different for Newton. Recall that the Hessian at the minimum has to be definite positive. There are cases where one eigenvalue is near zero, and is negative, when computed in double precision, but positive when computed in quadruple precision. This not the general case, so that we compute first the Hessian in double precision, and if this fails with a small negative eigenvalue, we re-compute it in quadruple precision.

The main problem with the Newton algorithm is the convergence test. We pretend that we have found the minimum in case: $q_{k+1} - q_k$ is small, $\psi_{k+1} - \psi_k$ is small, and ψ'_{k+1} is small. The question is in the definition of "small". In the matrix case, we know that $y \in S_u$ (recall that y is the equivalent of q and S_u does not depend on u). In the scalar case, q must be stable, and this gives a bound on the coefficients of q, which depends on the degree n. In the code, we say that $q_{k+1} - q_k$ is small if less than 10^{-14}, and large if greater than 10^{-8}. Moreover $0 \le \psi \le 1$. We say that $\psi_{k+1} - \psi_k$ is small if smaller than 10^{-13}. The big problem is with $\nabla\psi$ small. This is due to the fact that more operations (hence more rounding errors) are involved in computing $\nabla\psi$ than in computing ψ. This means that 10^{-8} can be considered a small value for $\nabla\psi$. In some cases, this is much too big. The trick is the following. We compare the norm of $\nabla\psi_k$, computed in quadruple precision, to the value of $\nabla\psi_0$, computed in double precision. Since the Newton method is quadratic, it is very easy to make the quotient small. This gives a method that works well in any case.

The integration/Newton method has an advantage over the QuasiNewton methods: it find $q(\infty)$ where $q(t)$ is the solution of the differential equation, instead of finding something that depends on how we chose the directions d_k, and how precise we are when we minimise in the direction d_k.

However, in the matrix case, we solve

$$\frac{dy}{dt} = -\nabla\psi(f(u, y(t))). \tag{2.21}$$

Obviously y depends on u, but $Q = f(u, y)$ depends also on u. This has two consequences: a) it can happen that, for some u, the trajectory remains inside the manifold, while for some other u, it goes out. This is bad news; b) the time spent to integrate depends on u. In particular, for the example given above, one could chose u such that $a = b$, in other words, chose u such that the Hessian of ψ has a good conditioning. This is good news; however we have absolutely no idea of how to chose u in order to do this.

Choice of initial conditions. Let $C(q)$ be the set of all q' such that the value at $t = \infty$ of the solution of the differential equation is q, when the initial condition is q'. Let q_0 be the minimum of ψ. Then any initial condition in $C(q_0)$ will work. We do not know q_0, neither $C(q_0)$. Let C be the union of all $C(q)$, where q is a local minimum of ψ. Points that are not in $C(q)$ are points for which the solution of the differential equation does not remain in the manifold, or the limit is a critical point of ψ that is not a local minimum. We can ignore these points: if q_0 is a saddle point, q_1 an initial condition for which $q(\infty) = q_0$, then rounding errors of the integrator process will modify q enough that the limit is another critical point.

The technique we use is the following. Let ψ_0 be the minimum of ψ at order $< n$. In case $\psi(q) < \psi_0$, then $q \in C$, because, if for some t, $q(t)$ is on the boundary of the manifold, we have $\psi(q(t)) < \psi(q) < \psi_0$. On the other hand, $q(t) = q_1 q_2$, where q_2 has all its roots on $\mathbb{T}$, so that $\psi(q(t)) = \psi(q_1) \ge \psi_0$.

Let $q = q_1 q_2$, and assume that q_1 is the minimum of ψ at order $n-1$. We know that $q(t)$ is stable for small t, for almost every q_2 having its roots on $\mathbb{T}$. Thus $q \in C$. The theory says: if F, the quantity we have to approximate, is near enough to a rational matrix of degree n, than ψ has a unique local minimum, which is the global minimum. In other words, starting with q gives us the minimum of ψ.

This is a beautiful theory, but the trouble is now: how to find q_1. At order $n-1$, ψ may have more than one local minimum (there is absolutely no reason why F should be near a matrix of degree $n-1$). We can chose a local minimum, and hope. This is how we do it (essentially, because we do not know what else can be done in the non-scalar case).

Another approach is the following. We start with any initial condition at order n. We integrate. If we find a local minimum we stop. If we go out, we write $q = q_1 q_2$, where q_1 is stable, q_2 has roots on $\mathbb{T}$. We use q_1 as initial condition at order n_1. This gives a local minimum q_1' at order n_1. We continue with $q_1' q_2'$. We shall see later that, for almost every q_2', this is a good initial condition.

2.3.2 Primality

The aim of this subsection is to show the following theorem.

Theorem 13
If Q is a local minimum of ψ_F, then Q is coprime to $L_F(Q)$ unless F is rational, of McMillan degree less than n.

Some assumptions have to be made on the norm we use, because the result is false for the infinity norm.

Lemma 17
Let q be a polynomial of degree n. There are at most $2n$ points on $\mathbb{T}$ such that $q(z)$ is real.

Proof. Let $z = (1 - t^2 + 2it)/(1 + t^2)$. If $z = e^{i\phi}$, then $t = \tan(\phi/2)$. Clearly, $(1 + t^2)^n \Im(q(z))$ is a polynomial of degree at most $2n$ as a function of t. $\square$

Lemma 18
Let f be a rational function, that has no pole on $\mathbb{T}$, no zero on $\mathbb{T}$. Let k be the number of stable zeroes minus the number of stable poles, and m the absolute value of k. There exists m numbers x_i and y_i, such that $x_1 \le y_1 \le \dots \le x_m \le y_m \le x_1 + 2\pi$, such that $f(e^{ix_j})$ is real negative, and $f(e^{iy_j})$ is real and positive.

Proof. Consider

$$I(f) = \frac{1}{2\pi} \int_0^{2\pi} \frac{f'(e^{i\theta}) e^{i\theta}}{f(e^{i\theta})} d\theta.$$

Then $I(z - \alpha)$ is zero if $|\alpha| > 1$, it is one if $|\alpha| < 1$. Since $I(f_1 f_2) = I(f_1) + I(f_2)$, we have $I(f) = k$, the number of stable zeroes minus the number of stable roots.

Assume $f(e^{i\theta}) = re^{i\phi}$ where ϕ is a continuous function. Near every point z_0 on $\mathbb{T}$, for which $f(z_0)$ is not zero, we can write $f(z) = \exp g(z)$, and ϕ is the imaginary part of g, modulo 2π. Since the real part of $zf'(z)/f(z)$ is the derivative of ϕ with respect to θ, we get

$$I(f) = \frac{\phi(2\pi) - \phi(0)}{2\pi}.$$

Thus $\phi(2\pi) = \phi(0) + 2k\pi$. Since the interval $[\phi(0), \phi(2\pi)]$ is of length $2k\pi$, it is possible to find $2|k|$ numbers of the form $j\pi$ (j integer) hence θ such that $\phi(\theta)$ is of this form. $\square$

Lemma 19
Let q be a stable polynomial of degree n. The best approximation in H^∞ norm of $\tilde{q}/(zq)$ at degree $\le n$ is zero.

Proof. Let's consider a function p/r. Replace f by $\check{f}$, so that all functions are analytic in $\mathbb{U}$. The distance is

$$D = \sup_{z\in\mathbb{T}} |\frac{q}{\tilde{q}} - \frac{\tilde{p}}{\tilde{r}}|.$$

Define $N = (\tilde{p}\tilde{q})/(\tilde{r}q)$. Since $|q/\tilde{q}| = 1$ on $\mathbb{T}$, we have

$$D = \sup_{z\in\mathbb{T}} |1 - N|.$$

If $\tilde{p} = 0$, then $D = 1$. All we have to show is that this quantity is greater than one otherwise. It suffices to find $z \in \mathbb{T}$ such that the real part of $N(z)$ is negative.

The main assumptions we make is that r is stable, and that the degree of $\tilde{p}$ is less than n, the degree of q. Assume first that $\tilde{p}$ has no zero on $\mathbb{T}$. Then $I(N)$ is negative, so that there exists $z \in \mathbb{T}$ such that $N(z)$ is real and negative.

Assume now that $\tilde{p}$ has k zeroes on $\mathbb{T}$. Write $\tilde{p} = p_1 p_2$, where p_2 has no zeroes on $\mathbb{T}$. Let $p_1 = \prod(z - a_i)$. Define $p_3 = \prod(z - a_i(1+\epsilon))$, where $\epsilon > 0$. Let $N_1 = (p_3 p_2 \tilde{q})/(\tilde{r}q)$. Then $I(N_1) < -k$. Hence, we can chose $k+1$ quantities z_i for which N_1 is real and negative, and $k+1$ quantities z_i' such that N_1 is real and positive. Consider what happens when ϵ is small. If one z_i is different from all a_j, then N/N_1 has a limit, which is one, and $N(z_i)$ is real and negative.

Assume that the limit of z_i is a_j. Suppose that a_j is a root of multiplicity s of $\tilde{p}$. Write

$$N_1 = (z/a_j - 1 - \epsilon)^s N_2.$$

The limit for $\epsilon \to 0$ of N_2 is some non-zero number. Thus, near a_j, N_1 behaves like a polynomial of degree s. It is impossible that near a_j, N_1 can be real $2s+1$ times. Thus, it is impossible that the limit of z_i, z_{i+1}, ..., z_{i+s+1} is a_j (because the limit of z_u' would be a_j for $1 \leq j \leq s$). Hence, the number of z_i that has a_j as a limit is the degree of p_1. But the number of z_i exceeds the degree of p_1. $\square$

Lemma 20
Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two rational, strictly proper matrices of McMillan degree n_1 and n_2. Then $\mathcal{H}_1 + \mathcal{H}_2$ is of degree $\leq n_1 + n_2$. Equality holds in case $\mathcal{H}_1$ and $\mathcal{H}_2$ have no common pole.

Proof. Let (H_i, F_i, G_i) be a minimal realization of $\mathcal{H}_i$. Define

$$H = (H_1\ H_2), \quad F = \begin{pmatrix} F_1 & 0 \\ 0 & F_2 \end{pmatrix}, \quad G = \begin{pmatrix} G_1 \\ G_2 \end{pmatrix}.$$

Now, (H, F, G) is a realization of $\mathcal{H}_1 + \mathcal{H}_2$. If $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, and $Fx = \lambda x$, the assumption that $\mathcal{H}_1$ and $\mathcal{H}_2$ have no common pole says that λ is not an eigenvalue of both F_1 and F_2. So that, either $x_1 = 0$ and $F_2 x_2 = \lambda x_2$ or $x_2 = 0$ and $F_1 x_1 = \lambda x_1$. If $Hx = 0$, the observability of (H_i, F_i) implies then $x_1 = x_2 = 0$. In the same fashion, (F, G) is controllable. $\square$

Define

$$x_{vw\omega} = \frac{vw^*}{z - \omega}, \qquad y_{vw\omega} = \frac{vw^*}{z - \omega} + \frac{\overline{v}\overline{w^*}}{z - \overline{\omega}}. \tag{2.22}$$

Lemma 21
$\mathcal{H}$ has McMillan degree one if and only if it has the form $x_{vw\omega}$, where v and w are non-zero constant vectors, $\omega \in \mathbb{U}$. In case ω is not real, $y_{vw\omega}$ is of McMillan degree two.

Proof. Let

$$Q = I - (1 - \beta_\omega)vv^* \qquad C = \frac{1 - \overline{\omega}}{1 - \omega} \frac{vw^*}{1 - \overline{\omega}z}.$$

Then $Q^{-1}C = vw^*/(z-\omega)$ is of McMillan degree one. On the other hand, if $\mathcal{H}$ is of degree one, it can be written as $Q^{-1}C$, where Q has the form above. If $Q = D/\tilde{q}$, then $\tilde{q}C$ and $\tilde{D}C$ are polynomials of degree < 1, hence constant. The first condition says that $C = x/(1-\overline{\omega}z)$, for some constant x, the second that x must be vw^*. $\square$

Lemma 22
If $\mathcal{H}_0$ is of degree less than n, we can write $Q = Q_0Q_1$ and $C = Q_0C_1$, where Q_0 is of McMillan degree one and $Q_0(1) = I$. Moreover, $\psi(Q) = \psi(Q_1)$.

In the real case, the matrix Q_0 is real. It can be of McMillan degree two, and in this case, it has two complex conjugate poles (it is of the form (1.8.b)).

In the scalar case, we have $\mathcal{H}_0 = P/q$, where $q = q_0q_1$, $P = q_0P_1$, q_0 is of degree one (is irreducible in the real case).

Proof. The result is obvious in the scalar case. Consider now the non-scalar case. We know that $\mathcal{H}_0$ can be factored as $\mathcal{H}_0 = Q_2^{-1}C_2$, where Q_2 and C_2 are coprime. Thus $Q = Q_3Q_2$, where Q_3 is of positive degree. We can use the Potapov decomposition (corollary to lemma 4) and write $Q_3 = Q_0Q_4$, where Q_0 is of degree one, or two in case Q_3 is real and has no real poles. Take $Q_1 = Q_4Q_2$. Then $Q = Q_0Q_1$, and $C = Q_0C_1$ for some C_1.

Let $C_1' = L_F(Q_1)$. We have $\psi(Q_1) \le \|F - Q_1^{-1}C_1\|^2 = \psi(Q)$, and $\psi(Q) \le \|F - (Q_0Q_1)^{-1}(Q_0C_1')\|^2 = \psi(Q_1)$, so that $\psi(Q) = \psi(Q_1)$. $\square$

From now on, we shall assume that Q is a local minimum of ψ, $C = L_F(Q)$, $\mathcal{H}_0 = Q^{-1}C$, $F' = F - \mathcal{H}_0$, and we shall assume that Q is not coprime to C. Thus $\mathcal{H}_0$ is of degree less than n, and $\mathcal{H}_0 + x_{vw\omega}$ is of McMillan degree at most n. We consider the following set:

$$M = \{\, \mathcal{H} = x_{vw\omega}, \quad \|F'\| \le \|F' - \mathcal{H}\| \,\}. \tag{2.22}$$

If Q is the global minimum of ψ, then evert $x_{vw\omega}$ is in M. If Q is only a local minimum, this is no longer true, and one part of the proof consists of showing that M has a lot of elements. In the real case, if Q_0 is of McMillan degree two, we have to replace $x_{vw\omega}$ by $y_{vw\omega}$ in the definition of M.

The only interest of the following lemma is formula (2.24) that will be used later again.

Lemma 23
The theorem is true, if ψ is defined by the H^2 norm.

Proof. Consider

$$G\tilde{D}_1 = V_1q_1 + R_1 \tag{2.23.a}$$

$$V_1\tilde{D}_2 = V_2q_2 + R_2 \tag{2.23.b}$$

Assume $Q' = Q_2Q_1$, $D' = D_2D_1$ and $q' = q_2q_1$. Then $G\tilde{D}' = Vq' + R$, with $V = V_2$ and $R = R_2q_1 + R_1\tilde{D}_2$, hence $C = C_2 + Q_2C_1$. Equation (2.18) says

$$\psi(Q') = \psi(Q_1) - \|C_2\|^2 = \psi(Q_1) - \|R_2/q_2\|^2. \tag{2.24}$$

If Q_2 is near Q_0, then $\psi(Q') \ge \psi(Q)$ since Q' is near Q. This gives $\psi(Q_1) - \|C_2\|^2 \ge \psi(Q_1)$ hence $C_2 = 0$, and $R_2 = 0$. Now (2.23.b) says $V_1 = V_2Q_2$. Theorem 28 says $V_1 = 0$, hence $F = Q_1^{-1}C_1$. $\square$

Lemma 24
M contains every element of the form $x_{vw\omega}$, where v is proportional to $Q_1^{-1}(\omega)u$, u is near u_0, ω is near ω_0, and w arbitrary.

In the real case, if Q_0 is of McMillan degree two, M contains $y_{v'w\overline{\omega}}$, where v' is proportional to $Q_1^{-1}(\overline{\omega})u'$, u' is near some u_0', ω is near ω_0, and w arbitrary.

In the scalar case, the result is the same, but there is no constraint on v and w.

Proof. 1. Consider the scalar case first. We have $q = q_0 q_1$, $\mathcal{H}_0 = P_1/q_1$ for some P_1. Let $q_0 = z - \omega_0$, $q = z - \omega$. If ω is near ω_0, then

$$\mathcal{H}_1 = x_{vw\omega} = \frac{P}{q}, \qquad \mathcal{H}_0 + \mathcal{H}_1 = \frac{P'}{qq_1}$$

for some P and P'. We have

$$\|F'\|^2 = \psi(q_0 q_1) \le \psi(qq_1) \le \|F - P'/(qq_1)\|^2 = \|F' - \mathcal{H}_1\|^2.$$

Now $\mathcal{H}_1$ is in M because it is of McMillan degree one.

2. Consider now the non-scalar case. Assume

$$Q_2 Q_1 = Q_1' Q_2'. \tag{2.25}$$

Let $Q' = Q_2 Q_1$, $C' = Q_2 C_1 + Q_1' C_2'$. Then

$$Q'^{-1} C' = Q_1^{-1} C_1 + Q_2'^{-1} C_2'. \tag{2.26}$$

Let $\mathcal{H}_1 = Q_2'^{-1} C_2'$. If Q_2 is near Q_0, and $\mathcal{H}_1$ has the good McMillan degree, then $\mathcal{H}_1$ is in M.

3. The Potapov theorem says that

$$Q_0 = I - (1 - \beta_{\omega_0}) u_0 u_0^*. \tag{2.27.a}$$

Take

$$Q_2 = I - (1 - \beta_\omega) u u^*. \tag{2.27.b}$$

Consider

$$Q_2' = I - (1 - \beta_\omega) v v^*, \qquad C_2' = \frac{1 - \overline{\omega}}{1 - \omega} \frac{v w^*}{1 - \overline{\omega} z} \tag{2.28.a}$$

Since $Q_2'^{-1} C_2' = x_{vw\omega}$, it has McMillan degree one. Equation (2.25) is true provided that

$$v = \frac{Q_1^{-1}(\omega) u}{\|Q_1^{-1}(\omega) u\|} \tag{2.28.b}$$

according to lemma 4.

4. Assume now that we are in the real case, and Q_0 has McMillan degree two. Then

$$Q_0 = [I - (1 - \beta_{\omega_0}) u_0 u_0^*][I - (1 - \beta_{\overline{\omega}_0}) u_0' u_0'^*]. \tag{2.29.a}$$

Take

$$Q_2 = [I - (1 - \beta_\omega) u u^*][I - (1 - \beta_{\overline{\omega}}) u' u'^*]. \tag{2.29.b}$$

If u and u' are related by (1.8.c), then Q_2 is real. Since u_0 and u_0' are related by a similar equation, if ω is near ω_0, and u is near u_0, then Q_2 is near Q_0. Consider now

$$Q_2' = [I - (1 - \beta_{\overline{\omega}}) \overline{v v^*}][I - (1 - \beta_\omega) \overline{v' v'^*}]. \tag{2.29.c}$$

Write this as $Q_5 Q_6$, and Q_2 as $Q_3 Q_4$. Now (2.25) is $Q_3 Q_4 Q_1 = Q_1' Q_5 Q_6$. Now $Q_1' Q_5$ is inner provided that $\overline{v}'$ is parallel to $(Q_4 Q_1)^{-1} u$ (cf. (2.28.b)). But the condition Q_2 real says that u is parallel to $Q_4(\omega) \overline{u}'$. If we take complex conjugates, and remember that Q_1 is real, this gives

$$v' = \frac{Q_1^{-1}(\overline{\omega}) u'}{\|Q_1^{-1}(\overline{\omega}) u'\|}. \tag{2.30}$$

We have now

$$Q_3 Q_4 Q_1 Q_6^{-1} = Q_1' Q_5.$$

Since Q_2 is real, we can replace the first two factors by $\overline{Q}_3\overline{Q}_4$. Now Q_1' is inner, provided that $\overline{v}$ is parallel to $(\overline{Q_4}Q_1Q_6^{-1})^{-1}(\overline{\omega})\overline{u}'$. This condition simplifies to $\overline{v}$ parallel to $Q_6(\overline{\omega})v'$, which is the condition that says that Q_2' is real.

If we chose $C_2' = Q_5C_2''$, we get $Q_2'^{-1}C_2' = Q_6^{-1}C_2''$. As in the previous case, we can chose C_2' such that this is $\overline{v}'w^*/(z-\omega)$. If we take C_2' plus its complex conjugate, we get a real result, namely $y_{\overline{v}'w\omega} = y_{v'\overline{w\omega}}$. □

We make now some assumptions on $\|F\|$. One assumption we can make is that it is associated to a scalar product. For u near u_0 and ω near ω_0, we have

$$\|F'\| \le \|F' - \frac{vw^*}{z-\omega}\|, \qquad v = \frac{Q_1^{-1}(\omega)u}{\|Q_1^{-1}(\omega)u\|}.$$

This gives

$$\|F'\| \le \|F' - \lambda\frac{vw^*}{z-\omega}\|, \qquad \lambda \in \mathbb{C}, v = Q_1^{-1}(\omega)u. \tag{$*$}$$

This implies

$$\langle F' \mid \frac{vw^*}{z-\omega}\rangle = 0. \tag{$**$}$$

Let E be the set of all $Q_1(\omega)x_{vw\omega}$ which are orthogonal to F', for w and ω fixed. This is a vector space and holds every $vw^*/(z-\omega)$ for u near u_0, hence every $uw^*/(z-\omega)$. Thus, if ϵ_{ij} is the matrix which has one at position (i,j) and zero elsewhere, we have $\langle F' \mid \epsilon_{ij}/(z-\omega)\rangle = 0$. In the case where Q_0 is real, of McMillan two, we get $\langle F' \mid \epsilon_{ij}/(z-\omega) + \epsilon_{ij}/(z-\overline{\omega})\rangle = 0$.

Assume $F \in H_2^-$. We can write $\langle F' \mid f\epsilon_{ij}\rangle = \langle X_{ij} \mid f\rangle_2$, for every $f \in H_2^-$, where X_{ij} is some element of H_2^-, and $\langle u \mid v\rangle_2$ is the scalar product of H_2^-. We get $X_{ij}(/\overline{\omega}) = 0$, for ω near ω_0. This implies that X_{ij} is identically zero, so that $\langle F' \mid f\epsilon_{ij}\rangle$ is zero whatever f, i and j, hence F' is identically zero, and this proves the theorem. In the real case, if Q_0 is of degree two, we have $\langle X_{ij} \mid y + \overline{y}\rangle = 0$, if $y = (a+ib)/(z-\omega)$. Assume $\langle X_{ij} \mid 1/(z-\omega)\rangle = u+iv$. Since X_{ij} is real, we get $\Re(a+ib)(u+iv) = 0$ whatever a and b, hence $u+iv = 0$, and $F' = 0$.

There are other cases in which the theorem is true. Assume that F has coordinates F_{ij} in the canonical basis. Define

$$F_k = \sum \gamma_{ijk}F_{ij},$$
$$\|F\| = \left[\frac{1}{2\pi}\int_0^{2\pi} |F_k(e^{i\theta})^p d\theta\right]^{1/p}.$$

We assume $F \in H_p^-$, this implies that F_{ij} has a limit almost everywhere on $\mathbb{T}$, and this limit is in L^p. We assume that γ_{ijk} is defined on $\mathbb{T}$ and is in L^∞, so that F_k is defined almost everywhere on $\mathbb{T}$, and is in L^p. Moreover, we assume that if F_k is zero for each k, then F_{ij} is zero, so that F itself is zero.

We have the equivalent of $(*)$ and $(**)$: if for all λ we have $\|F - \lambda x\| \ge \|F\|$, then $I(F,x) = 0$, where $I(f,x)$ is a linear function of x. In fact

$$I(F,x) = \sum_k \frac{1}{2\pi}\int_0^{2\pi} |F_k(e^{i\theta})|^{p-2}\overline{F_k(e^{i\theta})}x(e^{i\theta})d\theta.$$

Note that, if $1/p + 1/q = 1$, $G_k = |F_k|^{p-2}\overline{F}_k$, then $G_k \in L^q$ and $I(F,x)$ is defined for $x \in L^p$. We have the same result as above: $I(F', \epsilon_k/(z-\omega)) = 0$ if ω is near ω_0. This condition is

$$\forall i,j \quad \sum_k \frac{1}{2\pi}\int_0^{2\pi} |F_k(e^{i\theta})|^{p-2}\overline{F_k(e^{i\theta})}\gamma_{ijk}(e^{i\theta})\frac{d\theta}{e^{i\theta}-\omega} = 0.$$

If we apply lemma 1, we find that $\sum |F_k|^{p-2}\overline{F}_k\gamma_{ijk}$ is analytic outside $\mathbb{U}$. Multiply this by F_{ij} and add. Since F_{ij} vanishes at infinity, we get a function that is analytic outside $\mathbb{U}$, and vanishes at infinity. But this function agrees on $\mathbb{T}$, almost everywhere, with the function $\sum |F_k|^{p-2}\overline{F_k}F_k = \sum |F_k|^p$. This function is real. Since it is the Poisson integral of its value on $\mathbb{T}$, it follows that it is analytic and real, hence is constant, hence zero. Thus $F_k' = 0$, and $F' = 0$, and the theorem is true in this case.

2.3.3 Scalar case of degree one

In this subsection, we consider the scalar case. For simplicity, we shall assume that we have one input, and one output. For this reason, we shall write g instead of G. We shall assume that $\|f\| = 1$. We examine what happens if q is of degree one, say $q = z - a$. By definition, we have

$$\psi(q) = 1 - (1 - |a|^2)|g(a)|^2.$$

If $g(z) = \sum g_k z^k$, the Cauchy-Schwarz inequality and the fact that g has unit norm say:

$$|\sum a^k g_k|^2 \le \sum |a^k|^2.$$

In the case $|a| < 1$, we deduce:

$$0 \le \psi(a) \le 1.$$

Obviously, if $|a| > 1$, we have $\psi(a) \ge 1$.

From the definition, we can deduce all partial derivatives. Assume $a = u + iv$. Then

$$\frac{\partial \psi}{\partial u} = 2u|g|^2 - 2(1 - |a|^2)\Re(g\overline{g}') \tag{2.31.1}$$

$$\frac{\partial \psi}{\partial v} = 2v|g|^2 - 2(1 - |a|^2)\Im(g\overline{g}') \tag{2.31.2}$$

$$\frac{\partial^2 \psi}{\partial u^2} = 2|g|^2 + 8u\Re(g\overline{g}') - 2(1 - |a|^2)[|g'|^2 + \Re(g\overline{g}'')] \tag{2.32.1}$$

$$\frac{\partial^2 \psi}{\partial u \partial v} = 4u\Im(g\overline{g}') + 4v\Re(g\overline{g}') - 2(1 - |a|^2)[\Im(g\overline{g}'')] \tag{2.32.2}$$

$$\frac{\partial^2 \psi}{\partial v^2} = 2|g|^2 + 8v\Im(g\overline{g}') - 2(1 - |a|^2)[|g'|^2 - \Re(g\overline{g}'')]. \tag{2.32.3}$$

We shall write ψ' instead of $\partial\psi/\partial u + i\partial\psi/\partial v$, and ψ'' will be the Hessian of ψ.

These formula simplify in the case $g(a) = 0$. In fact, $\psi' = 0$ and ψ'' is the identity matrix times $-2(1 - |a|^2)|g'(a)|^2$. If a is a multiple root of g, then $\psi' = \psi'' = 0$. The same happens if a is of module one. Otherwise, a is a critical point of ψ. This is a maximum if a is inside the disc, a minimum otherwise.

As a consequence, in the real case, ψ has at least one more local minimum than g has stable roots. In the complex case, this is not true. Note that, in any case, since $\psi(a) = 1$ on the circle, and is less than 1 inside, there is a local minimum inside the disc.

Assume that a is of module one. In that case, ψ' is $2a|g(a)|^2$. This means that ψ' is orthogonal to the unit circle and points outwards. This has as consequence that the solution of the differential equation (2.19) remains stable. The only problem that can happen is that the limit is a critical point that is not a minimum. We pretend that these are in general numerically unstable. We shall present here an example, in which the critical point is real. If the initial condition is real, then $q(t)$ will be real for every t, and, in this case, the algorithm fails to find a minimum. For this reason, if we are looking for a complex minimum, we never use a real initial condition.

Note that critical points are all a such that $g(a) = 0$ or $\overline{a}g(a) = (1 - |a|^2)g'(a)$. Whether it is a minimum or not depends on $g''(a)$. In the last chapter of this report, we explain how to solve directly the equation $\overline{a}g(a) = (1 - |a|^2)g'(a)$.

Let's study completely the case where g is of degree one. For simplicity, we replace the condition $\|g\| = 1$ by the condition g monic, hence $g(z) = z + \beta$. We assume g real, and consider $\psi(q)$, in the real and complex cases. Say $q = z - (u + iv) = z - a$. We have

$$\psi(u, v) = 1 + \beta^2 - (1 - u^2 - v^2)[(u + \beta)^2 + v^2]. \tag{2.33}$$

We shall assume $\beta \ne 0$, $\beta \ne \pm 1$. In fact, if $\beta = 0$, we have $\psi(a) = 1 - |a|^2 + |a|^4$. In this case, every point with $|a|^2 = 1/2$ is an absolute minimum. In the real case, this gives two minima with the same value, in the complex case, we are in a degenerate case. The case $\beta = 1$ is special, because for $v = 0$ and $u = -1$, ψ' is zero, but ψ is monotonic near $u = -1$.

Let $A = 1 - u^2 - v^2$ and $B = (u+\beta)^2 + v^2$. Then the partial derivatives of ψ are

$$\psi_u = -2(-Bu + (u+\beta)A), \qquad \psi_v = 2(B - A)v.$$

Consider a point where the derivative of ψ is zero. In case $v \neq 0$, $\psi_v = 0$ implies $A = B$, hence $\psi_u = 0$ implies $A\beta = 0$. Then we get $A = B = 0$, which is absurd. This means that all critical points of ψ are real. These points are defined by

$$(u+\beta)^2 u = (u+\beta)(1-u^2),$$

in other words

$$u = -\beta, \qquad u = \frac{-\beta \pm \sqrt{\beta^2+8}}{4}. \tag{2.34}$$

Note that, for the last two roots, we have $2u^2 + \beta u - 1 = 0$. Hence, if $|\beta| < 1$, we have three critical points in the interval $[-1, 1]$, and otherwise a single one.

If we consider the second derivative, if $\psi_{uu} = (u+\beta)^2 + 4u(u+\beta) - (1-u^2)$ and $\psi_{vv} = (u+\beta)^2 - (1-u^2)$, then the second derivative of ψ in the real case is $2\psi_{uu}$, while in the complex case it is

$$\psi'' = 2\begin{pmatrix} \psi_{uu} & 0 \\ 0 & \psi_{vv} \end{pmatrix}.$$

Note that if $u = -\beta$, we get $\psi'' = -2(1-u^2)I$, and this is clearly a local maximum (assuming of course $-1 < u < 1$). Assume now $2u^2 + \beta u - 1 = 0$. Then

$$\psi'' = 2\frac{1-u^2}{u^2}\begin{pmatrix} 1+2u^2 & 0 \\ 0 & 1-2u^2 \end{pmatrix}.$$

Since the product of the roots is $-1/2$, we have a positive and a negative root. Since $1 - 2u^2 = \beta u$, for one of the roots $1 - 2u^2$ is positive, and for the other one, it is negative.

Hence, the situation is the following. If $|\beta| > 1$, there is one critical point. This critical point is a minimum. If $|\beta| < 1$, there are three critical points. The point $u = -\beta$ is a maximum, the point $1 - 2u^2 = \beta u$ with $\beta u > 0$ is a minimum. The other point is a local minimum in the real case, a saddle point in the complex case.

Lemma 25

There are cases where g is real, ψ has a unique real critical point (which is a minimum), but this minimum is not a complex minimum.

What we shall do is consider $g(z) = z^2 + az + b$, and find the conditions on a and b. We shall see that this set of conditions is not empty, this will prove the lemma.

We have, for $q = z - u$,

$$\psi(q) = 1 + a^2 + b^2 - (1-u^2)(u^2 + au + b)^2 \tag{2.35.a}$$

and

$$\psi'(q) = 2(u^2 + au + b)(3u^3 + 2au^2 + ub - 2u - a). \tag{2.35.b}$$

We know that if q has a root in $[-1, 1]$, then ψ has a maximum on $[-1, 1]$. Hence, we are looking for the conditions under which the first factor of (2.35.b) has no root in $[-1, 1]$, while the second factor has a single one.

Lemma 26

The polynomial $g(z) = z^2 + az + b$ has no root in $[-1, +1]$ if and only if $|b+1| > |a|$, and, if $|a| \leq 2$, then $b > a^2/4$ or $b < -1$.

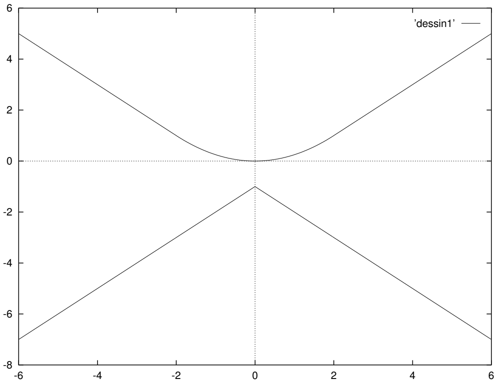

Figure 2.2: Set of points a and b such that $g = z^2 + az + b$ has no root on the interval $[-1, 1]$. If b is between the two curves, there is such a root. If b is above the upper curve, the polynomial is positive on $[-1, 1]$, if b is below the other one, the polynomial is negative.

Note that this condition becomes $b < f_1(a)$ or $b > f_2(a)$, for some functions f_1 and f_2. These functions are plotted on figure 2.2.

Proof. The condition $|b+1| \le |a|$ is $g(1)g(-1) \le 0$. If it is true, then g has a root in $[-1, 1]$ and a root outside. If it is false, then g has zero or two roots in $[-1, 1]$. In the case $|a| \ge 2$, the relation $|b+1| > |a|$ implies $|b| > 1$, so that g has a root α with $|\alpha| > 1$. This means that g cannot have two roots in $[-1, 1]$. Assume now $|a| \le 2$. If $b < -1$, then g has also a root with $|\alpha| > 1$; if $b > a^2/4$ then g has no real roots; finally if $-1 < b < a^2/4$, g has two real roots, the product is less than one in absolute value, thus one root lies in $[-1, 1]$. $\square$

Let $h(u) = 3u^3 + 2au^2 + ub - 2u - a$. Note that $h(1)h(-1) = a^2 - (b+1)^2$. This means that, if g has no root in $[-1, 1]$, then h has an odd number of roots in $[-1, 1]$. Let c be a root of h. We have

$$h(u) = (u-c)(3u^2 + (3c+2a)u + 3c^2 + 2ac + b - 2).$$

Assume $c = 0$. Then $h(c) = 0$ gives $a = 0$. It is then obvious that ψ has a unique critical point in $[-1, 1]$ if and only if $b > 2$ or $b < -1$. Note that replacing a by $-a$ replaces c by $-c$, so that we may assume $c > 0$. We have

$$h(u) = 3(u-c)(u^2 + (c + 2a/3)u + a/(3c)).$$

Let $d = a/(3c)$. The last factor of h is

$$f(u) = u^2 + c(1+2d)u + d.$$

We are looking for a condition on c and d for which this polynomial has no roots in $[-1, 1]$.

Lemma 27
In case $0 < c < 1$, the polynomial $f(z) = z^2 + c(1+2d)z + d$ has no roots in $[-1, 1]$ if and only if

$$\left(\frac{1-2d}{1+2d}\right)^2 < 1 - 2c^2 \tag{2.36.a}$$

if $0 \le d \le 1$ and

$$\frac{1-2c}{c-1} < \frac{1}{d} \tag{2.36.b}$$

otherwise.

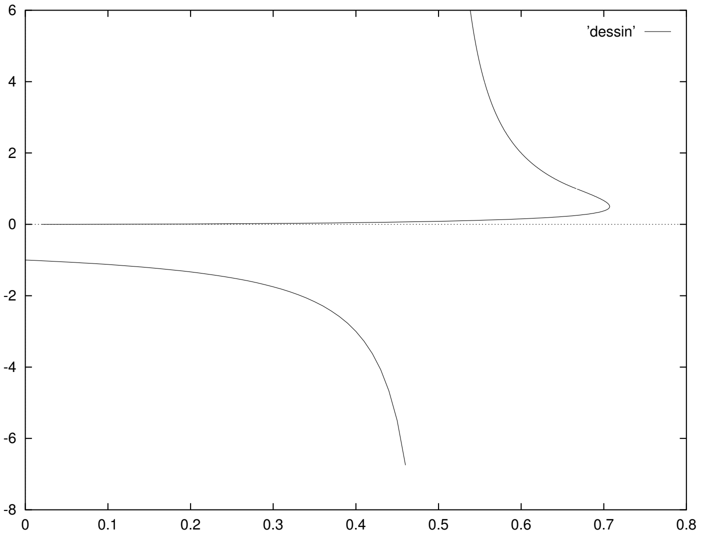

Figure 2.3: Coefficients c, d, for which ψ has a unique critical point. The two curves are defined by (2.36). If c (horizontally) is to the left of the curve, ψ has a unique real critical point.

This condition has the form $0 < c < c_1(d)$, for some function $c_1(d)$, which is zero on $[-1, 0]$ and non-zero elsewhere. See figure 2.3.

Proof. We just apply the previous lemma with $a = c(1 + 2d)$ and $b = d$.

Note that the lemma says: $|b| > 1$ and $|b + 1| > |a|$ or $0 \leq b \leq 1$ and $b > a^2/4$. Condition (2.36.a) is just $b > a^2/4$.

Condition $|b + 1| > |a|$ is $|d + 1| > c|1 + 2d|$. In case $d \geq 0$, it is $d(1 - 2c) \geq c - 1$, in case $d \leq -1$ it is $d(1 - 2c) \leq c - 1$. In both cases it is (2.36.b) because $c < 1$. Note that $(1 - 2c)/(c - 1) > -1$, so that (2.36.b) excludes the case $-1 \leq d \leq 0$. □

Note: in case $d < 0$, we get $c \leq 1/2$, and in case $d > 0$, we get $c \leq \sqrt{2}/2$. This gives the strange condition: if ψ has a critical point c with $|c| > \sqrt{2}/2$, then ψ has at least another critical point.

Note also that if the conditions of the lemma are true, then g has no root in $[-1, 1]$.

What we want to find is a critical point of ψ, which is the absolute real minimum, but is not a complex minimum.

Consider now what happens if $q = z - (u + iv)$ is complex. We have

$$\psi = 1 + a^2 + b^2 - (1 - u^2 - v^2)[(u^2 + au + b - v^2)^2 + (2u + a)^2 v^2]. \tag{2.37}$$

If we differentiate ψ with respect to v we get

$$\psi'_v = 2vA$$

with

$$A = (u^2 + au + b - v^2)^2 + (2u + a)^2 v^2 - (1 - u^2 - v^2)[(2u + a)^2 - 2(u^2 + au + b - v^2)].$$

The equation for the critical points is $A = 0$ or $v = 0$, together with the equation $\psi'_u = 0$. In the case $v = 0$, this is equation (2.35.b):

$$(u^2 + au + b)(3u^3 + 2au^2 + ub - 2u - a) = 0.$$

Let $\psi_1(u) = \psi(u, 0)$. At a point where $v = 0$, the second derivative is $\begin{pmatrix} C & 0 \\ 0 & A \end{pmatrix}$, where C is the second derivative of ψ_1. If u is a local minimum of ψ_1, then $C > 0$. This means that, if $A > 0$, then $(u, 0)$ is a local minimum of ψ, and if $A < 0$, this is a saddle point.

Now, if $v = 0$ we have

$$A = (u^2 + au + b)^2 - (1 - u^2)[(2u + a)^2 - 2(u^2 + au + b)].$$

In case $\psi_1' = 0$, we have $u = c$, $(1 - c^2)(2c + a) = c(c^2 + ac + b)$, hence

$$A = (c^2 + ac + b)(-3c^2 + b + 2).$$

The assumption is that g has constant sign on $[-1, 1]$. The sign is that of $g(0) = b$. Hence the condition $A < 0$ is $b(-3c^2 + b + 2) < 0$. Since we know $c \leq \sqrt{2}/2$, we have $2 - 3c^2 > 0$. Hence the condition is $3c^2 - 2 < b < 0$.

Recall that

$$3c^2 + 6dc^2 + b - 2 - 3d = 0.$$

The conditions are

$$6c^2(1 + d) < 4 + 3d \qquad 2 + 3d < 3c^2(1 + 2d).$$

These conditions can be rewritten as

$$(6c^2 - 3)(1 + d) \leq 1 \leq (6c^2 - 3)(1 + 2d).$$

Since $c \leq \sqrt{2}/2$, it implies $1+2d \leq 1+d$, hence $d < 0$. Hence (2.36) says $d < -1$, and $c \leq (d+1)/(2d+1)$ (this is (2.36.b)). This relation implies $1 \leq (6c^2 - 3)(1 + 2d)$ so that we are left with (see figure 2.4)

$$c^2 \geq \frac{4 + 3d}{6 + 6d} \qquad c \leq \frac{d + 1}{2d + 1} \qquad d < -1.$$

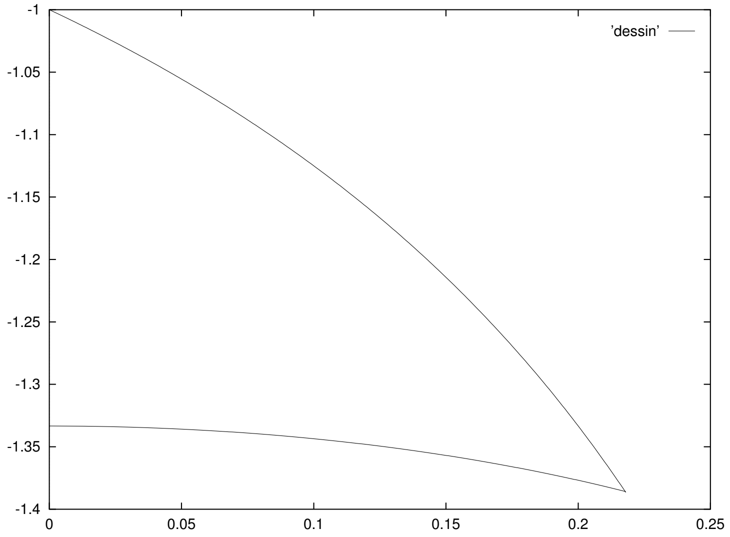

Figure 2.4: Conditions on which ψ has a unique real minimum, which is a complex saddle point. See the text for the notations.

If we go back to parameters a and b, it is easy to see that the second equation is $b + 1 < |a|$. This says that g has two real roots, with different sign, outside $[-1, 1]$. See figure 2.5. The other condition is

$$a^2 < 12b^2 \frac{b + 2}{(1 + 2b)^2}.$$

Example: Take $a = -0.35$, $b = -1.46$. On figure 2.6, we show the value of $\psi(u, v)$ for $v = 0$ and $v = 0.42$. The real minimum is obtained for $c = 1/10$.

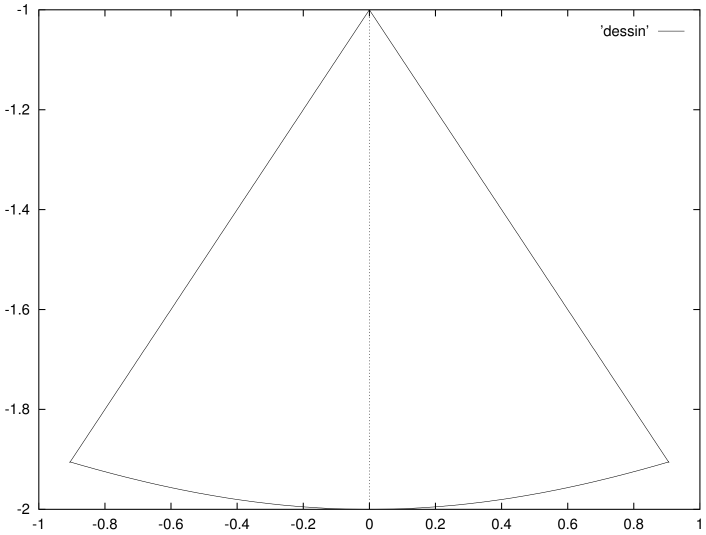

Figure 2.5: Set of points a and b such that ψ_G has a unique real critical point, which is a saddle point, at degree one, for $G = z^2 + az + b$. Between the two straight lines, G is negative on $[-1, 1]$.

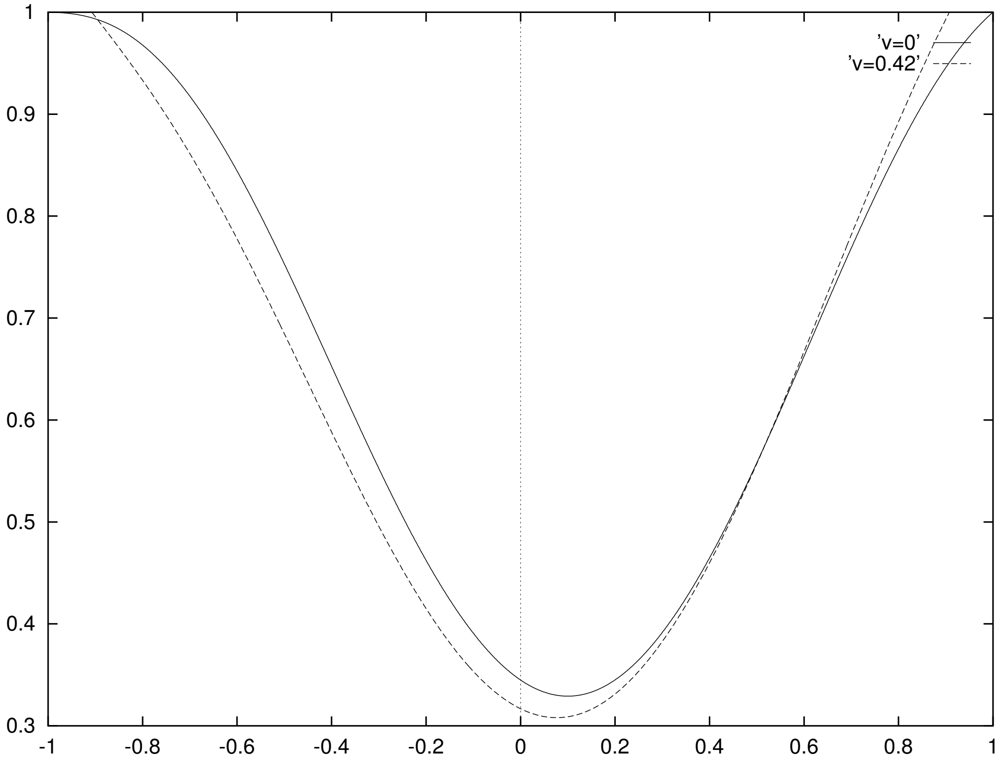

Figure 2.6: Values of $\psi_G(q)$, for $q = z - (u + iv)$, as a function of u for $v = 0$ and $v = 0.42$. We have $G = z^2 - 0.35z - 1.46$.

Stable polynomials The problem, in the scalar case, is to minimise $\psi(q)$ for stable q. In the case of degree 2, if $q = u + vz + z^2$, and q has real coefficients, then the set of stable polynomials is a triangle. In the complex case, the result is the following.

Lemma 28
The polynomial $q = u + vz + z^2$ is stable if and only if

$$|u|^2 + 1 > |\frac{v^2}{2} - 2u| + |\frac{v^2}{2}| \tag{2.38}$$

and $|u| \le 1$.

Proof. Write $u = a + ib$, $v = c + id$. The resultant of q and $\tilde q$ is

$$R = (a^2 + b^2 - 1)^2 - (a^2 + b^2)(c^2 + d^2) + 2a(c^2 - d^2) + 4bcd - c^2 - d^2.$$

Note that $R = (|u|^2 - 1)^2 - |u\overline{v} - v|^2$, and this leads to a factorisation of R. However, if

$$R_\pm = |u|^2 + 1 - |\frac{v^2}{2}| \pm |\frac{v^2}{2} - 2u|$$

then $R = R_+R_-$. Set $y = v^2/2$. Then $R_+ = |u|^2 + 1 - |y| + |y - 2u|$. The triangular inequality $|y| \le |y-2u|+|2u|$ implies that $|R_+| \ge (|u|-1)^2$, so that R_+ vanishes only if $|u| = 1$ and $|y| = |y-2u|+|2u|$. This last condition says that y/u is real and at least 2. Now, these equations are equivalent to say that q has two roots α and β with $\alpha\overline{\beta} = 1$.

Hence $R_- = 0$ is the condition under which q has a root of modulus one. Let $q_\lambda = \lambda^2 u + \lambda v + z^2$. If z is a root of q_λ, then z/λ is a root of q.

Define $f(\lambda)$ to be $R_-(\lambda^2 u, \lambda v)$. If $w = |v^2/2 - 2u| + |v^2/2|$, then $f(\lambda) = \lambda^4|u^2| - \lambda^2 w + 1 = g(\lambda^2)$. The triangular inequality says that $|w| \ge |2u|$, so that there are two real numbers a and b such that $g(a) = g(b) = 0$. Note that $ab = 1/|u|^2$ and $a + b > 0$. Hence a and b are positive. Then q has two roots, with modules $1/\sqrt{a}$ and $1/\sqrt{b}$. The condition q stable is now $a > 1$ and $b > 1$. In particular, 1 is not between a and b, hence $f(1) > 0$. Obviously q stable implies $|u| < 1$. On the other hand, if $|u| < 1$, then $ab = 1/|u|^2$ says $ab > 1$. If 1 is not between a and b, then $1 < a < b$. □

This lemma is not used in HYPERION: it is much easier to factor q, and check the roots (we have explicit formulas in the case of degree 2).

Note: assume q real. If $u \ge v^2/4$, the condition becomes $(u - 1)^2 > 0$, hence $u \ne 1$. Otherwise, it is $(u + 1)^2 > v^2$. Since we have $-1 \le u \le 1$, the set defined by the lemma is the triangle defined by $u < 1$, $v < u + 1$ and $-v < u + 1$ (see figure 2.7 on page 51).

2.3.4 Boundary conditions

Let's assume that we are in the scalar case. Hence, we can consider polynomials q which have roots outside $\mathbb{U}$.

One can show (see [2]) that there exists a C^∞ function, defined for all polynomials q, that is equal to $\psi(q)$ for q stable. In the case where F has only a finite number of Fourier coefficients, this is obvious, because ψ is a polynomial function with respect to the entries of F and q.

Theorem 14
Let $\psi_1(q)$ be the minimum of $\|F - P/q\|^2$. It is possible to define a C^∞ function ψ such that $\psi(q) = \psi_1(q)$, whenever q is stable. This function is uniquely defined as $\psi(q) = \psi(q_1)$ in the case where $q = q_1q_2$, q_1 is stable and q_2 has all its roots on $\mathbb{T}$.

Proof. For the first claim, see [2]. In the case where F has only a finite number of Fourier coefficients, the result is obvious, since $\psi(q)$ is a polynomial function of q.

Assume now $q = q_1 q_2$, where q_1 is stable, and roots of q_2 are outside $\mathbb{U}$. If P and S are polynomials, since q_1 and q_2 are coprime, we can write

$$\frac{P}{q} = \frac{P_1}{q_1} + \frac{P_2}{q_2} \qquad \frac{S}{q} = \frac{S_1}{q_1} + \frac{S_2}{q_2}.$$

Then

$$\langle F - \frac{P}{q} \mid \frac{S}{q} \rangle = \langle F - \frac{P_1}{q_1} \mid \frac{S_1}{q_1} \rangle - \langle \frac{P_2}{q_2} \mid \frac{S_2}{q_2} \rangle.$$

Since P/q is a minimum if and only if this vanishes for every S_1 and S_2, it follows that P_2 must be zero, and P_1 must be $L(q_1)$. This means that $\psi_1(q) = \psi(q_1)$. We get $\psi(q) = \psi(q_1)$ by continuity. □

Consider equations (2.23) in the scalar case.

$$G\tilde{q}_1 = V_1 q_1 + R_1$$

$$V_1 \tilde{q}_2 = V_2 q_2 + R_2$$

$$\psi(q) = \psi(q_1) - \|R_2/q_2\|^2. \tag{2.39.a}$$

The last equation was established under the condition that q_2 is stable. Assume that all roots are outside $\mathbb{U}$. Let $q_3 = \tilde{q}_2$. Then

$$V_2 \tilde{q}_3 = V_1 q_3 - R_2.$$

This gives

$$\psi(q) = \psi(q_1) + \|R_2/\tilde{q}_2\|^2. \tag{2.39.b}$$

Note that $\psi(q) \geq \psi_1(q)$: our algorithm does not chose the best numerator in this case. In fact, $\psi_1(q)$ is not C^1 on the boundary. The difference between (2.39.a) and (2.39.b) can be explained in the following way. When we write $V_1\tilde{q}_2 = V_2 q_2 + R_2$, we consider the division in H^2. But if q_2 has its roots outside $\mathbb{U}$, then the remainder R_2 is zero. In the case of (2.39.b), we consider the division of polynomials. Here we get a non-zero remainder.

In the special case where q_2 is of degree one, say $q_2 = z - \alpha$, we get

$$\psi(q) = \|V_1\|^2 + (|\alpha|^2 - 1)|V_1(\alpha)|^2. \tag{2.40}$$

This equation is valid, whether or not α is in $\mathbb{U}$. In the case where q_2 is of degree two, and has two different roots, a similar formula can be written (see [2, lemma 4.5]).

Another question is the following. Assume that q has some roots on the boundary, and let $q' = q - \epsilon\nabla\psi$. Then $\psi(q') < \psi(q)$ for small positive ϵ. The question is now: is q' stable? The answer is true if q has one root on $\mathbb{T}$, but may be false otherwise. We consider here a simple example:

$$g = z^2 + t, \qquad q = z^2 + uz + v.$$

We have

$$\psi = |v|^2 + |\overline{u} - \overline{v}u|^2 + |1 + \overline{v}t - |v|^2 - u(\overline{u} - \overline{v}u)|^2.$$

Write $u = a + ib$, $v = c + id$. If we assume that t is real, and define

$$A_1 = a - ac - bd, \qquad A_2 = -b - bc + ad$$

then

$$\psi = c^2 + d^2 + A_1^2 + A_2^2 + (1 - ct - c^2 - d^2 - aA_1 + bA_2)^2 + (-dt - aA_2 - bA_1)^2.$$

Let's compute the derivative of ψ, assuming that q has two roots of modulus one. This implies $|v| = 1$ and $\overline{u} = \overline{v}u$. In other words, $A_1 = A_2 = 0$, so that the derivative of $A_1^2 + A_2^2$ is zero. We have

$$\frac{1}{2}\frac{\partial\psi}{\partial a} = ct(bd - a(1-c)) - dt(-ad - b(1-c))$$

$$\frac{1}{2}\frac{\partial\psi}{\partial b} = ct(ad - b(1+c)) - dt(a(1+c)+bd)$$

$$\frac{1}{2}\frac{\partial\psi}{\partial c} = c + ct(t - 2c + a^2 - b^2) - dt(2ab)$$

$$\frac{1}{2}\frac{\partial\psi}{\partial d} = d + ct(-2d + 2ab) - dt(-t - a^2 + b^2).$$

We can simplify a bit these formulas. In effect, the coefficient of t in $\partial\psi/\partial a$ is $bd - ac + a(c^2+d^2)$. But $c^2 + d^2 = 1$ and $a = ac + bd$. Hence we get

$$\frac{1}{2}\left(\frac{\partial\psi}{\partial a} + i\frac{\partial\psi}{\partial b}\right) = tu(\overline{v} - v)$$

$$\frac{1}{2}\left(\frac{\partial\psi}{\partial c} + i\frac{\partial\psi}{\partial d}\right) = v[1 + t^2 + t(u^2 - v - \overline{v})].$$

The gradient of ψ is defined, if q has degree two, as

$$\nabla\psi = \left(\frac{\partial\psi}{\partial a} + i\frac{\partial\psi}{\partial b}\right)z + \frac{\partial\psi}{\partial c} + i\frac{\partial\psi}{\partial d}.$$

Let

$$q(\lambda) = q - \lambda\nabla\psi.$$

If we solve the differential equation $dq/dt = -\nabla\psi$, the solution for small t will be $q(t)$. So the question is: is $q(t)$ stable? (we write here $q(\lambda)$ because t appears in g in the example above). In the case where $\nabla\psi = Az + B$, the root of $q(\lambda)$ near α is

$$z = \alpha[1 + \gamma\lambda + o(\lambda)]$$

$$\gamma = \frac{A\alpha + B}{\alpha(\alpha - \beta)} = \frac{A + B\overline{\alpha}}{\alpha - \beta}, \tag{2.41}$$

where α is a root of $q(0)$. This root is stable if the real part of γ is negative.

Assume now that, for small positive λ, $q(\lambda)$ has two unstable roots. Now $\Re\gamma > 0$. The same is true if we exchange α and β in (2.41). If we take the sum, and use $v = (\overline{\alpha} - \overline{\beta})/(\beta - \alpha)$, we get $\Re B\overline{v} < 0$.

For the example above, we get

$$\Re[1 + t^2 + t(u^2 - v - \overline{v})] < 0$$

which is absurd, because $|u^2 - v - \overline{v}| \le 2$. Said otherwise, at least one root of $q(\lambda)$ is stable. However, some roots can be unstable. For instance, if $\alpha = 1$ and $\beta = i$, we have

$$\gamma = -t^2 - 6t - 1 + i(t-1)^2.$$

Thus, the root near 1 is unstable if $t^2 + 6t + 1 < 0$. This happens if t is near -1 (note that, for $t = -1$, $\alpha = 1$ is a root of g). In fact, the root is unstable if t is between the two roots of the equation $t^2+6t+1 = 0$, namely $-3 \pm \sqrt{8}$.

Consider now the general case. With the notations of (2.40),

$$\frac{\partial\psi}{\partial\alpha} = 2\alpha|V_1(\alpha)|^2.$$

Assume that all roots of q are in $\mathbb{T}$. Then V_1 is a constant time G so that

$$\frac{\partial\psi}{\partial\alpha} = 2\alpha|G(\alpha)|^2. \tag{2.42}$$

This formula allows us to find the derivatives of ψ with respect to the coefficients of q. For simplicity, we consider only the case where q has two roots α and β. Let $u = -(\alpha+\beta)$ and $v = \alpha\beta$. Then

$$A = \frac{1}{2}\frac{\partial\psi}{\partial u} = \frac{|G(\alpha)|^2 - |G(\beta)|^2}{\overline{\beta} - \overline{\alpha}}, \tag{2.43.a}$$

$$B = \frac{1}{2}\frac{\partial\psi}{\partial v} = \frac{\alpha|G(\alpha)|^2 - \beta|G(\beta)|^2}{\overline{\beta} - \overline{\alpha}}, \tag{2.43.b}$$

and (2.41) gives

$$\gamma = \frac{|G(\alpha)|^2 - |G(\beta)|^2 + (\alpha|G(\alpha)|^2 - \beta|G(\beta)|^2)\overline{\alpha}}{-|\beta-\alpha|^2}.$$

Let $A_1 = |G(\alpha)|^2$ and $A_2 = |G(\beta)|^2$. Then

$$\gamma = \frac{-2A_1 + (1+\alpha\overline{\beta})A_2}{|\beta-\alpha|^2}.$$

In the special case where $1+\alpha\overline{\beta} = 0$, we have $\Re\gamma < 0$ (we exclude the case where $G(\alpha) = 0$, this is a degenerate case). The condition $1+\alpha\overline{\beta} = 0$ is $\alpha = -\beta$. Thus, for $q = z^2 - 1$, $q(\lambda)$ is stable.

Let $x = \Re(1+\alpha\overline{\beta})$. We have $0 \le x \le 2$. The root near α is unstable if

$$-2A_1 + xA_2 > 0.$$

Note that this condition is false if $A_1 = A_2$. In particular, this is the case if q has two complex conjugate roots. The root near β is unstable if

$$-2A_2 + xA_1 > 0.$$

If both roots are unstable, by addition, we get $x > 2$, which is absurd.

Theorem 15

Consider $q(t)$, the solution of the differential equation

$$\frac{dq}{dt} = -\nabla\psi(q(t)).$$

Assume that $q(t_0) = q_1q_2$ where q_1 is stable and q_2 has all its roots on $\mathbb{T}$. Assume that we are in a generic case.

If q_2 is of degree one, and q_1 is a critical point of ψ, then $q(t)$ is stable for small positive $t - t_0$. On the other hand, if $q(t)$ is stable for small negative $t - t_0$, then q_2 is not a critical point of ψ.

Proof. 1. If $t - t_0$ is small, we can factor $q(t) = q_1(t)q_2(t)$, where $q_1(t)$ and $q_2(t)$ are continuous and $q_1(t_0) = q_1$, $q_2(t_0) = q_2$. The polynomial $q_1(t)$ will be stable for small $t - t_0$.

If $\lambda = t - t_0$, then $q(t)$ is near $q - \lambda\nabla\psi$, where $\nabla\psi$ is the gradient of ψ at $q(t_0)$.

2. Assume now that q_1 is a critical point of ψ. Equation (2.39) says $\psi(q(t)) = \psi(q_1(t_0)) + o(t-t_0) + f(t)$, where $f(t)$ is positive if $q_2(t)$ has all its roots out of $\mathbb{U}$, negative if $q_2(t)$ has all its roots in $\mathbb{U}$. The main trick now is that $\psi(t)$ is decreasing.

Let $\alpha(t)$ be a root of q_2. Since we are in a generic case, we may assume that $\alpha(t_0)$ is a root of multiplicity one. If this is not the case, it will in general happen that the two roots split, and we get a stable and an unstable root. [Consider the function $1 - |\alpha(t)|^2$. This function is zero at $t = t_0$. We assume that its derivative is not zero. If this is not the case, in general we have an unstable root for $t < t_0$ and $t > t_0$ (consider $\alpha(t) = 1 + it$).]

Thus, the assumption we make is just that $f(t)$ has a non-zero derivative at $t = t_0$, so is $f(t) = A(t - t_0) + o(t - t_0)$, with $A \neq 0$. We have $A < 0$ if q_1 has only stable roots for $t > t_0$, $A > 0$ if q_2 has only unstable roots.

3. Assume now that q_2 is stable for $t < t_0$. This gives $A > 0$, and contradicts the fact that ψ is decreasing. This shows that q_1 cannot be a critical point of ψ.

4. The same argument shows that q_2 must have at least one stable root for $t > t_0$. The example above shows that it can have some unstable roots. Thus, let's assume that it is of degree one. Then it must be stable, at least generically. In fact, a straight-forward computation says that the root is

$$\alpha(t) = \alpha + 2(t - t_0)|V_1(\alpha)|^2 \frac{n}{|q_1(\alpha)|^2} + o(t)$$

where $\alpha = \alpha(t_0)$. Thus $q(t)$ is stable, provided that $V_1(\alpha) \neq 0$. If this condition fails, then q is a critical point of ψ. □

What this theorem says is the following. Assume that q_0 is stable. Consider the solution of the differential equation, and t_0 such that $q(t)$ is stable for $t < t_0$, and not stable at t_0. If no such t_0 exists, then the limit $q(\infty)$ is a critical point of ψ. Write $q(t_0) = q_1 q_2$ where q_1 is stable, and q_2 is not. Then all roots of q_2 are in $\mathbb{U}$. Thus, q_1 is not a critical point of ψ. We can consider the differential equation with q_1 as initial condition. We then find q_1' such that $\psi(q_1') < \psi(q_1)$ (if we are lucky, q_1' is just the solution of the differential equation at $t = \infty$). This q_1' is a critical point of ψ (in the generic case, it is a local minimum, if we are lucky, it is the global minimum).

Let's take $q_1' q_2$ as initial condition. What the theorem says now is that for small positive t, $q(t)$ will be stable. Our example shows that this may be false in the case where q_2 has degree ≥ 2. Note that generically, only one root will be of modulus one (unless we are in the real case, and we have two complex conjugate roots). Hence, the algorithm will be: consider $q_1'(z - \alpha)$ as initial condition. This will give q_1''. Then consider $q_1''(z - \beta)$, and iterate until we find a polynomial of degree n. This will come to an end, because ψ is strictly decreasing.

Note: the question is how to chose α. What we need is $V_1(\alpha) \neq 0$. This is always possible, because, if V_1 is identically zero, then F, the quantity to approximate at degree n, is rational, of lesser degree. There is however a small problem: it can happen that we are looking for a real solution and V_1 vanishes at ± 1.

Consider for instance $g(z) = t + z^2 - z^4$, with $t = 8$. Let's compute the critical points at order one. If $q = z - \alpha$ is a critical point, then α is a root of a polynomial of degree 9. Note that g does not vanish in the interval $[-1, 1]$, and this will exclude 4 values. It remains $\alpha = 0$ and $5\alpha^4 - 7\alpha^2 - 6$. This last equation has a negative root in α^2, and $\alpha^2 = 2$. Thus, $q = z$ is the only critical root of ψ at order one. Now, we have $V_1 = z(1 - z^2)$, which vanishes at ± 1. (Note that if $g(z) = t + z(1 - z^2)f(z)$, V_1 vanishes at ± 1; $z = 0$ is a critical point if g is an even function, the simplest case is when $f(z) = z$, we adjust t such that $z = 0$ is the unique critical point).

Thus, it is impossible to find the minimum at order 2 with the method explained above. Consider $q = z^2 + uz + v$. Then

$$V = -z^4 v + (-u + vu)\, z^3 + \left(v - 1 + v^2 + u^2 - vu^2\right) z^2 + \left(2\,u - 2\,v^2 u - u^3 + vu^3\right) z +$$

$$+ tv + 1 - v^2 + v - v^3 - vu^2 + 3\,v^2 u^2 - 2\,u^2 + u^4 - vu^4.$$

We have

$$\psi = 2 + t^2 + \psi_2,$$

$$\psi_2 = (v^2 - 1)(t - v - v^2)^2 + u^2(1 - v)B,$$

$$B = u^6 - vu^6 + 6\,v^2 u^4 - 3\,u^4 - 3\,vu^4 - 11\,v^3 u^2 - v^2 u^2 + 9\,vu^2 + 2\,tvu^2 + 3\,u^2 + 6\,v^4 + 6\,v^3 - 4\,v^2 - 6\,tv^2 - 4\,tv - 5\,v - 1.$$

If we ask Maple for the critical points of ψ, we get a list of 7 solutions.

- $u = 0$, two values for v.
- $u = 0$, three values for v.
- $v = 1$, four values for u.

- $v = \pm\sqrt{-t}$, four values for u.
- $P_4(v) = 0$, $u = \pm\sqrt{f(v)}$, where P_4 is a polynomial of degree four.
- $P_6(v) = 0$, $u = \pm\sqrt{f(v)}$, where P_6 is a polynomial of degree six.
- $v = 0$, $u = \pm 1$.

This gives a total of 35 solutions. Of course, what we want is q real and stable. In the special case $t = 8$, Maple can factor P_6. The critical points are

- $u = 0$, $v^2 + v - 8 = 0$; numerically, $v = -3.372281323$ and $v = 2.372281323$.
- $u = 0$, $3v^3 + 2v^2 - 10v - 1 = 0$, i.e. $v = -2.146944380$, $v = -.09835082394$ and $v = 1.578628537$.
- $v = 1$, $8u^4 - 39u^2 - 36 = 0$. This gives $u = \pm 2.380927653$ (and two complex roots).
- $v = i\sqrt{8}$, and u is a function of v.
- $v^3 + 3v^2 + 6v + 14 = 0$, $2u^2 - 4 - 4v - v^2 = 0$. There is a unique real solution for v, namely $v = -2.698885490$. We get $u = \pm 0.4941866692$.
- $w^3 - 11w^2 + 72w - 408 = 0$, $v = w/5$, $15u^2 + 9 - 6w + w^2 = 0$. There is no real solution.
- $60v^4 - 104v^3 + 151v^2 - 152v + 54 = 0$, $P(u) = 0$. No real solution.
- $v = 0$, $u = \pm 1$.

Thus, we have 11 real solutions, and only one stable solution, $u = 0$, $v = -.09835082394$.

Note that in general, computer algebra systems are unable to solve $P(u, v) = Q(u, v) = 0$. In our case, there are lots of solutions with $u = 0$ (since g is even), and the solutions $v = 0$, $u = \pm 1$. This helps a lot.

Intersection with the boundary Assume that $q(t)$ is the solution of the differential equation, $q(t_1)$ is stable and $q(t_2)$ is not. We want to find is t_0, defined such that $q(t_0)$ has a root on $\mathbb{T}$, all roots are in $\overline{\mathbb{U}}$. If $r(t)$ is the maximum of the modulus of the roots of $q(t)$, we want $r(t_0) = 1$. We have t_1 and t_2 with $r(t_1) < 1 < r(t_2)$. Using dichotomy, we can find t_3 and t_4, with $t_4 - t_3$ small, and $r(t_3) < 1 < r(t_4)$. In practice, the best thing to do seems to factor q_3, find the root α with greatest modulus, consider $\beta = \alpha/|\alpha|$, and pretend that $q(t_0) = q(t_3)(z - \beta)/(z - \alpha)$.

We propose here an alternate solution: define

$$q_\lambda = \lambda q_3 + (1 - \lambda) q_4.$$

Then $q(t_0)$ is q_λ for some $0 \le \lambda \le 1$, with a very good approximation ($q_3 = q(t_3)$, $q_4 = q(t_4)$). Note that $r(0) < 1 < r(1)$ if $r(\lambda)$ is now the largest modulus of the roots of q_λ, so that there exists a λ with $r(\lambda) = 1$.

Let α be a root of q_λ with modulus one. Then

$$\lambda = \frac{q_4(\alpha)}{q_4(\alpha) - q_3(\alpha)}. \tag{2.44}$$

Consider the complex case first. For λ to be real, we need $q_3(\alpha)\overline{q_4(\alpha)}$ to be real. Assume $q_3 = \sum a_i z^i$ and $q_4 = \sum b_i z^i$. Of course, we have $a_n = b_n = 1$. We get

$$\Im \sum_{ij} a_i \overline{b_j} \alpha^i \overline{\alpha}^j = 0. \tag{2.45}$$

Let $S_k = \sum_i a_i \bar{b}_{i+k}$. Then (2.45) becomes

$$\Im[S_0 + \sum_{k=1}^{n} (S_k \overline{\alpha}^k + S_{-k}\alpha^k)] = 0.$$

If $S_k = A_k + iB_k$, this is

$$B_0 + \sum_{k=1}^{n} (B_k + B_{-k}) \cos k\phi + (A_{-k} - A_k) \sin k\phi = 0 \tag{2.46}$$

with $\alpha = e^{i\phi}$. If $t = \tan \phi/2$, we get, after multiplication by $(1+t^2)^n$, an equation of the form $P(t) = 0$, where P is a polynomial of degree $2n$, hence $2n$ solutions. Note that, if P has degree less than $2n$, then $t = \infty$ is a solution, this is $\alpha = -1$.

Consider as an example the case $n = 1$. Write $q_3 = z - \beta = z - re^{i\psi}$, $q_4 = z - \gamma = z - se^{i\theta}$. With $A = s - r\cos(\psi - \theta)$, $B = r\sin(\psi - \theta)$, $C = rs\sin(\psi - \theta)$ and $\phi' = \phi - \theta$, this becomes

$$A \sin \phi' + B \cos \phi' = C. \tag{2.47}$$

This equation does not always have a solution. We know that the condition is $A^2 + B^2 \geq C^2$ (in general, not all roots of P are real). If this equation has a solution, it can always be written as

$$\cos(\phi' - \phi_0) = \cos \phi_1$$

so that $\phi = \theta + \phi_0 \pm \phi_1$. Details are left to the reader. However, equation (2.47) just says that the line passing through the points β and γ intersects the unit circle. It is well known that, unless the line is tangent, it has zero or two intersection points. Assume that β in the disk, and γ is out of the disc (this is just: q_3 stable, q_4 not stable). In this case, the line does intersect the circle. In fact, the segment $[\beta, \gamma]$ intersects the circle at a unique point. The conditions: the line (resp. the segment) intersects the circle are equivalent to say that (2.44) gives a real value for λ (resp. a value with $0 \leq \lambda \leq 1$). In higher dimension, the geometry is not easy (remember equation (2.38)).

Consider now the real case. Here α may be real. In this case it has to be ± 1, and λ is automatically real. The condition $0 \leq \lambda \leq 1$ is then $q_3(\alpha)q_4(\alpha) < 0$. But α can also be complex. It has then the form

$$\alpha = -a \pm \sqrt{a^2 - 1}$$

for some a with $-1 \leq a \leq 1$. Define a sequence of polynomials Q_k and R_k by $Q_0 = 0$, $R_0 = 1$ and

$$Q_{k+1} = R_k \qquad R_{k+1} = -2zR_k - Q_k. \tag{2.48}$$

We have

$$(-z \pm \sqrt{z^2 - 1})^k = R_k + zQ_k \pm \sqrt{z^2 - 1} Q_k.$$

Assume $q_3 = \sum \alpha_k z^k$, $q_4 = \sum \beta_k z^k$, and define

$$P_1 = \sum \alpha_k (R_k + zQ_k), \quad P_2 = \sum \alpha_k Q_k,$$

$$P_3 = \sum \beta_k (R_k + zQ_k), \quad P_3 = \sum \beta_k Q_k.$$

Then

$$q_3(-a \pm \sqrt{a^2 - 1}) = P_1(a) \pm \sqrt{a^2 - 1} P_2(a),$$
$$q_4(-a \pm \sqrt{a^2 - 1}) = P_3(a) \pm \sqrt{a^2 - 1} P_4(a).$$

Now (2.44) becomes

$$\lambda P_1(a) + (1 - \lambda) P_3(a) = 0, \quad \lambda P_2(a) + (1 - \lambda) P_4(a) = 0.$$

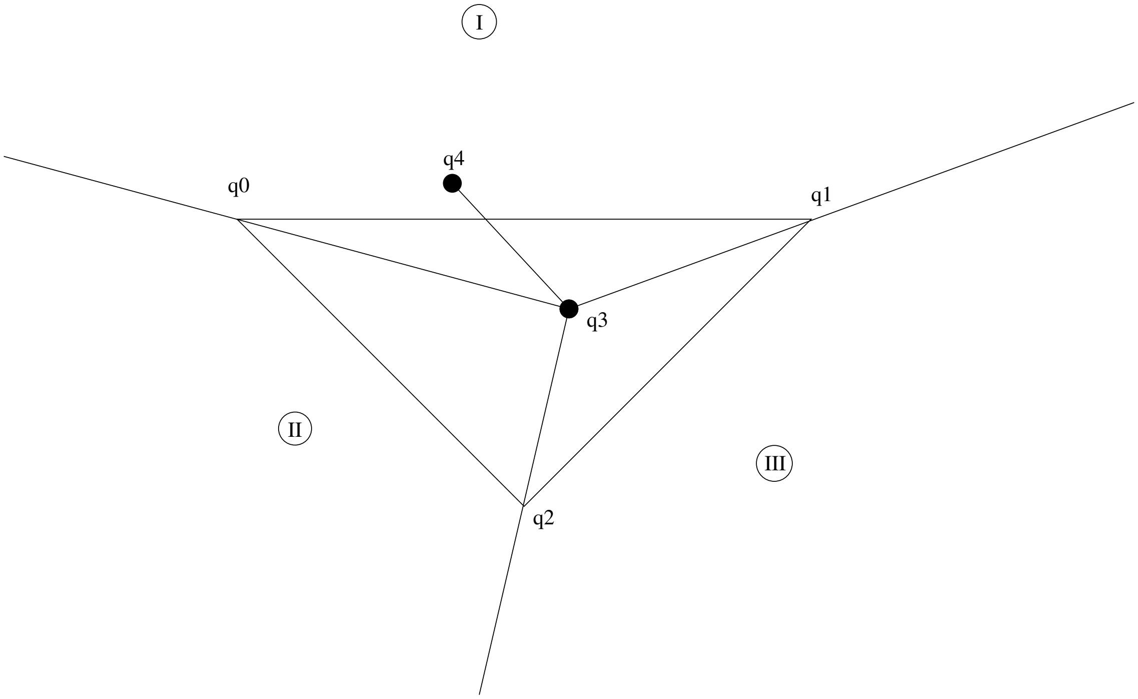

Figure 2.7: Two polynomials q_3, q_4, and the set of stable polynomials, which is in the triangle defined by $q_0 = (z-1)^2$, $q_1 = (z+1)^2$ and $q_2 = z^2 - 1$. A polynomial $q = z^2 + az + b$ is defined by the point $x = a$ and $y = b$. The half lines that start at q_3 and pass by q_i define three sectors out of the triangle.

If we eliminate λ, we get

$$P_1(a)P_4(a) - P_2(a)P_3(a) = 0. \tag{2.49}$$

A priori, this is an equation of degree $2n$. In fact, it has only degree $n-1$. This is because it simplifies to

$$\sum \alpha_i \beta_j (R_i Q_j - R_j Q_i) = 0.$$

Note that, if $S_{ij} = R_i Q_j - R_j Q_i$, then equation (2.48) says $S_{i+1,j+1} = S_{ij}$, so that we get

$$\sum_{ik} (\alpha_i \beta_{i+k} - \alpha_{i+k}\beta_i) Q_k = 0. \tag{2.50}$$

This is of degree $n-1$. Note that each solution a gives two complex conjugate values α, thus $2n-2$ solutions. If we add $\alpha = \pm 1$, we get $2n$ solutions, as in the complex case.

In order to understand what happens, consider the case $n = 2$. To each polynomial $q = z^2 + uz + v$, we associate the point (u, v) in the plane. Let $q_0 = (z-1)^2$, $q_1 = (z+1)^2$ and $q_2 = z^2 - 1$ (see figure 2.7).

These three polynomials define a triangle. A polynomial q_3 is stable if and only if it is in the triangle. On the figure, we have shown the triangle, a stable polynomial q_3, an unstable one q_4, the line that joins them. The half lines that start at q_3 and that pass through q_i $(0 \le i \le 2)$ define three sectors in the plane (shown as I, II and III).

Let's write $q_3 = \alpha_0 + \alpha_1 z + z^2$ and $q_4 = \beta_0 + \beta_1 z + z^2$. Consider (2.44) with $\alpha = 1$. We assume q_3 stable, so that, in particular $q_3(1) > 0$. The condition $0 \le \lambda \le 1$ says $q_4(1) > 0$, so that q_4 is not in the triangle (in fact, the line $(q_0 q_2)$ intersects the segment $[q_3 q_4]$). Since q_λ has 1 as root, the other root is easy to find. It is stable if and only if

$$|\alpha_0(\beta_0 + 1) - \beta_0(\alpha_1 + 1)| \le |\beta_0 + \beta_1 - \alpha_0 - \alpha_1|.$$

This condition (together with $q_4(1) < 0$) says that q_4 is not in the triangle, and is in the sector II.

On the other hand, the solution to (2.50) is

$$2a = \frac{\beta_1 - \alpha_1}{\beta_0 - \alpha_0}(1 - \alpha_0) + \alpha_1, \qquad 1 - \lambda = \frac{1 - \alpha_0}{\beta_0 - \alpha_0}.$$

Since q_3 is stable, we have $\alpha_0 < 1$. Hence the condition $0 \le \lambda \le 1$ says $\beta_0 > 1$. It is easy to check that this condition, together with $-1 \le a \le 1$ says that q_4 is not in the triangle, and is in the sector I. Geometrically, it is obvious that if q_3 is in the triangle, and q_4 is outside of the triangle, then the segment $[q_3, q_4]$ intersects one and only one side of the triangle.

In dimension greater than two, things are more complicated. For instance, the set of stable polynomials is not convex. Thus, (2.44) can have more than one solution λ with $0 \le \lambda \le 1$.

2.3.5 Derivatives of ψ

We give here formulas in the scalar case for the derivative of ψ with respect to the coefficients of q. The complexity of these formulas will be given in the last chapter. We can also use automatic differentiation, see chapter 4. There is another technique we once used: let $q = \prod(z - \alpha_i)$. We get $d\psi/d\alpha_i$ via (2.40). This works only if q has simple roots.

We write here f, g and p instead of F, G and P, assuming that everything is a scalar. In the case of a vector, nothing changes. In order to get the first derivative of ψ, we introduce the following quantities, obtained by division by q:

$$g z^{n-i} = U_i q + C_i \qquad V z^i = W_i q + D_i. \tag{2.51.a}$$

These are needed for computing the second derivative.

$$U_i z^j = X_{ij} q + Z_{ij} \qquad W_i z^j = Y_{ij} q + T_{ij}. \tag{2.51.b}$$

We shall use the relation

$$\langle A \,|\, BC \rangle = \langle A\tilde{C} \,|\, B z^n \rangle$$

if C is a polynomial of degree n.

We have, for $\deg(r) < 1$,

$$\langle f - \frac{p}{q} \,|\, \frac{rs}{q^2} \rangle = \langle \frac{\tilde{r}}{q} \,|\, \frac{Vs}{q} \rangle. \tag{2.52}$$

In fact

$$\langle f - \frac{p}{q} \,|\, \frac{rs}{q^2} \rangle = \langle \check{V} \frac{\tilde{q}}{q} \,|\, \frac{rs}{q^2} \rangle = \langle \frac{z^{n-1}}{q} \,|\, \frac{Vrs}{q} \rangle = \langle \frac{\tilde{r}}{q} \,|\, \frac{Vs}{q} \rangle.$$

We introduce the polynomial Y defined so that, for each D of degree $n - 1$:

$$\langle f \,|\, \frac{D}{q} \rangle = \langle \tilde{D} \,|\, Y \rangle$$

Such an Y exists, because the left-hand side is a linear operator on polynomials of degree less than n. Note that if $Y = \sum y_i z^i$, then, by linearity, $y_i = \langle f \,|\, z^{n-i-1}/q \rangle = \langle f z^i \,|\, z^{n-1}/q \rangle$.

Write $q = \sum_k (q_k + i r_k) z^k$, where q_k and r_k are real coefficients. If ∂r is the partial derivative of r w.r.t. r_k or q_k, since $\psi = \langle f - p/q \,|\, f - p/q \rangle$ we have

$$\partial\psi = 2\Re \langle f - \frac{p}{q} \,|\, \frac{-\partial p}{q} + \frac{p \partial q}{q^2} \rangle = 2\Re \langle f - \frac{p}{q} \,|\, \frac{p \partial q}{q^2} \rangle.$$

We applied relation (2.10) to ∂p. If we apply relation (2.52), this gives:

$$\partial\psi = 2\Re \langle \frac{\tilde{p}}{q} \,|\, \frac{V \partial q}{q} \rangle.$$

If s and D are the quotient and remainder of the division of $V\partial q$ by q, we get

$$\partial\psi = 2\Re\langle \frac{\tilde{p}}{q} \,|\, \frac{D}{q} + s\rangle = 2\Re\langle \frac{\tilde{p}}{q} \,|\, \frac{D}{q}\rangle = 2\Re\langle \frac{\tilde{D}}{q} \,|\, \frac{p}{q}\rangle = 2\Re\langle \frac{\tilde{D}}{q} \,|\, f\rangle = 2\Re\langle Y \,|\, D\rangle.$$

Computing ∂q is obvious. We have hence:

$$\frac{\partial\psi}{\partial q_i} = 2\Re\langle D_i \,|\, Y\rangle, \tag{2.53.a}$$

$$\frac{\partial\psi}{\partial r_i} = 2\Im\langle D_i \,|\, Y\rangle. \tag{2.53.b}$$

We can also use the relation $\psi = \|V\|^2$, so that $\partial\psi = 2\Re\langle V \,|\, \partial V\rangle$. Differentiating (2.11.a) gives:

$$gz^{n-i} = \frac{\partial V}{\partial q_i}q + Vz^i + \frac{\partial R}{\partial q_i},$$

$$-igz^{n-i} = \frac{\partial V}{\partial r_i}q + iVz^i + \frac{\partial R}{\partial r_i}.$$

Since derivatives of R are of degree less than n, we get

$$\frac{\partial V}{\partial q_i} = U_i - W_i \qquad \frac{\partial V}{\partial r_i} = -i(U_i + W_i).$$

Hence:

$$\frac{\partial\psi}{\partial q_i} = 2\Re\langle V \,|\, U_i - W_i\rangle, \tag{2.54.a}$$

$$\frac{\partial\psi}{\partial r_i} = 2\Im\langle V \,|\, U_i + W_i\rangle. \tag{2.54.b}$$

Computing second derivatives is now obvious, by differentiation of (2.54). The derivatives of U_i and W_i are given by

$$\frac{\partial U_i}{\partial q_j} = -X_{ij},$$

$$\frac{\partial U_i}{\partial r_j} = -iX_{ij},$$

$$\frac{\partial W_i}{\partial q_j} = -X_{ji} - Y_{ji} - Y_{ij},$$

$$\frac{\partial W_i}{\partial r_j} = -i(X_{ji} + Y_{ij} + Y_{ji}).$$

Hence:

$$\frac{\partial^2\psi}{\partial q_i \partial q_j} = 2\Re\langle U_i - W_i \,|\, U_j - W_j\rangle + 2\Re\langle V \,|\, Y_{ij} + Y_{ji} - X_{ij} - X_{ji}\rangle, \tag{2.55.a}$$

$$\frac{\partial^2\psi}{\partial q_i \partial r_j} = -2\Im\langle U_j + W_j \,|\, U_i - W_i\rangle + 2\Im\langle V \,|\, -Y_{ij} - Y_{ji} + X_{ij} - X_{ji}\rangle, \tag{2.55.b}$$

$$\frac{\partial^2\psi}{\partial r_i \partial r_j} = 2\Re\langle U_i + W_i \,|\, U_j + W_j\rangle - 2\Re\langle V \,|\, Y_{ij} + Y_{ji} + X_{ij} + X_{ji}\rangle. \tag{2.55.c}$$

2.3.6 Other formulas

Assume that q has n distinct roots, $q = \prod(z - \alpha_i)$. Equation (2.11.a) gives $R(\alpha_i) = G(\alpha_i)\tilde{q}(\alpha_i)$. Now $P/q = \sum \beta_i/(z - \alpha_i)$,

$$\langle F \mid \frac{P}{q} \rangle = \sum \langle F \mid \frac{\beta_i}{z - \alpha_i} \rangle = \sum \beta_i G(\alpha_i). \tag{2.56}$$

Question: what is the complexity of this algorithm compared to the complexity of using (2.11.a)? We may assume $\|F\|$ is pre-computed (it is independent of q). Let M be the number of coefficients of G, and $m \times p$ the dimension of G. In the scalar case, we can assume $m = p = 1$. It will cost Mn multiplications to evaluate $G(\alpha_i)$. It will costs a_n multiplications to find β_i (use Lagrange interpolation to find the coefficients of R, and β_i is nothing else than the derivative at $z = \alpha_i$ of $\tilde{R}(z - \alpha_i)/q$. Computing these quantities is independent of M). Finally n multiplications are required for computing $\beta_i G(\alpha_i)$.

This means that we can compute ψ using $nM + b_n$ multiplications, where b_n is a function of n only. There must surely exists an algorithm with the same complexity, that works even if q has multiple roots. Note that the complexity of (2.11.a) is $2Mn$, because multiplying G by $\tilde{q}$, and dividing this by q costs nM multiplications. In fact, we gain because we exploit the relation between q and $\tilde{q}$. In the matrix case, the only relation between $\tilde{D}$ and q is that the quotient is inner. Can we exploit it?

Consider

$$G = V_1 q + R_1 \qquad R_1\tilde{D} = V_2 q + R_2. \tag{2.57}$$

We have $R = R_2$, and $V = V_1\tilde{D} + V_2$. Note that computing $V_1\tilde{D}$ costs almost as many multiplications as $G\tilde{D}$. Hence the idea is to not compute V, but use (2.18) twice. In fact $\|F\|^2 - \|V\|^2$ is the square of the norm of R/q, and $\|R_1\|^2 - \|V_2\|^2$ is the square of the norm of R_2/q. Since $R = R_2$ we get

$$\psi = \|F\|^2 - \|R_1\|^2 + \|V_2\|^2. \tag{2.58}$$

The complexity for the general algorithm is $mpM(n+1)(p+1)$: it costs $mp^2M(n+1)$ to multiply G which is a $m \times p$ matrix of degree M by a $\tilde{D}$, which is a $p \times p$ matrix of degree n. It costs $mpMn$ multiplications to divide $G\tilde{D}$ which is a $m \times p$ matrix of degree $M+n$ by the monic polynomial q. Finally, it costs mpM multiplications to compute the norm of V.

For the new algorithm, we need $mp(M-n)n$ multiplications for computing V_1, $2n^2mp^2$ multiplications for computing V_2, $2nmp$ multiplications for computing the norms of R_1 and V_2.

The complexity is now $mpn(M + pn + 2 - n)$. The ratio between the two complexities is

$$r = \frac{n}{M}\frac{M + 2pn + 2 - n}{(p+1)(n+1)}.$$

In the scalar case ($p = 1$), we can exploit the fact that $\tilde{D} = \tilde{q}$ is monic, so that we get

$$r = \frac{n(M + 2 + n)}{M(2n+1)}.$$

This is roughly 1/2. In the general case, it is roughly $1/(p+1)$, see the last chapter for details. Note: this trick does not work in the case of weighted approximation. The formulas established in the previous subsection are now invalid.

2.3.7 Weighted approximation

In this section, we study some properties of orthogonal polynomials. Classically, if we have a scalar product of the form

$$\langle f \mid g \rangle = \int_a^b \overline{f(x)} g(x) h(x) dx$$

(where h is a positive function) which is defined for all polynomials, there exists a unique sequence P_n of polynomials of degree n, which is orthonormal, and such that the leading coefficient of P_n is real and positive. The main property of the scalar product is $\langle xP \,|\, Q\rangle = \langle P \,|\, xQ\rangle$, from which one can deduce two properties: there is a recurrence relation of the form

$$P_{n+1} = (a_n + b_n x)P_n + c_n P_{n-1},$$

and P_n has n real roots on the interval $[a, b]$. For more details, see [18].

In this section, we consider a scalar product of the form

$$\langle f \,|\, g\rangle = \frac{1}{2\pi}\int_0^{2\pi} \overline{f(e^{i\theta})}g(e^{i\theta})h(e^{i\theta})d\theta \tag{$*$}$$

and show that there is a recurrence relation, and the roots of P_n satisfy some properties (see also [18]). We start with the recurrence relation, study it, and then show that it is associated to a scalar product. Finally, we explain how these polynomials can be used (see [13]).

Assume that Φ_i is defined by

$$\Phi_{i-1} = \frac{1}{z}\frac{\widetilde{\Phi_i}(0)\Phi_i - \Phi_i(0)\widetilde{\Phi_i}}{\sqrt{|\widetilde{\Phi_i}(0)|^2 - |\Phi_i(0)|^2}}. \tag{2.59}$$

If Φ_i is a polynomial of degree i, then Φ_{i-1} is a polynomial of degree $i-1$, and its leading coefficient is

$$\kappa_{i-1} = \sqrt{|\widetilde{\Phi_i}(0)|^2 - |\Phi_i(0)|^2}. \tag{2.60}$$

An alternate formula is

$$\overline{\widetilde{\Phi_{i-1}}(0)}\widetilde{\Phi_{i-1}} = \overline{\widetilde{\Phi_i}(0)}\widetilde{\Phi_i} - \overline{\Phi_i(0)}\Phi_i. \tag{2.61}$$

A consequence of (2.60) is

$$\kappa_i^2 = \sum_{j=0}^{i} |\Phi_j(0)|^2. \tag{2.62}$$

A consequence of (2.59) is

$$\widetilde{\Phi_i} = \frac{1}{\kappa_i}\sum_{k=0}^{i} \overline{\Phi_k(0)}\Phi_k. \tag{2.63}$$

We shall see later the interest of these formulas. In particular, we shall prove the converse of the following theorem.

Theorem 16
If Φ_i is defined for $0 \le i \le n$, then Φ_i is stable.

Proof. The main assumption is that the quantity that appears in the square root of (2.59) is positive. Combining all equations gives

$$z\Phi_{i-1} = \frac{\kappa_i}{\kappa_{i-1}}\Phi_i - \frac{\Phi_i(0)}{\kappa_i\kappa_{i-1}}\sum_{k=0}^{i}\overline{\Phi_k(0)}\Phi_k. \tag{2.64}$$

Consider now the scalar product, defined for polynomials of degree $\le n$, by $\langle \Phi_i \,|\, \Phi_j\rangle = \delta_{ij}$. Assume $j \le i$. We have

$$\langle z\Phi_{i-1} \,|\, z\Phi_{j-1}\rangle = A - B - C + D$$

with

$$A = \frac{\kappa_i}{\kappa_{i-1}}\frac{\kappa_j}{\kappa_{j-1}}\langle \Phi_i \,|\, \Phi_j\rangle = \frac{\kappa_i\kappa_j}{\kappa_{i-1}\kappa_{j-1}}\delta_{ij}.$$

$$B = \frac{\kappa_i}{\kappa_{i-1}} \frac{\Phi_j(0)}{\kappa_j \kappa_{j-1}} \langle \Phi_i \,|\, \sum_{k=0}^{j} \overline{\Phi_k(0)} \Phi_k \rangle.$$

In case $j \le i$, the last scalar product is $\overline{\Phi_i(0)}\delta_{ij}$, so that

$$A - B = \frac{\kappa_i}{\kappa_{i-1}} \left(\frac{\kappa_i}{\kappa_{i-1}} - \frac{|\Phi_i(0)|^2}{\kappa_i \kappa_{i-1}} \right) \delta_{ij} = \delta_{ij}.$$

We have

$$C = \frac{\overline{\Phi_i(0)}}{\kappa_i \kappa_{i-1}} C' \qquad D = \frac{\overline{\Phi_i(0)}}{\kappa_i \kappa_{i-1}} D',$$

$$C' = \langle \sum_{k=0}^{i} \overline{\Phi_k(0)} \Phi_k \,|\, \frac{\kappa_j}{\kappa_{j-1}} \Phi_j \rangle = \frac{\kappa_j}{\kappa_{j-1}} \Phi_j(0)$$

$$D' = \frac{\Phi_j(0)}{\kappa_j \kappa_{j-1}} \langle \sum_{k=0}^{i} \overline{\Phi_k(0)} \Phi_k \,|\, \sum_{k=0}^{j} \overline{\Phi_k(0)} \Phi_k \rangle = \frac{\Phi_j(0)}{\kappa_j \kappa_{j-1}} \sum_{k=0}^{j} |\Phi_k(0)|^2 = \Phi_j(0) \frac{\kappa_j^2}{\kappa_j \kappa_{j-1}} = C'.$$

Hence $\langle z\Phi_{i-1} \,|\, z\Phi_{j-1} \rangle = \delta_{ij}$ if $j \le i$. By symmetry, it is true for all i and j. From this, we deduce

$$\langle zA \,|\, zB \rangle = \langle A \,|\, B \rangle \tag{2.65}$$

whenever A and B are polynomials of degree $< n$. Hence

$$\|(z - \alpha)A\|^2 = \|A\|^2(1 + |\alpha|^2) - 2\Re(\alpha \langle zA \,|\, A \rangle)$$

from which we get $|\langle zA \,|\, A \rangle| < \|A\|^2$ if $A \ne 0$.

Hence, if $\Phi_i = (z - \beta)A$, $\langle \Phi_i \,|\, A \rangle = 0$ implies $\beta = \langle A \,|\, zA \rangle / \|A\|^2$, hence $|\beta| < 1$ and Φ_i is stable. □

Note: if Φ_i is a polynomial of degree exactly i, then we have a relation

$$z\Phi_{n-1} = \sum_{i=0}^{n} a_{in} \Phi_i.$$

If Φ_i is orthogonal for a scalar product that satisfies $\langle zP \,|\, Q \rangle = \langle P \,|\, zQ \rangle$, then $a_{in} = 0$ unless $i = n$, $n-1$, $n-2$. On the other hand, if the scalar product satisfies (2.65), then coefficients a_{in} are defined by (2.64). The argument is as follows.

Assume that we have a scalar product defined for all polynomials that satisfies (2.65). There exists an orthonormal basis Φ_i, where Φ_i is of degree i (if we require that the leading coefficient of Φ_i is real and positive, this basis is unique). Define

$$K_n(z, a) = \sum_{i=0}^{n} \overline{\Phi_i(a)} \Phi_i(z). \tag{2.66}$$

Lemma 29
If p is a polynomial of degree $\le n$, then $|p(a)| \le \|p\| \sqrt{K_n(a, a)}$, and equality holds if and only if there exists a constant λ such that $p(z) = \lambda K_n(z, a)$.

Proof. Write $p = \sum \lambda_i \Phi_i$. Apply Cauchy-Schwarz to $p(a) = \sum \lambda_i \Phi_i(a)$, and use $\sum |\lambda_i|^2 = \|p\|^2$. □

Lemma 30

$$K_n(z, a) = (z\overline{a})^n K_n(1/\overline{a}, 1/\overline{z}). \tag{2.67}$$

Proof. Take p of degree $\leq n$. Let $r = \tilde{p}$. The main assumption on the scalar product is $\|p\| = \|r\|$. Apply the lemma for $p(a)$ and $r(1/\overline{a})$. We get

$$|p(a)|^2 \leq \|p\|^2 K_n(a,a),$$

$$|p(a)|^2 \leq \|p\|^2 |a|^{2n} K_n(1/\overline{a}, 1/\overline{a}).$$

Now, equality can hold in both cases. Hence

$$K_n(a,a) = |a|^{2n} K_n(1/\overline{a}, 1/\overline{a}). \tag{$*$}$$

Moreover, equality holds if $p = K_n(\cdot, a)$, so that r must be a multiple of $K_n(\cdot, 1/\overline{a})$. This means

$$z^n K_n(a, 1/\overline{z}) = \lambda K_n(z, 1/\overline{a})$$

(because $K_n(a,b) = \overline{K_n(b,a)}$). Replace a by $1/\overline{a}$, and use $(*)$ to identify λ. □

Corollary 6

$$K_n(z,a) = \sum_{i=0}^{n} (z\overline{a})^n \overline{\widetilde{\Phi_n}(a)} \widetilde{\Phi_n}(z).$$

$$K_n(z,0) = \overline{\widetilde{\Phi_n}(0)} \widetilde{\Phi_n}(z).$$

Moreover, equations (2.59) and (2.61) hold.

Proof. Replace in the right-hand side of (2.67) the value of K_n obtained from (2.66). This gives the first relation. Evaluate at $a = 0$, this gives the second one. Evaluate this at $n = i$, $n = i - 1$, take the difference. This gives (2.61) hence (2.59). □

In what follows, we consider the scalar product

$$\langle f \,|\, g \rangle_h = \frac{1}{2\pi} \int_0^{2\pi} \overline{f(e^{i\theta})} g(e^{i\theta}) \frac{d\theta}{|h(e^{i\theta})|^2}. \tag{2.68}$$

We shall make the assumption that h is continuous and non-zero on $\mathbb{T}$ (later on that h is a polynomial), so that the scalar product is defined whenever f and g are in H^2. In particular, there exists an orthonormal basis Φ_i, denoted by $\Phi_i(h)$, that satisfies equation (2.59).

Lemma 31
Let q be a stable polynomial of degree n. There exists a sequence of polynomials Φ_i, that satisfy the recurrence relation (2.59) and $\Phi_n = q$.

This is the converse of Theorem 16.
Proof. Assume first that the leading coefficient of q is real and positive. We pretend that q is orthogonal to each polynomial p of degree $< n$ for the scalar product $\langle f \,|\, g \rangle_q$, this is trivial to check. This implies $\Phi_n(q) = q$ and the sequence $\Phi_i(q)$ satisfies the requirements. If we multiply q by an element of $\mathbb{T}$, the value of Φ_{n-1} is unchanged. □

Note that $\Phi_{i+n}(q) = z^i q$. Hence, it is possible to compute every Φ_i. In the general case, assume that K_n has a limit K_∞ when $n \to \infty$. Then

$$\langle K_\infty(z,a) \,|\, f \rangle = f(a) \tag{$*$}$$

whenever everything is defined. Moreover, there exists a function D such that

$$K_\infty(z,a) = \frac{1}{\overline{D(a)} D(z)} \frac{1}{1 - \overline{a} z}$$

(see [18, Formula 12.3.17] for details).

In the special case under consideration, we have

$$K_\infty(z,a) = K_{n-1}(z,a) + \frac{\overline{q(a)}q(z)}{1-\overline{a}z}.$$

In fact

$$K_\infty(z,a) = \frac{\overline{\tilde{q}(a)}\tilde{q}(z)}{1-\overline{a}z}$$

(because of the next lemma). Quantities K that satisfy relation $(*)$ are called reproducing kernels, and are studied in the next chapter.

Lemma 32
We have

$$K_{n-1}(z,a) = \frac{\overline{\widetilde{\Phi_n}(a)}\widetilde{\Phi_n}(z) - \overline{\Phi_n(a)}\Phi_n(z)}{1-\overline{a}z}. \tag{2.69}$$

Proof. Let T be the numerator of the right-hand side of (2.69), and $S = T/(1-\overline{a}z)$. It is obvious that S is a polynomial of degree at most n. The leading coefficient of S is the complex conjugate of

$$\frac{\Phi_n(a)\widetilde{\Phi_n}(0) - \widetilde{\Phi_n}(a)\Phi_n(0)}{a}$$

which is $\kappa_{n-1}\Phi_{n-1}(a)$ according to (2.59). According to (2.66), this is the complex conjugate of the leading coefficient of K_{n-1}.

Let p be a polynomial of degree $< n$. Write $r = (p(z)-p(a))/(z-a)$ so that $p = (z-a)r + p(a)$. Now $\langle S \mid p\rangle = \langle T \mid rz\rangle + \langle S \mid p(a)\rangle$. Since rz is of degree $< n$, it is orthogonal to Φ_n. It is also orthogonal to $\widetilde{\Phi_n}$ because $\langle \widetilde{\Phi_n} \mid rz\rangle = \langle \widetilde{r} \mid \Phi_n\rangle$, and $\tilde{r}$ is of degree $n-1$. Hence $\langle S \mid p\rangle = p(a)\langle S \mid 1\rangle = C_a p(a)$, where C_a depends only on a.

By definition, $\langle K_n \mid p\rangle = p(a)$ whenever p has degree less than n. Hence $K_n(z,a) = C_a S(z,a)$. Comparing leading coefficients gives $C_a = 1$. $\square$

Theorem 17
Assume w stable, of degree k. For each n, there exists a number $\alpha_n > 1$, such that if q is of degree n, and has roots less than α_n in module, then the recurrence relations (2.59) *are satisfied for $k \le i \le n+k$ with $\Phi_{n+k} = qw$.*

Proof. 1. Let's define $\Phi_i(q)$ to be the polynomial of degree i defined by the recurrence relation (2.59) with $\Phi_n(q) = q$ for $n = \deg(q)$. This agrees with the previous definition in case q is stable.

2. Assume $q = q_1q_2$, where q_i has degree n_i. Suppose that q_1 is stable and q_2 has its roots on $\mathbb{T}$. The previous theorem says that $\Phi_i(q_1)$ is defined. Since $q_2 = q_2(0)\widetilde{q_2}$, equation (2.59) says $\Phi_{i+n_2}(q_1q_2) = q_2\Phi_i(q_1)$. Hence $\Phi_i(q_1q_2)$ exists for $i \ge n_2$. Note that $\Phi_{n_2}(q_1q_2)$ is a constant times q_2. Trying to apply (2.59) to it will fail.

3. Let λ be a real number. Define $q_\lambda(z) = \lambda^n q(z/\lambda)$. Let

$$\kappa_i(q,\lambda) = |\widetilde{\Phi_i}(wq_\lambda)(0)|^2 - |\Phi_i(wq_\lambda)(0)|^2.$$

Polynomials $\Phi_i(wq_\lambda)$ are defined if $\kappa_i(q,\lambda)$ is real and positive. For fixed q, this is a continuous function of λ. Assume that q has all its roots in $\overline{\mathbb{U}}$. Then $\kappa_i(q,1) > 0$ for $i > k$, because w is stable. Hence $\kappa_i(q,\lambda) > 0$ if $\lambda \le \lambda_0$ for some $\lambda_0 > 1$. The conclusion follows now from the compactness of $\overline{\mathbb{U}}$. $\square$

Approximation of type $(n-s,n)$ Let $\langle f \,|\, g\rangle_q$ be the scalar product defined by (2.68), Φ_i the orthogonal polynomials associated to this scalar product. Equation (2.10) is

$$\langle Fq - P \,|\, S\rangle_q = 0.$$

Let $(e_1,\dots,e_m)$ be a basis of $\mathbb{C}^m$. We can write $P = \sum_{ij} \lambda_{ij}\Phi_j e_i$. The condition is, for each Φ_k, e_l,

$$\langle Fq - \sum \lambda_{ij}\Phi_j e_i \,|\, \Phi_k e_l\rangle_q = 0. \tag{$*$}$$

Hence

$$\langle Fq \,|\, \Phi_j e_i\rangle_q = \overline{\lambda_{ij}}$$

and

$$P = \sum_{i=1}^{m}\sum_{j=0}^{n-1} \langle \frac{\Phi_j e_i}{q} \,|\, F\rangle \Phi_j e_i. \tag{2.70.a}$$

Assume that we want to minimise $\|F - P/q\|$ with $\deg(P) < n-s$. The minimum P_0 is like P above, but the sum is for $j = 0$ to $j = n-s-1$.

$$P_0 = \sum_{i=1}^{m}\sum_{j=0}^{n-s-1} \langle \frac{\Phi_j e_i}{q} \,|\, F\rangle \Phi_j e_i. \tag{2.70.b}$$

If we apply (2.14) to P and P_0 we get

$$\|F - P_0/q\|^2 = \|F - P/q\| + \langle F \,|\, (P-P_0)/q\rangle$$

hence

$$\|F - \frac{P_0}{q}\|^2 = \|V\|^2 + \sum_{i=1}^{m}\sum_{j=n-s}^{n-1} |\langle \frac{\Phi_j e_i}{q} \,|\, F\rangle|^2. \tag{2.70.c}$$

Weighted approximation Assume that we want to approximate $\|F - P/q\|_w$, where the scalar product is defined by (2.68). We consider here only the case where w is a polynomial. Assume that w is stable, of degree k. Let $F_w = F/w$. Then $\|F - P/q\|_w = \|F_w - P/(qw)\|$. We want $\deg(P) < \deg(qw) - k$. Applying equations (2.70) gives

$$\langle \Phi_i \,|\, \Phi_j\rangle_{qw} = \delta_{ij}, \tag{2.71.1}$$

(this equation is equivalent to $\Phi_{n+k} = qw$, together with (2.59)),

$$Gz^k\tilde{q} = Vqw + R, \tag{2.71.2}$$

(note that $G_w = Gz^k/\tilde{w}$),

$$\psi_w(q) = \|V\|^2 + \sum_{i=1}^{m}\sum_{j=n}^{n+k-1} |\langle \frac{\Phi_j e_i}{q} \,|\, F\rangle_w|^2, \tag{2.71.3}$$

and

$$P = \tilde{R} - \sum_{i=1}^{m}\sum_{j=n}^{n+k-1} \langle \frac{\Phi_j e_i}{q} \,|\, F\rangle_w \Phi_j e_i. \tag{2.71.4}$$

Theorem 17 says that polynomials Φ_j that appear in (2.71.3) and (2.71.4) are defined even if q is unstable, but its roots are not too big. In other words, ψ_w is still C^∞ on the boundary of the set of stable polynomials. In fact, if $q = q_1q_2$, where q_1 is stable, and q_2 has roots on $\mathbb{T}$, if Φ'_i is the orthogonal polynomial defined for the scalar product $\langle\cdot\,|\,\cdot\rangle_{wq_1}$, we have $\Phi'_{m+i} = q_2\Phi_i$ (m is the degree of q_1) so that equations (2.71) say $\psi_w(q_1q_2) = \psi_w(q_1)$ and $L_w(q)/q = L_w(q_1)/q_1$.

Note: assume that we have any scalar product. If we can compute $\langle z^i/q \,|\, z^j/q\rangle$, it is easy to find polynomials Φ_i such that

$$\langle \frac{\Phi_i}{q} \,|\, \frac{\Phi_j}{q} \rangle = \delta_{ij}. \tag{2.72}$$

Then (2.70.a) is still true, and

$$\frac{P}{q} = \sum_i \sum_j \langle \frac{\Phi_j e_i}{q} \,|\, F \rangle \frac{\Phi_j e_i}{q}. \tag{2.73}$$

We have

$$\|F - P/q\|^2 = \|F\|^2 - \sum_{i=1}^{m} \sum_{j=0}^{n-1} |\langle \frac{\Phi_j e_i}{q} \,|\, F \rangle|^2. \tag{2.74}$$

We can compute this quantity provided that we can compute a scalar product of the form $\langle F \,|\, z^i/q \rangle$.

2.4 Continuous time systems

Consider the equations

$$x_{k+1} = Fx_k + Gu_k, \qquad y_k = Hx_k. \tag{2.75}$$

Define

$$\mathcal{H} = H(zI - F)^{-1}G. \tag{2.76}$$

The matrix $\mathcal{H}$ is called the transfer function of the discrete time system (2.75). If we define $X = \sum_{k=0}^{\infty} x_k/z^{k+1}$, and likewise for U and Y, we get

$$Y = \mathcal{H}U + \sum_{k=0}^{\infty} HF^k x_0/z^{k+1}.$$

If $x_0 = 0$, then $Y = \mathcal{H}U$. But if F is stable, the contribution of the additional term to y_k is small for large k. This means that, if we wait long enough, we can measure $\mathcal{H}$ with great precision.

Consider now

$$\frac{dx}{dt} = Fx + Gu, \qquad y = Hx. \tag{2.77}$$

We have

$$y(t) = \int_0^t He^{(t-s)F} Gu(s)ds + He^{tF}x(0). \tag{2.78}$$

Assume that F is stable. This means here that the eigenvalues of F satisfy $\Re z < 0$. It implies that the second term is small if t is large. Consider now $u = e^{i\omega t}u_0$. Let $\mathcal{H}(i\omega) = Ae^{i\phi}$. Then

$$y(t) = e^{i(\omega t + \phi)} A u_0. \tag{2.79}$$

Assume now $\mathcal{H}$ real. This means $\mathcal{H}(-i\omega) = Ae^{-i\phi}$. Replace in (2.79) ω by $-\omega$, and add. We get

$$u = \cos(\omega t)u_0 \Longrightarrow y = \cos(\omega t + \phi)Au_0. \tag{2.80}$$

Of course, these equations are valid only if $x(0) = 0$. In the case where F is stable, they are valid, if we neglect the transitory term.

What we can say is the following. Assume that equations (2.77) describe a physical stable system, and that (H, F, G) is canonical. Then we can measure $\mathcal{H}(i\omega)$ for some values of ω. Of course, the equations are only approximations, that are in general false if ω is too big. On the other hand, if ω is too small, we have to wait a long time before we can find something (at least $2\pi/\omega$). One part of the HYPERION software solves this problem. Given some measurements, it returns a function $\mathcal{H}$ which is stable, and approximates the measurements; this is called the completion algorithm. However, this $\mathcal{H}$ is not rational.

It has a priori a big McMillan degree. The second part of HYPERION is the rational approximation: find a rational matrix of given McMillan degree that approaches at best the result of the previous part of the software. This report explains how to do rational approximation.

One difficulty however, is that the rational approximation algorithm works only for functions that are stable in the discrete case. The remainder of this chapter explains how to deal with this problem. The idea is the following

1. We start with a device with transfer function $\mathcal{H}'_0 \in \Sigma_{cn}$. We measure it and obtain $\mathcal{H}'$.
2. We consider the discrete time equivalent $\mathcal{H}$ of $\mathcal{H}'$: $\mathcal{H} = \sigma_n^{-1}(\mathcal{H}')$.
3. We consider the completion $\mathcal{H}_1$ of $\mathcal{H}$: $\mathcal{H}_1 = c_k(\mathcal{H})$.
4. We compute the rational approximation $\mathcal{H}_2$ of $\mathcal{H}_1$: $\mathcal{H}_2 = a_n(\mathcal{H}_1)$.
5. We convert $\mathcal{H}_2$ to a continuous time system: $\mathcal{H}'_2 = \sigma_n(\mathcal{H}_2)$.

Let's start with some definitions. If U is a simply connected region of the complex plane (but not the plane itself), it is conformally equivalent to $\mathbb{U}$ (see [17, 14.8]). For instance, if $\mathbb{P}$ is the half plane, the set of complex numbers s such that $\Re(s) < 0$, it is conformally equivalent to the disk. This gives one possibility for σ_n. More generally, it allows us to define the Hardy space of U. This definition is not unique. For instance, in the case of the half plane, there are are two natural possibilities.

We shall define $H(\mathbb{P})$ and $\overline{H}(\mathbb{P})$ to be the set of analytic functions in $\mathbb{P}$ and outside $\mathbb{P}$, that have a limit (in fact a non-tangential limit) almost everywhere on the imaginary axis such that

$$c\int_{-\infty}^{\infty} \frac{|f(i\omega)|^2}{1+\omega^2} d\omega \tag{2.81}$$

is finite (we could omit the denominator, this gives the alternate Hardy space of the half plane). Note that the constant function one is in the Hardy space. We can chose c such that the quantity (2.81) is 1 for this function. It happens that a function of $H(\mathbb{P})$ is uniquely defined by its value on the imaginary axis. If we define the norm of f to be the square root of the quantity (2.81), then $H(\mathbb{P})$ is a Hilbert space, as well as $\overline{H(\mathbb{P})}$, and the mapping $f(z) \to f(-z)$ is an isometry between these two spaces.

If U is a set, we define $\Sigma(U)$ to be $\overline{H(U)}$. We shall use this notation only in the case where the set U is the disk or the half plane. It is a subset of the functions analytic outside U. We define also $\Sigma'(U)$ to be the functions that vanish at infinity.

If U is any set, we define $\Sigma_n(U)$ to be the set of rational functions of McMillan degree n which are analytic outside U (more precisely, which have poles in U). We define $\Sigma'_n(U)$ to be the functions of $\Sigma_n(U)$ which are strictly proper (vanish at infinity). Moreover

$$\Sigma_{dn} = \Sigma_n(\mathbb{U}) \qquad \Sigma_{cn} = \Sigma_n(\mathbb{P}) \tag{2.82.a}$$

$$\Sigma_d = \Sigma(\mathbb{U}) \qquad \Sigma_c = \Sigma(\mathbb{P}). \tag{2.82.b}$$

Note that Σ_{cn} is a subset of Σ_c (of course, Σ_{dn} is a subset of Σ_d).

In the algorithm proposed above, we assume that $\mathcal{H}'_0$, the transfer function of the device to approximate is in Σ_{cn}. This means that it is rational, of McMillan degree n. The first requirement we make is that the result $\mathcal{H}'_2$ is also in Σ_{cn}. The second requirement is that $\mathcal{H}'_2$ should be near $\mathcal{H}'_0$. This means that the functions σ_n^{-1}, c_k, a_n and σ_n has to satisfy some properties. Let's discuss them.

We first start with a_n and c_k. The function c_k is the completion algorithm. It satisfies the following properties: it takes as argument some measurements of the transfer function and returns a stable transfer function. Since only a finite number of measurements can be made, it takes as input K numbers and returns an infinite number of Fourier coefficients, and hence is not well-defined. To solve the problem, we shall assume that the measurements are done on a whole interval $[a, b]$ (numerically, we interpolate the finite number of data by cubic splines). The function c_k depends on the interval $[a, b]$, and on the norm of the function outside the interval $[a, b]$. This means that the user has to guess this. We shall assume that, if the device to approximate is rational, of McMillan degree n, if the interval $[a, b]$ is big enough, if enough measurements are made in the interval, and if k is well chosen, then $c_k(\mathcal{H})$ is near $\mathcal{H}_0$.

The function a_n is the rational approximation algorithm:

$$a_n : \Sigma_d \quad \rightarrow \quad \Sigma_{dN}.$$

It can be defined as the best rational approximation of McMillan degree N (for the H^2 norm) of its argument. We shall make the assumption that, if $\mathcal{H}_k$ converges in some sense to $\mathcal{H}$, which is rational, of McMillan degree n then $a_n(\mathcal{H}_k)$ converges in the H^∞ norm to $\mathcal{H}$ (note that if a sequence of rational functions f_k of degree N converge to a rational function of degree N, it converges in the H^∞ norm, because everything under consideration is in a finite dimensional space; this is true if, for instance, the convergence of f_k implies that $f_k(z_j)$ converges for an infinite number of values of z_j).

In short, we shall assume that, if the device to approximate is rational, of McMillan degree N, if the interval $[a, b]$ is big enough, if enough measurements are made in the interval, and if k is well chosen, then $a_n(c_k(\mathcal{H}))$ is near $\mathcal{H}_0$, the discrete-time equivalent of $\mathcal{H}'_0$.

The question is now: how to chose σ_n and σ_n^{-1}. Note first that σ_n^{-1} maps a function defined on an interval $[\omega_1, \omega_2]$ into a function defined on $[\theta_1, \theta_2]$. For simplicity, we may assume that $[\theta_1, \theta_2]$ is $[-\pi/2, \pi/2]$ (this makes the code of c_k easier). This is OK if the measurement interval is fixed, but is inadequate if we let $-\omega_1 = \omega_2 \rightarrow \infty$. In this latter case, we can chose $\tan(2\theta_i) = \omega_i$.

Consider a generic system with n poles, and apply σ_n^{-1}, then the completion algorithm. If the rational approximation gives N poles, applying σ_n must give n poles. This will imply $N \geq n$ (in general every element of Σ_{cn} must be in the image of σ_n). The question is: can we have $N > n$? The answer is, yes, of course, provided that we add additional conditions on c_k and a_n. An example is the following. Let σ_n be as below (i.e. MacMillan degree preserving). We consider $\sigma_n'^{-1}(f) = \sigma_n^{-1}(f)/z$. The completion algorithm takes a function, multiplies it by z, applies c_k, and multiplies the result by $1/z$. The rational approximation takes its argument, multiplies it by z, calls a_n, and divides the result by z. Now, σ takes its argument, multiplies it by z, and calls σ_n. What is wrong here? The trouble is that the modified a_n can only be used in this case (otherwise, it is non-continuous, or non McMillan degree preserving). Moreover, the computations are exactly the same as for a McMillan degree preserving σ_n.

We shall from now on assume that σ_n is McMillan degree preserving. In fact, we shall moreover assume that it is linear, so that

$$\sigma_n(\sum \frac{a_i}{z - \alpha_i}) = \sum a_i \sigma_n(\frac{1}{z - \alpha_i}). \tag{2.83.a}$$

This gives a function σ such that

$$\sigma(\frac{a}{1 - z\alpha}) = \frac{ab}{z - \beta}. \tag{2.83.b}$$

Since every $a/(z - \alpha)$ can be the result of the rational approximation, we get a function $\beta = \phi(\alpha)$ defined on $\mathbb{U}$, with values in $\mathbb{P}$. On the other hand, σ_n^{-1} maps $ab/(z - \beta)$ onto some $a/(z - \alpha)$, this gives a function $\alpha = \psi(\beta)$, defined on $\mathbb{P}$, with values in $\mathbb{U}$. Moreover

$$\forall \beta \in \mathbb{P} \qquad \phi(\psi(\beta)) = \beta. \tag{2.84}$$

We assume that ϕ and ψ are continuous. The open mapping theorem says that the image of ψ is open. The continuity of ϕ and ψ, together with equation (2.76) says that the image of ψ is closed (with respect to the topology induced on $\mathbb{U}$). Thus, ψ is surjective. Hence ϕ and ψ are bijections.

The result of this discussion is: a good σ_n is such that its action on the poles is well defined, and is a bijection between the set of discrete-time stable poles and the set of continuous-time stable poles.

Let's define three spaces, X_0, X_1, X_2, formed of bijections $\mathbb{P} \rightarrow \mathbb{P}$, $\mathbb{U} \rightarrow \mathbb{U}$ and $\mathbb{U} \rightarrow \mathbb{P}$. Now ϕ is in X_2. Note that X_1 contains a lots of functions, for instance the mapping $z \rightarrow \overline{z}$. If we assume that $\phi_1 \in X_1$ is rational, it has to be an inner function, hence a Blaschke product, with a single pole. The Schwarz lemma shows that if ϕ_1 is analytic, then it has to be rational (in the proof of the next theorem, point 2 is nothing else than the Schwarz lemma).

Theorem 18
The sets X_0 and X_1 are groups. If we restrict elements to be analytic, then the spaces X_i are manifolds of dimension three. Each X_i is formed of functions ϕ_i of the form

$$\phi_0(z) = \frac{az+ib}{icz+d}, \qquad \exists\lambda, \lambda a, \lambda b, \lambda c, \lambda d \textit{ real }, \Re\phi_0(-1) < 0, \tag{2.85.a}$$

$$\phi_1(z) = c\frac{z-\alpha}{1-\overline{\alpha}z}, \qquad |c| < 1, |\alpha| < 1, \tag{2.85.b}$$

$$\phi_2(z) = \lambda\frac{ze^{-i\psi_1} - e^{i\psi_1}}{ze^{-i\psi_2} + e^{i\psi_2}}, \qquad \lambda \textit{ real }, \Re\phi_2(0) < 0. \tag{2.85.c}$$

Proof. 1. The first claim is obvious. It remains true if every function is assumed analytic.

2. Let ϕ be an element of X_1. Let c be an element of $\mathbb{U}$ such that $\overline{c}\phi'(0)$ is real and positive (note that ϕ' is never zero), and $\alpha = -\overline{c}\phi(0)$. Let

$$\phi_1(z) = \frac{\overline{c}\phi(z)+\alpha}{1+\overline{c\alpha}\phi(z)}.$$

We have

$$\phi(z) = c\frac{\phi_1(z)-\alpha}{1-\overline{\alpha}\phi_1(z)}.$$

These formulas show that ϕ_1 is an element of X_1. If we show that $\phi_1(z) = z$, then ϕ has the form (2.85.b). Note that elements of the form (2.85.b) map $\mathbb{U}$ into itself, and are obviously bijective.

By construction, we have $\phi_1(0) = 0$. Because

$$\phi_1'(0) = \frac{\overline{c}}{1-|\alpha|^2}\phi'(0)$$

the quantity $\phi_1'(0)$ is real positive.

Let now $\psi(z) = \phi_1(z)/z$. This function is analytic in $\mathbb{U}$. If $\psi(z) = \sum a_k z^k$, we have

$$\sum |a_k|^2 r^{2k} = \frac{1}{2\pi}\int_0^{2\pi} |\psi(re^{i\theta})|^2 d\theta,$$

if $0 < r < 1$. Since ϕ_1 maps $\mathbb{U}$ into itself, the last term is at most $1/r^2$. Taking the limit for $r = 1$ gives $\sum |a_k|^2 \leq 1$ (this is just the maximum modulus principle). In particular, we get $|a_0| \leq 1$. But $a_0 = \phi_1'(0)$. If ϕ_2 is the inverse mapping of ϕ_1, it satisfies the same condition, $|\phi_2'(0)| \leq 1$. But the chain rule says $\phi_1'(0)\phi_2'(0) = 1$. Thus $|\phi_1'(0)| = 1$. Since $|a_0| = 1$, the condition $\sum |a_k|^2 \leq 1$ says $a_k = 0$ for $k > 0$, so that ϕ is constant. Since we assumed that a_0 is real and positive, we have $a_0 = 1$, $\psi(z) = 1$ and $\phi_1(z) = z$.

3. Let

$$\phi_3(z) = \frac{z-1}{z+1}$$

It is obvious to check that this is an element of X_2. If ϕ_2 is any element of X_2, $\phi_3^{-1} \circ \phi_2 \in X_1$. If this has the form (2.85.b) then

$$\phi_2(z) = C\frac{z-A}{z+B}$$

with

$$A = \frac{c\alpha+1}{c+\overline{\alpha}}, \quad B = \frac{1-c\alpha}{c-\overline{\alpha}}, \quad C = \frac{c+\overline{\alpha}}{c-\overline{\alpha}}.$$

Note that $|A| = |B| = 1$, $C^2A/B = |(1+c\alpha)/(1-c\alpha)|^2 > 0$, and $AC/B = -1/\phi_3(c\alpha)$, hence $\Re(AC/B) > 0$. If $A = e^{2i\psi_1}$ and $B = e^{2i\psi_2}$, then for $\lambda = Ce^{i\psi_2 - i\psi_1}$ we get (2.85.c). The condition $C^2A/B > 0$ is $\lambda^2 > 0$, so that λ is real.

4. If $\phi_0 \in X_0$, then $\phi_3^{-1} \circ \phi_0 \circ \phi_3 \in X_1$, from which it is easy to deduce (2.85.a). Note that this can also be written as

$$\phi_0(z) = \frac{z - i\beta + i\lambda c}{ic(z - i\beta) + \lambda}$$

where β, λ and c are real, and λ is positive. □

Theorem 19
Let $\phi = (az+b)/(cz+d)$ with $ad - bc \neq 0$. Let V be an open subset of the plane, and $U = \phi(V)$. Define

$$\mathcal{H}_1(z) = \mathcal{H}(\frac{az+b}{cz+d}) \quad \mathcal{H}_2(z) = \frac{1}{cz+d}\mathcal{H}(\frac{az+b}{cz+d}). \tag{2.86}$$

Define σ_ϕ and σ'_ϕ by $\sigma_\phi(\mathcal{H}) = \mathcal{H}_1$ and $\sigma'_\phi(\mathcal{H}) = \mathcal{H}_2$.

Assume that ∞ is neither in V nor U (this means that there is no λ such that $|z| \geq \lambda$ implies $z \in V$). Then σ_ϕ is a bijection from $\Sigma_n(U)$ into $\Sigma_n(V)$ and σ'_ϕ is a bijection from $\Sigma'_n(U)$ into $\Sigma'_n(V)$.

Proof. Consider a minimal realization of $\mathcal{H}$

$$\mathcal{H} = H(zI - F)^{-1}G + J. \tag{2.87.a}$$

Assume that $\mathcal{H}$ is in $\Sigma_n(U)$ or $\Sigma'_n(U)$. Then F is of size n, and its eigenvalues are in U. Since $a/c = \phi(\infty)$ is not in $\phi(V)$, it implies that a/c is not an eigenvalue of F, so that $aI - cF$ is an invertible matrix.

Define

$$F' = (-bI + dF)(aI - cF)^{-1}$$

then

$$\mathcal{H}_1 = \sigma_\phi(\mathcal{H}) = (ad - bc)H(aI - cF)^{-2}(zI - F')^{-1}G + cH(aI - cF)^{-1}G + J \tag{2.87.b}$$

$$\mathcal{H}_2 = \sigma'_\phi(\mathcal{H}) = H(aI - cF)^{-1}(zI - F')^{-1}G. \tag{2.87.c}$$

In fact, to the quantity $\mathcal{H}_2$ defined above, we have to add the term $J/(cz+d)$. If $J \neq 0$, the McMillan degree of $\mathcal{H}_2$ is in general not the same as the McMillan degree of $\mathcal{H}$. When we consider σ'_ϕ, we assume that its argument is in Σ', i.e. that $J = 0$.

Assume $F'x = \lambda x$, with $x \neq 0$. In other words, x is an eigenvector of F', with eigenvalue λ. Then $(d + \lambda c)Fx = (\lambda a + b)x$. Since $d + \lambda c$ and $\lambda a + b$ cannot vanish simultaneously, we have $d + \lambda c \neq 0$. Let $\mu = \phi(\lambda)$. Then x is an eigenvector of F, with eigenvalue μ.

Let $y = (aI - cF)^{-i}x$. Then $y = (a - c\mu)^{-i}x$, so that y is also an eigenvector of F. If $Hy = 0$, then $y = 0$, hence $x = 0$. Said otherwise, if $H_i = H(aI - cF)^{-i}$, then the pair (H_i, F') is observable. Since (F', G) is reachable (the transpose of F and F' having the same eigenvalues and eigenvectors), it follows that $\mathcal{H}_1$ and $\mathcal{H}_2$ have McMillan degree n. There poles are in $\sigma(U)$, hence V.

Thus, σ_ϕ and σ'_ϕ map an element of $\Sigma_n(U)$ or $\Sigma'_n(U)$ onto an element of $\Sigma_n(V)$ or $\Sigma'_n(V)$. Note that

$$\sigma_{\phi_1 \circ \phi_2} = \sigma_{\phi_2} \circ \sigma_{\phi_1}. \tag{2.88}$$

The reason why σ_ϕ is bijective is that, if λ is an eigenvalue of F', then $\phi(\lambda) \neq \infty$. Since $\infty \notin U$, this explains the surjectivity of σ_ϕ. □

One question is now: is σ_ϕ an isometry? Note that, if in ϕ, we multiply a, b, c and d by λ, this does not change the function ϕ, but multiplies σ'_ϕ by $1/\lambda$. We can always divide by $\sqrt{ad - bc}$, but this defines σ'_ϕ only up to a sign. Moreover, in the real case, $ad - bc$ could be negative.

Lemma 33
Assume that $\phi \in X_1$, and $|ad - bc| = 1$. Then σ'_ϕ is an isometry from Σ'_{dn} into itself.

Proof. If we apply the previous theorem with $U = \mathbb{U}$, we get the first result, namely that σ'_ϕ is a bijection from Σ'_{dn} into itself. According to $(2.85.b)$ we have

$$\mathcal{H}_2 = \frac{\lambda}{1 - z\overline{\alpha}} \mathcal{H}(c\frac{z-\alpha}{1-\overline{\alpha}z})$$

where $|\lambda|^2 = 1 - |\alpha|^2$. We have

$$\|\mathcal{H}_2\|^2 = \frac{1}{2\pi} \int_0^{2\pi} \frac{1-|\alpha|^2}{|1-e^{i\theta}\overline{\alpha}|^2} \left|\mathcal{H}(c\frac{e^{i\theta}-\alpha}{1-\overline{\alpha}e^{i\theta}})\right|^2 d\theta. \tag{2.89}$$

Let $e^{it} = \phi(e^{i\theta})$, so that $e^{it}dt = \phi'(e^{i\theta})e^{i\theta}d\theta$. This is

$$c\frac{e^{i\theta}-\alpha}{1-\overline{\alpha}e^{i\theta}}dt = c\frac{1-|\alpha|^2}{(1-e^{i\theta}\overline{\alpha})^2}e^{i\theta}d\theta$$

and simplifies to

$$dt = \frac{1-|\alpha|^2}{|1-e^{i\theta}\overline{\alpha}|^2}d\theta.$$

If we make this change of variables in (2.89), we get $\|\mathcal{H}_2\| = \|\mathcal{H}\|$. $\square$

Assume now that $\phi \in X_2$. Now σ_ϕ maps Σ_{cn} into Σ_{dn}. We can ask the same question. Consider

$$\phi_0(z) = \frac{1-z}{1+z}. \tag{2.90}$$

$$\phi(z) = \frac{z-1}{1+z}. \tag{2.91}$$

Since every element of X_2 is the composition of this ϕ and an element of X_1, according to (2.88), it suffices to answer the question for this ϕ. In fact, we consider:

$$\mathcal{H}'(s) = \frac{1}{1-s}\mathcal{H}(\frac{1+s}{1-s}), \qquad \mathcal{H}(z) = \frac{2}{z+1}\mathcal{H}'(\frac{z-1}{z+1}). \tag{2.92}$$

This is the equation $\mathcal{H} = \sigma_n^{-1}(\mathcal{H}')$, $\mathcal{H}'_2 = \sigma_n(\mathcal{H}_2)$ considered in the introduction of this section, and used by default in HYPERION. Note that the input to the rational approximation algorithm is not $\mathcal{H}$ (an element of H_2^-), but $G = \mathcal{H}(1/z)/x$. This gives

$$G(z) = \frac{2}{z+1}\mathcal{H}'(\phi_0(z)).$$

Note that, if $z = e^{i\theta}$, then $\phi_0(z) = i\omega$, with $1/\omega = \tan(\theta/2)$. A change of variable in the integral that gives the norm of $\mathcal{H}$ gives

$$\|\mathcal{H}\|^2 = \frac{1}{2\pi}\int_{-\infty}^{\infty} |\mathcal{H}'(i\omega)|^2 d\omega. \tag{2.93}$$

As said above, we can define $H(\mathbb{P})$ to be the set of all functions $\mathcal{H}'$ analytic in the half plane for which the previous integral converges. Thus σ'_ϕ is an isometry.

Note that $\mathcal{H}'$ has to be strictly proper. One way of solving the problem is just the following: we take $\mathcal{H}'$, subtract it constant term J (the value at infinity), convert it, consider its rational approximation, convert it back to a continuous time system, and then add the constant term J.

There is another solution: we can use σ_ϕ instead of σ'_ϕ. Then

$$\mathcal{H}'(s) = \mathcal{H}(\frac{1+s}{1-s}), \qquad \mathcal{H}(z) = \mathcal{H}'(\frac{z-1}{z+1}). \tag{2.94}$$

We have now

$$\|\mathcal{H}\|^2 = \frac{1}{2\pi}\int_{-\infty}^{\infty} |\mathcal{H}'(i\omega)|^2 \frac{d\omega}{1+\omega^2}. \tag{2.95}$$

If we adjust the constant c in (2.81), we get an isometry. Note that $\mathcal{H}$ is not strictly proper. If $\mathcal{H}$ is defined by (2.87.a), the value at infinity of $\mathcal{H}'$ is

$$J - H(I+F)^{-1}G.$$

What we do is now the following: we convert $\mathcal{H}'$ to a discrete time system, remove the constant term, approximate this, add the constant term, and convert back to a continuous time system. This mechanism is only partly implemented in HYPERION.

Chapter 3

The Schur algorithm

3.1 Schur functions

In this chapter, we study the set of inner matrices of given McMillan degree. The main section is section 3.5 where we shall show how to construct a matrix B of degree $n+k$ given a matrix A of degree n.

In this short section, we analyse the scalar case. We know that the mapping $(c, \omega_1, \dots, \omega_n) \to c\prod(z-\omega_i)/(1-z\overline{\omega_i})$ maps an element of $\mathbb{T}$ and n elements of $\mathbb{U}$ onto an inner function of degree n. The main trouble is that this function has no well-defined inverse.

If we consider polynomials instead of inner functions, the remedy is obvious: take as parameters the coefficients of the polynomials instead of the roots. If q is a polynomial, we define $s(q)$ to be $q(0)$ and $S(q) = (q - q(0))/z$. Let $q_0 = q$, and $q_{k+1} = S(q_k)$. Then $q = \sum s(q_k)z^k$.

The generalisation of this is the following. We define

$$S(f) = \frac{1}{z}\frac{f(z) - f(0)}{1 - \overline{f(0)}f(z)}.$$

Define $f_0 = f$, $f_{k+1} = S(f_k)$. The quantities $f_k(0)$ are called *the Schur parameters* of f.

Recall that a rational function, analytic in $\mathbb{U}$, is called inner if $f(\mathbb{T}) \subset \mathbb{T}$. We call it a Schur function if $f(\mathbb{T}) \subset \overline{\mathbb{U}}$. Hence inner functions are just special Schur functions. Note that the maximum modulus principle says that, if f is a Schur function, then $f(\mathbb{U}) \subset \overline{\mathbb{U}}$. Recall that equation (1.5) says that, if $|a| < 1$, then $(z-a)/(1-\overline{a}z)$ is in $\mathbb{T}$ (resp. $\mathbb{U}$) if and only if z is in $\mathbb{T}$ (resp. $\mathbb{U}$).

Lemma 34
Assume $h = \dfrac{zg+f}{1+z\overline{f}g}$. *In the case* $|f| \le 1$ *and* $|z| = 1$, *we have* $h \in \mathbb{T}$ *(resp.* $h \in \mathbb{U}$*) if and only if* $g \in \mathbb{T}$ *(resp.* $g \in \mathbb{U}$*).*

Proof. Let $a = -\overline{z}f$. Then $|h| = (g-a)/(1-g\overline{a})$. ☐

Consider now

$$C = \frac{A - \overline{y}}{1 - Ay} = \frac{(1-z\overline{\omega})(1-\omega)}{(z-\omega)(1-\overline{\omega})}\frac{B-\overline{y}}{1-By}, \quad \overline{y} = B(\omega). \qquad (*)$$

Lemma 35
Assume f rational, and $|f(0)| < 1$. Then f is a Schur function (resp. an inner function) if and only if $S(f)$ is a Schur function (resp. an inner function).

Moreover, if f is inner, of McMillan degree n, then $S(f)$ is inner, of McMillan degree $n-1$. In $(*)$, if one of A, B, C is inner and rational, then the others are inner and rational. If A has degree n, then C has degree n, and B has degree $n+1$.

Proof. Let $g = S(f)$. We have

$$f = \frac{zg + f(0)}{1 + zg\overline{f(0)}}.$$

If g is a Schur function, then $1 + zg\overline{f(0)}$ does not vanish on $\overline{\mathbb{U}}$ so that f is analytic in $\mathbb{U}$. On the other hand, if f is a Schur function, then g is analytic in $\mathbb{U}$. The previous lemma gives the first part of the proof.

The same argument shows that A is inner if and only if C is inner. Assume $A = cq/\tilde{q}$. Then

$$C = c\frac{q - \overline{cy}\tilde{q}}{\tilde{q} - cqy}.$$

Hence A and C have the same McMillan degree.

Assume now $g = cq/\tilde{q}$, q polynomial, $c \in \mathbb{T}$. Then

$$f = c\frac{zq + \overline{c}a\tilde{q}}{\tilde{q} + zcq\overline{a}}, \qquad a = f(0).$$

Since $\tilde{q} + zcq\overline{A}$ does not vanish on $\overline{\mathbb{U}}$, it follows that f is of degree $n+1$ in case g is of degree n. Evaluating at ω instead of at 0, shows that B is inner, and has degree $n+1$ if A has degree n. $\square$

Thus, if f is inner and rational, its Schur parameters are s_0, s_1, …, s_n, with $|s_n| = 1$ and $|s_i| < 1$ for $0 \le i < n$ (the maximum modulus principle says that, if f is a Schur function, and $|f(0)| = 1$, then f is constant). Instead of using $S(f)$, we shall use the transformation $(*)$, that gives A from B. Take

$$\beta_\omega(z) = \frac{(z-\omega)(1-\overline{\omega})}{(1-z\overline{\omega})(1-\omega)}. \tag{3.1}$$

Note that β_ω is a Blaschke product of degree one such that $\beta_\omega(1) = 1$. Now, equation $(*)$ is just

$$\frac{B - \overline{y}}{1 - By} = \beta_\omega \frac{A - \overline{y}}{1 - Ay}.$$

Expressing B gives

$$B = \frac{(\beta - |y|^2)A + (1-\beta)\overline{y}}{1 - \beta|y|^2 - (1-\beta)yA}. \tag{3.2}$$

Note that $\beta(1) = 1$ implies $A(1) = B(1)$.

In the matrix case, we shall extend relation (3.2) as

$$B = T_\Theta(A) = (\Theta_{11}A + \Theta_{12})(\Theta_{21}A + \Theta_{22})^{-1}. \tag{3.3}$$

One possibility for Θ is

$$\Theta = \begin{pmatrix} \Theta_{11} & \Theta_{12} \\ \Theta_{21} & \Theta_{22} \end{pmatrix} = \begin{pmatrix} \dfrac{\beta - |y|^2}{1-|y|^2} & \dfrac{(1-\beta)\overline{y}}{1-|y|^2} \\ \dfrac{(\beta-1)y}{1-|y|^2} & \dfrac{1-\beta|y|^2}{1-|y|^2} \end{pmatrix} = I - \frac{1-\beta}{1-|y|^2}\begin{pmatrix} 1 & -\overline{y} \\ y & -\overline{y}y \end{pmatrix}. \tag{3.4}$$

3.2 The Schur algorithm

The formula we use is a bit more general than (3.4). We use a vector y, a vector u and a number ω. We assume $\|u\| = 1$, $\|y\| < 1$ and $|\omega| < 1$. The matrix Θ is

$$\Theta = \begin{pmatrix} \Theta_{11} & \Theta_{12} \\ \Theta_{21} & \Theta_{22} \end{pmatrix} = I - \frac{1-\beta_\omega}{1-\|y\|^2}\begin{pmatrix} uu^* & -uy^* \\ yu^* & -yy^* \end{pmatrix}. \tag{3.5}$$

We shall introduce

$$b = (z-\omega)(1-\overline{\omega}), \qquad \tilde{b} = (1-z\overline{\omega})(1-\omega), \qquad \alpha = \frac{1-\beta_\omega}{1-\|y\|^2}. \tag{3.6}$$

Note that $\beta_\omega = b/\tilde{b}$, and that α is the factor that appears in (3.5).

3.2.1 Direct formulas

Introduce the vector $v = A^*u - y$. Then

$$B = T_\Theta(A) = (A - \alpha uv^*)(I - \alpha yv^*)^{-1}.$$

Using relations like $(uv^*)(yv^*) = (v^*y)(uv^*)$ we get

$$B = (A - \alpha uv^*)(I + \frac{\alpha yv^*}{1-\alpha v^*y}) = A + \frac{\alpha(Ay-u)v^*}{1-\alpha v^*y}. \tag{3.7.a}$$

$$B = \frac{A(1-\alpha\|y\|^2) + \alpha[Ayu^*A - u^*AyA + uy^* - Ayy^* - uu^*A]}{1+\alpha\|y\|^2 - \alpha u^*Ay}. \tag{3.7.b}$$

If A is inner and rational, we can write

$$A = \frac{D_A}{\tilde{q}_A}, \qquad A^{-1} = \frac{\tilde{D}_A}{q_A}, \qquad \det A = \epsilon_A \frac{q_A}{\tilde{q}_A}. \tag{3.8}$$

Now formulas (3.7) can be written as

$$B = \frac{D_B}{\tilde{q}_B}, \qquad B^{-1} = \frac{\tilde{D}_B}{q_B}, \qquad \det B = \epsilon_A \frac{q_B}{\tilde{q}_B} \tag{3.9}$$

with

$$\tilde{q}_B = (\tilde{b} - b\|y\|^2)\tilde{q}_A - (\tilde{b}-b)u^*D_Ay \tag{3.10.a}$$

$$D_B = (\tilde{b} - b\|y\|^2)D_A + (\tilde{b}-b)[\frac{D_Ayu^*D_A - u^*D_AyD_A}{\tilde{q}_A} + \tilde{q}_Auy^* - D_Ayy^* - uu^*D_A] \tag{3.10.b}$$

$$q_B = (b - \tilde{b}\|y\|^2)q_A + (\tilde{b}-b)y^*\tilde{D}_Au \tag{3.10.c}$$

$$\tilde{D}_B = (b - \tilde{b}\|y\|^2)\tilde{D}_A + (\tilde{b}-b)[\tilde{D}_Auu^* + yy^*\tilde{D}_A - yu^*q_A] + \frac{b-\tilde{b}}{q_A}[\tilde{D}_Auy^*\tilde{D}_A - y^*\tilde{D}_Au\tilde{D}_A]. \tag{3.10.d}$$

The fact that B has the desired form is a simple computation, the same is true for the formulas that give the inverse of B. Now, (3.7.a) and $\det(I + XY^*) = 1 + Y^*X$ give

$$\det B = \det A \frac{1-\alpha v^*A^{-1}u}{1-\alpha v^*y},$$

$$1 - \alpha v^*y = \frac{\tilde{q}_B}{\tilde{b}(1-\|y\|^2)\tilde{q}_A}, \qquad 1 - \alpha v^*A^{-1}u = \frac{q_B}{\tilde{b}(1-\|y\|^2)q_A}.$$

This shows the formula for $\det B$.

We know that, if $a = u^*Ay$, then $|a| \leq \|y\|$. Since β and y are in $\mathbb{U}$, relation (1.5) says that $(1-\beta)\|y\|/(1-\beta\|y\|^2)$ is also in $\mathbb{U}$, hence $t = (1-\beta)a/(1-\beta\|y\|^2) \in \mathbb{U}$. Since $1 - \alpha v^*y = (1 - \beta\|y\|^2)(1-t)/(1-\|y\|^2)$, this cannot vanish for $z \in \mathbb{U}$. Thus q_B is a stable polynomial.

3.2.2 Inverse formulas

Assume that B satisfies (3.9) and $B = T_\Theta(A)$. An easy computation says that equations (3.8) are satisfied if

$$q_A = (\tilde{b} - b\|y\|^2)q_B - (\tilde{b} - b)y^*\tilde{D}_B u, \tag{3.11.a}$$

$$\tilde{q}_A = (b - \tilde{b}\|y\|^2)\tilde{q}_B + (\tilde{b} - b)u^* D_B y, \tag{3.11.b}$$

$$D_A = (b - \tilde{b}\|y\|^2)D_B + (\tilde{b} - b)[\frac{u^* D_B y D_B}{\tilde{q}_B} - \frac{D_B y u^* D_B}{\tilde{q}_B} + D_B y y^* + u u^* D_B - \tilde{q}_B u y^*], \tag{3.11.c}$$

$$\tilde{D}_A = (\tilde{b} - b\|y\|^2)\tilde{D}_B + (b - \tilde{b})[\frac{y^* \tilde{D}_B u \tilde{D}_B}{q_B} - \frac{\tilde{D}_B u y^* \tilde{D}_B}{q_B} + \tilde{D}_B u u^* + y y^* \tilde{D}_B - q_B y u^*]. \tag{3.11.d}$$

3.2.3 Properties of the Schur algorithm

Lemma 36
Assume A inner, $B = T_\Theta(A)$. Then D_B and $\tilde{D}_B$ are polynomial matrices. In the same fashion, if B is inner, then D_A and $\tilde{D}_A$ are polynomial matrices.

Proof. Let $Y = \tilde{D}_A u y^* \tilde{D}_A - y^* \tilde{D}_A u \tilde{D}_A$. Let D_{ij} and Y_{ij} the entries of $\tilde{D}_A$ and Y. Then

$$Y_{ij} = \sum_{I,J} u_J \overline{y_I}(D_{iJ}D_{Ij} - D_{IJ}D_{ij}).$$

In other words, Y_{ij} is a linear combination of the minors of order 2 of $\tilde{D}_A$. These are multiple of q_A, hence $\tilde{D}_B$ is a polynomial. In the same fashion, D_B is a polynomial. □

In the case $p = 1$, we have $Y = 0$. In the case $p = 2$, we have

$$Y = (uy^* - y^*u)\det \tilde{D}_A.$$

In the special case $\epsilon_A = 1$, we have $\det \tilde{D}_A = q_A\tilde{q}_A$, $Y/q_A = \tilde{q}_A(uy^* - y^*u)$.

Theorem 20
If A is inner of McMillan degree n, then $B = T_\Theta(A)$ is inner, of McMillan degree $n+1$ and $B(\omega)^*u = y$.

Proof. Note that

$$B^*u - y = \frac{v}{1 - \overline{\alpha}y^*v}[1 + \overline{\alpha}(\|y\|^2 - 1)]$$

and that β vanishes at $z = \omega$, so that $\alpha(\omega) = 1/(1 - \|y\|^2)$. Hence $B(\omega)^*u = y$.

Since $\tilde{D}_B$ is a polynomial matrix, and q_B is a stable polynomial, it follows from (3.9) that B is inner. Since q_B and $\tilde{D}_B$ are of degree $n+1$, the McMillan degree of B must be $n+1$. □

Theorem 21
Assume that the property "B is inner and $B(\omega)^*u = y$" implies that $A = T_\Theta^{-1}(B)$ is inner. Then, if B is of degree $n+1$, A is of degree n, and $(z-\omega)(1-z\overline{\omega})$ divides each terms of the right-hand side of (3.10).

Proof. Assume A inner. Then $A = D/\tilde{q}$, $A^{-1} = \tilde{D}/q$ and $\det A = c_1 q/\tilde{q}$, where c_1 is some constant. Now, $\det A = c_2 q_A/\tilde{q}_A$. Since q and $\tilde{q}$ are coprime, there exists a polynomial r such that

$$q_A = rq \qquad \tilde{q}_A = c_3 r\tilde{q}$$

for some constant c_3. Since ω is a zero of $\tilde{q}_A$, but not of $\tilde{q}$, $z-\omega$ divides r, and $r = c_4(z-\omega)(1-\overline{\omega})$ for some polynomial c_4. Now, A has McMillan degree $n - \deg(c_4)$, so that B has McMillan degree $n+1-\deg(c_4)$, and c_4 is constant. □

Our objective now is to show the assumption of the theorem, namely to show that the condition $B(\omega)^*u = y$ is sufficient for $A = T_\Theta^{-1}(B)$ to be inner. We know that is is necessary. Obviously, it suffices to show that $(z-\omega)(1-z\overline{\omega})$ divides each terms of the left-hand side of (3.10). The non-trivial point is to show that it divides D_A. This is rather easy to show directly in case $z-\omega$ is coprime to q_B, but not so in the general case.

3.3 Reproducing kernel Hilbert spaces

Definition 4
A reproducing kernel Hilbert space is a Hilbert space H formed of functions f which are analytic in some set Ω with values in $\mathbb{C}^p$, together with a $p \times p$ matrix function $K(z,\omega)$, called the kernel, such that, for every $\omega \in \Omega$, every $c \in \mathbb{C}^p$, $K(\cdot,\omega)c$ is in H, and

$$\langle K(\cdot,\omega)c \,|\, f\rangle = c^* f(\omega). \tag{3.12}$$

Note that the mapping $f \to c^* f(\omega)$ is continuous. On the other hand, if this mapping is continuous, there exists a (unique) element $K_{\omega,c} \in H$ such that

$$\langle K_{\omega,c} \,|\, f\rangle = c^* f(\omega).$$

Obviously, $K_{\omega,c}$ is a linear function of c, so that there exists a function K such that $K_{\omega,c}(z) = K(z,\omega)c$, so that H is a reproducing kernel Hilbert space.

As a consequence, if H is a finite dimensional space, there exists always a reproducing kernel. For instance, if H contains all polynomials of degree at most n, and Φ_i is a polynomial of degree i, such that the set of all Φ_i is an orthonormal basis, then formula (2.66) gives the kernel.

The typical example of a space of infinite dimension is H^2, where $K(z,\omega) = 1/(1-z\overline{\omega})$, because (3.12) is just the Cauchy formula. On the other hand, if H' is a subset of H, which is a Hilbert space for the same inner product (in particular H' must be closed), then H' is also a reproducing kernel Hilbert space. We shall see that, if Q is inner, and H' the orthogonal of QH^2, then the kernel is

$$K_Q(z,\omega) = \frac{I - Q(z)Q(\omega)^*}{1-z\overline{\omega}}. \tag{3.13}$$

A consequence of (3.12), replacing f by $K(\cdot,\mu)d$ is

$$\langle K(\cdot,\omega)c_1 \,|\, K(\cdot,\mu)c_2\rangle = c_1^* K(\omega,\mu)c_2. \tag{3.14}$$

From this we deduce

$$\|\sum_{i=1}^{n} K(\cdot,\omega_i)c_i\|^2 = \sum_{ij} c_i^* K(\omega_i,\omega_j)c_j \tag{3.15}$$

and

$$\|K(\cdot,\omega)c\|^2 = c^* K(\omega,\omega)c. \tag{3.16}$$

Theorem 22
If Q is inner and rational, then $H(Q)$, the orthogonal of QH^2, is the reproducing kernel Hilbert space associated to K_Q. Its dimension is the McMillan degree of Q. This space is the linear span of elements of the form $K_Q(\cdot,\omega)c$, where $c \in \mathbb{C}^p$ and $\omega \in \Omega$, where Ω is an infinite subset of $\mathbb{U}$.

We prove here only the first claim, namely that $H(Q)$ is a reproducing kernel Hilbert space.

Proof. Since Q is inner, we have $\langle Qx \,|\, Qy\rangle = \langle x \,|\, y\rangle$ whenever x and y are in H^2. Now, the Cauchy formula says that Qx is orthogonal to $K_Q(\cdot,\omega)c$, whatever c and ω, if x is in H^2. From this we deduce (3.12). □

Corollary 7
If Q is inner and rational, then, for $\omega \in \overline{\mathbb{U}}$ we have $\|Q(\omega)^* u\| \leq \|u\|$. If Q is not constant, for any $\omega \in \mathbb{U}$, for almost all u, we have $\|Q(\omega)^* u\| < \|u\|$.

Note that this is the same as corollary 3.

Proof. Since Q is inner, we have $Q(z)Q(z)^* = I$ whenever $|z| = 1$, so that $\|Q(\omega)^* u\| = \|u\|$ if $|\omega| = 1$. Assume now $|\omega| < 1$. We have

$$(1 - |\omega|^2)\|K_Q(\cdot, \omega)u\|^2 = \|u\|^2 - \|Q(\omega)^* u\|^2.$$

Let $T = Q(\omega)Q(\omega)^*$. We have $T = T^*$ and $0 \leq u^* T u \leq \|u\|^2$. There exists an orthonormal basis $(e_i)_i$ such that $Te_i = \lambda_i e_i$, and $0 \leq \lambda_i \leq 1$. If, for at least one j, we have $\lambda_j < 1$, then $u^* T u < \|u\|^2$ whenever u is not orthogonal to e_j. Hence, for almost every u, we have $\|Q(\omega)^* u\| < \|u\|$.

On the other hand, if $\lambda_i = 1$ for every i, we have $\|Q(\omega)^* u\| = \|u\|$ for each u, and $\|K_Q(\cdot, \omega)u\| = 0$. This means $I = Q(z)Q(\omega)^*$, and Q is constant. □

Note: if $\|Q(\omega)^* u\| = \|u\|$, then $K_Q(\cdot, \omega)u = 0$, hence for every μ, $\|Q(\mu)^* u\| = \|u\|$.

Definition 5
A positive function P is a function $P(z, \omega)$, defined for z and ω in Ω, with values in $\mathbb{C}^{p \times p}$, such that, for every $\omega \in \Omega$, $P(\cdot, \omega)$ is analytic in Ω, and $\sum_{ij} c_i^* P(\omega_i, \omega_j) c_j \geq 0$, whenever $\omega_i \in \Omega$ and $c_i \in \mathbb{C}^p$.

Theorem 23
A reproducing kernel is a positive function. If P is a positive function on Ω, there exists a unique reproducing kernel Hilbert space with reproducing kernel P.

Proof. 1. If K is a reproducing kernel, then equation (3.15) says that K is a positive function.

2. Assume now that P is a positive function. Consider two vectors c_1 and c_2, two elements z and ω in Ω. We have

$$c_1^* P(z, z) c_1 + c_2^* P(\omega, \omega) c_2 + c_1^* P(z, \omega) c_2 + c_2^* P(\omega, z) c_1 \geq 0. \tag{3.17}$$

The first two terms in this expression are real, and $c_1^* P(z, \omega) c_2 + c_2^* P(z, \omega)^* c_1$ is also real. Taking the difference,

$$c_2^* [P(\omega, z) - P(z, \omega)^*] c_1$$

is real. Since this is true for every c_1 and c_2, the terms in brackets must be zero, hence $P(z, \omega)^* = P(\omega, z)$.

3. Let H_0 be the set of finite sums $x = \sum_i P(\cdot, \omega_i) c_i$, with $\omega_i \in \Omega$ and $c_i \in \mathbb{C}^p$. Each element of H_0 is analytic in Ω.

If $y = \sum_j P(\cdot, \omega_j') c_j'$, we define

$$f(x, y) = \sum_{ij} c_i^* P(\omega_i, \omega_j') c_j'. \tag{3.18}$$

We have $f(x, y) = \overline{f(y, x)}$, and $f(x, \lambda y) = \lambda f(x, y)$ whenever λ is a complex number.

Note that the representation $y = \sum_i P(\cdot, \omega_j') c_j'$ may not be unique. Nevertheless, $f(x, y)$ depends only on y since $f(x, y) = \sum_i c_i^* y(\omega_i)$. In the same fashion, f depends only on x, not on the particular choice of c_i and ω_i.

4. Define $\langle x \,|\, y \rangle_P = f(x, y)$ and $\|x\|_P = \sqrt{\langle x \,|\, x \rangle_P}$. This is a scalar product on H_0. In fact, we know $\langle x \,|\, x \rangle_P \geq 0$ for every x. Assume $\langle x \,|\, x \rangle_P = 0$. Then, for every vector u, every complex λ, we have $\|x + \lambda u\|_P^2 \geq 0$, and this implies $\langle x \,|\, u \rangle_P = 0$. Take $u = P(\cdot, \omega)c$. We get $c^* x(\omega) = 0$, hence $x = 0$.

5. Let H be the completion of H_0. Consider an element f of H, which is the limit of some sequence f_n of elements of H_0. Note that f_n is a Cauchy sequence, so that, for every ϵ, there exists N such that, if $n \geq N$ and $m \geq N$, $\|f_n - f_m\|_P \leq \epsilon$. For every x we have

$$|\langle x \,|\, f_n - f_m \rangle_P| \leq \epsilon \|x\|_P.$$

Let C be any compact subset of Ω. Take $x = P(\cdot,\omega)c$. Since $P(\omega,\omega)$ is continuous, there exists M such that $\|x\|_P \le M$, if $\omega \in C$. Hence

$$|c^* f_n(\omega) - c^* f_m(\omega)| \le \epsilon M.$$

This shows that $c^* f_n$ converges uniformly on every compact subset of Ω. The limit F_c is an analytic function. Moreover,

$$\langle P(\cdot,\omega)c \,|\, f\rangle_P = F_c(\omega).$$

This equation shows that there exists an analytic function F such that $F_c = c^* F$,

$$\langle P(\cdot,\omega)c \,|\, f\rangle_P = c^* F(\omega).$$

If we identify now f and F, H is a Hilbert space of analytic functions on Ω that satisfies the required conditions.

6. Unicity. If H' is a reproducing kernel Hilbert space with kernel P, it is obvious that it contains H_0, hence H. Moreover, if f is an element of H' orthogonal to every element of H, it is orthogonal to every $P(\cdot,\omega)c$, hence $c^* f(\omega) = 0$, and $f = 0$, so that $H = H'$. $\square$

In what follows, we shall use the following result: Assume that H is a finite dimensional vector space, which is the linear span of all $P(\cdot,\omega)c$. Assume that $\langle x \,|\, y\rangle$ is a hermitian form that satisfies equation (3.14) with K replaced by P. Then $\langle x \,|\, y\rangle$ is a scalar product on H.

3.4 J-inner functions

3.4.1 Introduction

In what follows, J will be a signature matrix, a matrix such that $J^2 = I$ and $J = J^*$. Define $J_+ = (J+I)/2$, $J_- = (J-I)/2$. We have $J_+^2 = J_+$, $J_-^2 = -J_-$, $J_+J_- = J_-J_+ = 0$, $J_+^* = J_+$, $J_-^* = J_-$.

These relations say that $\mathbb{C}^p$ is the direct, orthogonal sum of the kernels of the orthogonal projectors J_+ and $-J_-$. Said otherwise, there exists a matrix A with $A^{-1} = A^*$ and $J = A\begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}A^*$. We could, without loss of generality, assume that A is the identity matrix. Define

$$\langle x \,|\, y\rangle_J = \langle x \,|\, Jy\rangle = \langle Jx \,|\, y\rangle. \tag{3.19}$$

If Ax has components x_1 and x_2, we get $\langle x \,|\, x\rangle_J = x_1^2 - x_2^2$. Unless J is the identity matrix, $\langle x \,|\, y\rangle_J$ defines a Hermitian form which is not positive definite.

Definition 6
A matrix A is called J-unitary in case $J = AJA^*$, it is called J-contractive in case $J - AJA^*$ is positive, semi-definite.

By definition, if A is J-unitary, then

$$\langle Ax \,|\, Ay\rangle_J = \langle x \,|\, y\rangle_J. \tag{3.20}$$

We shall consider in what follows matrices Θ that are J-unitary on $\mathbb{T}$. If $\Theta \in H^\infty$, x and y are in H^2, then the previous relation is true, for the H^2 scalar product. A J-inner matrix will satisfy this condition, plus some others. These additional conditions are justified as follows. Introduce first

$$K_\Theta(z,\nu) = \frac{J - \Theta(z)J\Theta(\nu)^*}{1 - z\overline{\nu}} \tag{3.21}$$

and

$$K_\Sigma(z,\nu) = \frac{I - \Sigma(z)\Sigma(\nu)^*}{1 - z\overline{\nu}}. \tag{3.22}$$

If Σ is defined by the equivalent formulas

$$\Sigma = (J_+ - \Theta J_-)^{-1}(\Theta J_+ - J_-) = (J_+\Theta + J_-)(J_+ + J_-\Theta)^{-1}, \tag{3.23.a}$$

then Θ can be obtained from

$$\Theta = (J_+\Sigma + J_-)(J_+ + J_-\Sigma)^{-1} = (J_+ - \Sigma J_-)^{-1}(\Sigma J_+ - J_-). \tag{3.23.b}$$

In the case

$$J = \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}, \qquad \Theta = \begin{pmatrix} \Theta_{11} & \Theta_{12} \\ \Theta_{21} & \Theta_{22} \end{pmatrix}$$

we have

$$\Sigma = \begin{pmatrix} \Theta_{11} - \Theta_{12}\Theta_{22}^{-1}\Theta_{21} & -\Theta_{12}\Theta_{22}^{-1} \\ \Theta_{22}^{-1}\Theta_{21} & \Theta_{22}^{-1} \end{pmatrix},$$

so that Σ is defined if Θ_{22} is invertible. Let $F(z) = J_+ - \Theta(z)J_-$. Then

$$K_\Theta(z,\omega) = F(z)K_\Sigma(z,\omega)F(\omega)^*. \tag{3.24}$$

Assume Θ rational. If Θ is J-unitary on $\mathbb{T}$, then Σ is unitary on $\mathbb{T}$. If Σ is inner, then K_Σ, hence K_Θ are positive functions. On the other hand, if Θ is J-contractive on $\mathbb{U}$ (this is a weaker condition), then Σ will be bounded, hence inner, and K_Θ will be a positive function. Note that, if $\Theta\begin{pmatrix} 0 \\ x \end{pmatrix} = \begin{pmatrix} y \\ 0 \end{pmatrix}$, the condition $\langle \Theta u \,|\, \Theta u\rangle_J = \langle u \,|\, u\rangle_J$ gives $-\|x\|^2 = \|y\|^2$, hence $x = y = 0$. Thus, if Θ is J-unitary on $\mathbb{T}$, Θ_{22} is invertible on $\mathbb{T}$, so that Σ is always defined.

Definition 7
We say that Θ is J-inner if it is in H^∞, is J-unitary almost everywhere on $\mathbb{T}$, and K_Θ is a positive function.

We introduce now two vector spaces

$$H(\Theta) = \{\sum_i K_\Theta(z,\nu_i)c_i, c_i \in \mathbb{C}^p, \nu_i \in \Omega\} \tag{3.25.a}$$

$$H_\Theta = \{x \in H^2, \forall y \in H^2, \langle x \,|\, J\Theta y\rangle = 0\}. \tag{3.25.b}$$

3.4.2 Basic properties

Lemma 37
If Θ is rational and J-unitary on $\mathbb{T}$, then $H(\Theta)$ has finite dimension.

Proof. Since Θ is J-unitary on $\mathbb{T}$ we have

$$J - \Theta(1/\overline{\omega})J\Theta(\omega)^* = 0 \tag{3.26}$$

whenever ω is in $\mathbb{T}$, but since Θ is rational, this is true almost everywhere, and we get

$$\Theta(\omega)^{-1} = J\Theta(1/\overline{\omega})^*J. \tag{3.27}$$

From this, we deduce

$$K_\Theta(z,\omega)J\Theta(\mu)/\mu = \frac{\Theta(z) - \Theta(\mu)}{z - \mu} \qquad \mu = 1/\overline{\omega}. \tag{3.28}$$

Write $\Theta = N/q$, where q is a polynomial of degree $\leq n$, N a matrix of polynomials of degree $\leq n$. Let

$$P(z,\mu) = \frac{N(z)q(\mu) - N(\mu)q(z)}{z - \mu}.$$

This is a polynomial of degree $< n$, hence can be written as

$$P(z,\mu) = \sum_{i=0}^{n-1} z^i P_i(\mu),$$

hence

$$K_\Theta(z,\omega) J\Theta(\mu) q(\mu)/\mu = \sum_{i=0}^{n-1} \frac{z^i}{q(z)} P_i(\mu). \tag{3.29}$$

Let Ω be an open set, and $\nu_1, \dots, \nu_m$ be the points of Ω on which $\Theta(1/\overline{\nu})$ is not defined, or not invertible. For any x in $H(\Theta)$, we can write

$$x = \sum_i K_\Theta(z,\omega_i) c_i + \sum_{j=1}^{m} K_\Theta(z,\nu_j) d_j$$

for some constant vectors c_i, d_j, and elements $\omega_i \in \Omega$, with $\omega_i \neq \nu_j$. Let $\mu_i = 1/\overline{\omega}_i$, $e_i = \mu_i \Theta(\mu_i)^{-1} J c_i / q(\mu_i)$, so that (3.29) gives

$$x = \sum_{i=0}^{n-1} \frac{z^i}{q(z)} (\sum_j P_i(\mu_j) e_j) + \sum_{j=1}^{m} K_\Theta(z,\nu_j) d_j.$$

This shows that $H(\Theta)$ is included in a finite dimensional space. □

The dimension of this space is a priori $(n+m)p$, but we shall see later that is it is less than that. This lemma has a converse.

Lemma 38
If the space $H(\Theta)$ is a finite dimensional vector space then Θ is rational and J-unitary on $\mathbb{T}$.

Proof. Consider a set of generators of $H(\Theta)$, of the form $K_\Theta(z,\omega_i)c_i$. Take ω in $\mathbb{U}$, different from all ω_i, for which Θ is defined. There exists some matrices M_i (depending on ω alone) such that

$$K_\Theta(z,\omega) = \sum_{i=1}^{n} K_\Theta(z,\omega_i) M_i. \tag{3.30}$$

This is equivalent to

$$J\left(\frac{I}{1-z\overline{\omega}} - \sum_i \frac{M_i}{1-z\overline{\omega}_i}\right) = \Theta(z) J \left(\frac{\Theta(\omega)^*}{1-z\overline{\omega}} - \sum_i \frac{\Theta(\omega_i)^* M_i}{1-z\overline{\omega}_i}\right). \tag{3.31}$$

Write this as $JA = \Theta(z)JB$. Note that A and B are rational, so that $\det A$ is a rational function. By construction, it has a singularity at $z = 1/\overline{\omega}$, so that $\det A$ is not identically zero, $\det B$ is not identically zero, hence $\Theta = JAB^{-1}J$ is rational.

Multiply relation (3.31) by $1 - z\overline{\omega}$, and take the limit $z \to 1/\overline{\omega}$. If Θ is analytic at $z = 1/\overline{\omega}$ we get

$$J - \Theta(1/\overline{\omega}) J \Theta(\omega)^* = 0 \tag{3.32}$$

Since Θ is rational, this relation is true whenever it is defined, hence almost everywhere. □

From now on, we shall assume $\Theta \in H^\infty$. Now $K_\Theta(z,\omega)$ is defined for z, $\omega \in \mathbb{U}$. The set Ω that appears in the definition of $H(\Theta)$ will be $\mathbb{U}$. We also assume Θ J-unitary on $\mathbb{T}$. If x and y are in H^2, then $J\Theta x$ and $\Theta J y$ are in H^2 and

$$\langle J\Theta x \,|\, \Theta J y\rangle = \langle x \,|\, y\rangle. \tag{3.33}$$

This can also be written as

$$\langle \Theta x \,|\, \Theta y\rangle_J = \langle x \,|\, y\rangle_J.$$

Let c be a constant vector, $\omega \in \mathbb{U}$. Applying the Cauchy formula twice gives

$$\langle K_\Theta(z,\omega)c \,|\, J\Theta x\rangle = \langle \frac{c}{1-z\overline{\omega}} \,|\, \Theta x\rangle - \langle \frac{\Theta(\omega)^* c}{1-z\overline{\omega}} \,|\, x\rangle = 0. \tag{3.34}$$

Thus, if c_1 and c_2 are constant vectors, ω and μ are in $\mathbb{U}$, we obtain a relation similar to (3.14):

$$\langle K_\Theta(z,\omega)c_1 \,|\, JK_\Theta(z,\mu)c_2\rangle = c_1^* K_\Theta(\omega,\mu)c_2. \tag{3.35}$$

Lemma 39
If Θ is rational and J-inner, then $H(\Theta) = H_\Theta$.

Proof. Since $\Theta \in H^\infty$, the space H_Θ is well defined. Equation (3.34) says that it contains $H(\Theta)$. The note after theorem 23 tells us that $\langle x \,|\, y\rangle_J$ is a scalar product on $H(\Theta)$ (recall that K_Θ is a positive function, and $H(\Theta)$ is of finite dimension). Let $(e_i)_i$ be an orthonormal basis of $H(\Theta)$ for this scalar product. Take $x \in H_\Theta$ and write

$$x = \sum_{i=1}^{n} \langle e_i \,|\, x\rangle_J e_i + y. \tag{3.36}$$

Now $e_i \in H(\Theta)$ implies $y \in H_\Theta$, hence $\langle Jy \,|\, \Theta(z)J\Theta(\omega)^* c/(1-z\overline{\omega})\rangle = 0$. Since e_i is an orthonormal basis of $H(\Theta)$, $\langle e_i \,|\, y\rangle_J = 0$, hence $\langle t \,|\, y\rangle_J = 0$ whenever $t \in H(\Theta)$, hence $\langle Jy \,|\, K_\Theta(z,\omega)c\rangle = 0$. Thus

$$\langle Jy \,|\, K_\Theta(z,\omega)c - \frac{\Theta J\Theta(\omega)c}{1-z\overline{\omega}}\rangle = 0.$$

By definition of K_Θ, this is $\langle Jy \,|\, Jc/(1-z\overline{\omega})\rangle = 0$, and the Cauchy formula says that $c^* y(\omega) = 0$. Thus $y = 0$ and $x \in H(\Theta)$. □

Theorem 24
If Θ is J-inner and rational, then $H(\Theta)$ is left shift invariant and finite dimensional.

Proof. Take x in H^2.

$$\langle \frac{K_\Theta(z,\omega) - K_\Theta(0,\omega)}{z} c \,|\, J\Theta x\rangle = \langle K_\Theta(z,\omega)c \,|\, zJ\Theta x\rangle - \langle K_\Theta(0,\omega)c \,|\, zJ\Theta x\rangle.$$

Both terms are zero, so that $R_0(K_\Theta)$ is in H_Θ, hence in $H(\Theta)$. □

3.4.3 J-inner functions and left shift invariant spaces

Assume Θ rational and J-inner. The previous theorem says that $H(\Theta)$ is left shift invariant and finite dimensional, and theorem 10 describes these spaces; in this section, we show that it gives a characterisation of Θ.

We know that $H(\Theta)$ is the set of elements of the form $A(I-zC)^{-1}v$, $v \in \mathbb{C}^n$, where (A, C) is an observable pair.

Fix $\omega \in \Omega$ and $c \in \mathbb{C}^n$. Chose v such that

$$K_\Theta(z,\omega)c = A(I-zC)^{-1}v.$$

Take $w \in \mathbb{C}^n$ and let $y = A(I-zC)^{-1}w$. Now (3.35) says

$$\langle K_\Theta(z,\omega)c \,|\, Jy\rangle = c^* y(\omega). \tag{3.37}$$

Define

$$P = \sum_{k=0}^{\infty} C^{*k} A^* J A C^k. \tag{3.38}$$

Lemma 40
P is a positive definite matrix.

Proof. Note that elements of $H(\Theta)$ are analytic in $\mathbb{U}$, and have no singularity on $\mathbb{T}$. This means that eigenvalues of C are in $\mathbb{U}$, and the series that defines P is convergent. If we replace in (3.37) $K_\Theta(z,\omega)c$ by $A(I-zC)^{-1}v$ and expand, we get $v^*Pw = c^*y(\omega)$ hence

$$Pv = (I-\overline{\omega}C^*)^{-1}A^*c, \tag{3.39}$$

and

$$K_\Theta(z,\omega) = A(I-zC)^{-1}P^{-1}(I-\overline{\omega}C^*)^{-1}A^*. \tag{3.40}$$

The matrix P is invertible, because, if $Pw=0$, then $c^*y(\omega)=0$ whatever c and ω, so that y is identically zero, and $w=0$. Now, since any vector d can be written as $\sum_i (I-\overline{\omega}_i C^*)^{-1}A^*c_i$, we get

$$\sum_{ij} c_i^* K_\Theta(\omega_i,\omega_j)c_j = d^*Pd. \tag{3.41}$$

Now $d^*Pd \geq 0$ because K_Θ is positive. □

Lemma 41
If (A,C) is an observable pair, and eigenvalues of C are in $\mathbb{U}$, then the equation

$$A^*JA = P - C^*PC \tag{3.42}$$

has a unique solution P. If $J=J^*$ then $P=P^*$. If J is positive definite, so is P. Moreover, P satisfies equation (3.38).

Proof. Let P be a solution. If $T_k = \sum_{i\leq k} C^{*i}A^*JAC^i$, then $T_k = P - C^{*\,k+1}PC^{k+1}$. Hence T_k converges to P, hence (3.38). Since (3.42) is a set of n^2 linear equations in n^2 variables, it has a unique solution, provided that the associated homogeneous system has only $P=0$ as solution. But, if $J=0$, we have $T_k=0$, hence $P=0$.

Thus, if J is positive definite, we get $x^*Px \geq 0$ for each vector x. In the case where this is zero, we get $AC^ix=0$ for each i, so that the observability of (A,C) tells us $x=0$. □

Note: take

$$J = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \qquad A = \begin{pmatrix} \tan\phi\tan\theta & 1/\cos\phi \\ \tan\theta/\cos\phi & \tan\phi \end{pmatrix}, \qquad C = \begin{pmatrix} 0 & 0 \\ 1/\cos\phi & 0 \end{pmatrix}.$$

Then $A^*JA = I - C^*C$. Hence P can be positive, even though A^*JA is not positive.

From the relation

$$K_\Theta(z,a) = \frac{J-\Theta(z)J\Theta(a)^*}{1-z\overline{a}}$$

we get

$$\Theta(z) = [I-(1-z\overline{a})K_\Theta(z,a)J]\Theta(a)$$

in the case $|a|=1$, because $\Theta(a)$ is J-unitary. Hence

$$\Theta(z) = [I-(1-z\overline{a})A(I-zC)^{-1}P^{-1}(I-\overline{a}C^*)^{-1}A^*J]\Theta(a), \tag{3.43.a}$$

$$a\in\mathbb{T}, \Theta(a) \text{ is J-unitary.} \tag{3.43.b}$$

Theorem 25
If Θ is J-inner and rational, there exists an observable pair (A,C), where the eigenvalues of C are in $\mathbb{U}$. such that the matrix P defined by (3.42) *is positive definite, and Θ is defined by* (3.43)*, K_Θ by* (3.40)*. On the other hand, given an observable pair (A,C), if the matrix P defined by* (3.42) *is positive definite, then the matrix Θ defined by* (3.43) *is J-inner. The McMillan degree of Θ is the dimension of $H(\Theta)$, the size of the matrix C.*

Note that this completes the proof of theorem 22

Proof. 1. Note that the condition $a \in \mathbb{T}$ implies that both members of (3.43.a) evaluate to $\Theta(a)$ in case $z = a$. The non-trivial part of the theorem is to prove that Θ satisfies (3.43). Everything else is a straightforward computation.

2. Assume that (3.43) is satisfied. Let's compute K_Θ. Write

$$X(z) = (1 - z\overline{a})A(I - zC)^{-1}P^{-1}(I - \overline{a}C^*)^{-1}A^*.$$

Then

$$K_\Theta(z, \omega) = \frac{X(z) + X(\omega)^* - X(z)JX(\omega)^*}{1 - z\overline{\omega}}.$$

Write $Y(z) = (I - zC)^{-1}P^{-1}(I - \overline{a}C^*)^{-1}$. Then

$$K_\Theta(z, \omega) = \frac{(1 - z\overline{a})(1 - \overline{\omega}a)}{1 - z\overline{\omega}} A^*Y(z)\left[\frac{Y^{-1}(z)}{1 - \overline{\omega}a} + \frac{Y^{-1}(\omega)^*}{1 - z\overline{a}} - A^*JA\right]Y(\omega)^*A^*.$$

If we use equation (3.42), the term in brackets becomes $\lambda(I - \overline{a}C^*)P(I - aC)$, where $\lambda = (1 - z\overline{\omega})/(1 - z\overline{a})(1 - \overline{\omega}a)$ is the inverse of the factor that appears in K_Θ. Thus, K_Θ has the form (3.40).

3. Since P is positive definite, K_Θ is a positive function. Since the eigenvalues of C are in $\mathbb{U}$, the function Θ is analytic in $\mathbb{U}$. Since K_Θ is defined on $\mathbb{T}$, the definition (3.21) of K_Θ says that Θ is J-unitary on $\mathbb{T}$. Hence Θ is J-inner.

4. Let's now compute the McMillan degree of Θ. We may assume $a = 1$, and ignore the factor $\Theta(a)$. Applying (3.27) gives

$$\Theta(z)^{-1} = I - (z - 1)A(I - C)^{-1}P^{-1}(zI - C^*)^{-1}A^*J. \tag{3.44}$$

Write $B = P^{-1}(I - C^*)^{-1}A^*$. Using the relation $(z - 1)(zI - F)^{-1} = I + (F - I)(zI - F)^{-1}$ gives

$$\Theta(z)^{-1} = I - B^*A^*J - B^*(C^* - I)(zI - C^*)^{-1}A^*J. \tag{3.45}$$

This equation has the form $\Theta^{-1} = H(zI - F)^{-1}G + K$, it is a proper, stable, rational transfer function. We know that, if the realization is minimal, the McMillan degree of Θ^{-1}, which is the same as the McMillan degree of Θ, is the dimension of the matrix F, which is C^* in the equation above, and the dimension of the matrix is, because of the observability of (A, C), the dimension of $H(\Theta)$.

5. Let's show that the realization (3.45) is minimal. Since (A, C) is observable, it is clearly reachable. Let's show that it is observable. We consider x such that $C^*x = \lambda x$ and $B^*(C^* - I)x = 0$, and we want to show that x is zero. Notice first that these equations give $B^*(1 - \lambda)x = 0$, and, since $\lambda \neq 1$, $B^*x = 0$. Let $y = (I - C)^{-1}P^{-1}x$. Then $x = P(I - C)y$, $Ay = 0$, and $C^*x = \lambda x$. Now, equation (3.42) says $Py = C^*PCy$. Applying this relation twice gives $x = (C^* - I)PCy$ and $C^*x = (C^* - I)Py$. Now $C^*x = \lambda x$ and the invertibility of $(C^* - I)P$ gives $y = \lambda Cy$. Hence, if $\lambda = 0$, we have $y = 0$; otherwise y is an eigenvector of C, but $Ay = 0$ and the observability of (A, C) says now $y = 0$. Thus $x = 0$. □

The next lemma will play a role in the proof of the Potapov theorem.

Lemma 42
If u is a non-zero vector such that $u^*\Theta(\omega) = 0$ for some $\omega \in \mathbb{U}$, then $u^*Ju > 0$.

Proof. The assumption on u implies $u^*K_\Theta(\omega, \omega)u = u^*Ju/(1 - |\omega|^2) \geq 0$. If $u^*K_\Theta(\omega, \omega)u = 0$, then $(I - \overline{\omega}C^*)^{-1}A^*u = 0$, because P is positive definite. This implies $A^*u = 0$, and $u^*\Theta(z) = u^*\Theta(\omega) = 0$. Since this is true for every z, taking $z = 1$ gives $u = 0$, because $\Theta(1)$ is invertible. □

3.4.4 The theorem of Potapov

What we are trying to show here is that every J-inner rational function is the product of J-inner functions of degree one. The next lemma says that the product of two J-inner matrices is J-inner, and gives a converse.

Lemma 43
If Θ_1 and Θ_2 are two J-inner matrices, then the product $\Theta_1\Theta_2$ is J-inner, and the McMillan degree is the sum of the McMillan degrees of Θ_1 and Θ_2. On the other hand, if Θ and Θ_1 are J-inner and rational, then $\Theta_1^{-1}\Theta$ is J-inner if it is analytic in $\mathbb{U}$.

Proof. 1. If $\Theta = \Theta_1\Theta_2$ we have

$$K_\Theta(z,\omega) = K_{\Theta_1}(z,\omega) + \Theta_1(z)K_{\Theta_2}(z,\omega)\Theta_1(\omega)^*. \tag{3.46}$$

If each Θ_i is J-unitary on $\mathbb{T}$, so is the product. Moreover, if K_{Θ_i} are positive functions, then K_Θ is a positive function. This shows the first claim.

2. We assume now that Θ and Θ_1 are J-inner, and Θ_2 is analytic in $\mathbb{U}$. Then Θ_2 is J-unitary on $\mathbb{T}$. We have to show that K_{Θ_2} is a positive function.

3. We have $H(\Theta_1) \subset H(\Theta)$. This is because, if $y \in H(\Theta_1)$, Jy is orthogonal to every $\Theta_1 t$ ($t \in H^2$), hence to every $\Theta_1\Theta_2 t$. Thus $y \in H_\Theta$, and $y \in H(\Theta)$.

4. Let $x_{\omega c} = \Theta_1(z)K_{\Theta_2}(z,\omega)\Theta_1(\omega)^*c$. Let $x_1 = K_\Theta(z,\omega)c$ and $x_2 = K_{\Theta_1}(z,\omega)c$. Then $x_1 \in H(\Theta)$ and $x_2 \in H(\Theta)$ so that $x_{\omega c} = x_1 - x_2 \in H(\Theta)$. Let $t = JK_\Theta(z,\omega')c'$. Then $\langle Jt \,|\, x_{\omega c}\rangle = \langle Jt \,|\, x_1 - x_2\rangle = c'^* x_{\omega c}(\omega')$.

Now, $x_{\omega c} \in \Theta_1 H^2$, hence is orthogonal to $JK_{\Theta_1}(z,\omega')c'$. Thus

$$\langle x_{\omega c} \,|\, x_{\omega' c'}\rangle_J = c^* x_{\omega' c'}(\omega).$$

Hence, if $y = \sum \Theta_1(z)K_{\Theta_2}(z,\omega_i)\Theta_1(\omega_i)^*c_i$ we have

$$\langle y \,|\, Jy\rangle = \sum_{ij} c_i^*\Theta_1(\omega_i)K_{\Theta_2}(\omega_i,\omega_j)\Theta_1(\omega_j)^*c_j. \tag{3.47}$$

5. Since $y \in H(\Theta)$, we have $\langle y \,|\, y\rangle_J \geq 0$. Take some numbers ω_i such that $\Theta_1(\omega_i)$ is invertible, and $d_i = \Theta_1(\omega_i)^*c_i$. Now (3.47) says

$$\sum d_i^* K_{\Theta_2}(\omega_i,\omega_j)d_i \geq 0.$$

This is true, by continuity, even if $\Theta(\omega_i)$ is not invertible.

6. Every element of $H(\Theta)$ can be written as $x + \Theta_1 y$, where $x \in H(\Theta_1)$ and $y \in H(\Theta_2)$. Such a decomposition is obviously unique (if x is in $H(\Theta_1)$ and $\Theta_1 H^2$, then x is zero). We showed that x and Jy are orthogonal. Thus the dimension of $H(\Theta)$ is the sum of the dimensions of $H(\Theta_1)$ and $H(\Theta_2)$. □

The question is now: what are the J-inner matrices of degree one? In equation (3.43), C is a 1×1 matrix, hence a scalar. Write $C = \overline{\gamma}$. Equation (3.42) says now

$$P = \frac{A^*JA}{1 - |\gamma|^2}.$$

Equation (3.43), with $a = 1$ and $\Theta(1) = I$, gives now

$$\Theta = I + \frac{(z-1)(1-|\gamma|^2)}{(1-z\overline{\gamma})(1-\gamma)}\frac{AA^*J}{A^*JA}.$$

This can be written as

$$\Theta = I - (1 - \beta_\gamma)\frac{AA^*J}{A^*JA}. \tag{3.48}$$

Recall that β_γ is the normalised Blaschke product of degree one with root γ. We have two equivalent formulas for K_Θ.

$$K_\Theta(z,\omega) = \frac{1-|\gamma|^2}{(1-z\overline{\gamma})(1-\overline{\omega}\gamma)}\frac{AA^*}{A^*JA}. \tag{3.49.a}$$

$$K_\Theta(z,\omega) = \frac{1-\beta_\gamma(z)\overline{\beta_\gamma(\omega)}}{1-z\overline{\omega}}\frac{AA^*}{A^*JA}. \tag{3.49.b}$$

In the case where $J = \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}$, we can write $A = \begin{pmatrix} u \\ v \end{pmatrix}$, and the condition $A^*JA > 0$ (which is equivalent to $P > 0$) is $\|y\| < \|u\|$. Multiplying A by a scalar constant does not change Θ, so that we can always assume $\|u\| = 1$. Thus, Θ has the form (3.5).

Assume now

$$\Theta_1 = I - (1-\beta_\omega)uu^*J/(u^*Ju), \tag{3.50.1}$$

$$\Theta_2 = I - (1-\beta_{\omega'})vv^*J/(v^*Jv). \tag{3.50.2}$$

Lemma 44
If Θ is J-inner, $u^*J\Theta(\omega) = 0$, then $\Theta_1^{-1}\Theta$ is J-inner.

Proof. Apply the previous two lemmas. The first one says that $u^*Ju > 0$, so that Θ_1 is J-inner. Since $\Theta_1^{-1} = I - (1-1/\beta_\omega)uu^*J/(u^*Ju)$, the matrix $\Theta_1^{-1}\Theta$ is analytic in $\mathbb{U}$. The second lemma says that it is J-inner. □

Now, the hard point is the following: under which condition is the product $\Theta_1\Theta_2$ real. Introduce the following quantities

$$U = \frac{uu^*J}{u^*Ju}, \qquad V = \frac{vv^*J}{v^*Jv}, \qquad \omega = a+ib, \qquad c = \frac{2b}{1-a^2-b^2}. \tag{3.51}$$

Lemma 45
In the case ω is not real, the product $\Theta_1\Theta_2$ is real if and only if $\omega' = \overline{\omega}$, and

$$\Im[U+V] = 0 \tag{3.52.a}$$

$$\Im[ic(U-V) - 2UV] = 0. \tag{3.52.b}$$

Proof. Since the determinant of $\Theta_1\Theta_2$ is $\beta_\omega\beta_{\omega'}$, the condition $\omega' = \overline{\omega}$ is necessary. We have a second condition, which is now

$$\Im[\frac{-U}{1-\beta_{\overline{\omega}}} + \frac{-V}{1-\beta_\omega} + UV] = 0.$$

We have

$$\frac{1}{1-\beta_\omega} = 1 - \frac{ic}{2} + \frac{|1-\omega|^2}{(1-z)(1-|\omega|^2)} - \frac{2-\omega-\overline{\omega}}{2(1-|\omega|^2)}, \tag{3.53}$$

from which we deduce (3.52.a), then (3.52.b). □

Lemma 46
If $\omega' = \overline{\omega}$, the two conditions v proportional to $J\Theta_1(\overline{\omega})^*J\overline{u}$ and u proportional to $\Theta_2(\omega)\overline{v}$ are equivalent.

Proof. Note that $1/(1-\beta_\omega(\overline{\omega})) = 1 - ic$. The first condition is now

$$v = \lambda_v(1 + ic - \frac{uu^*J}{u^*Ju})\overline{u}.$$

Define

$$\lambda = \frac{u^* J \overline{u}}{u^* J u}. \tag{3.54}$$

We have now

$$v = \lambda_v[(1+ic)\overline{u} - \lambda u]. \tag{3.55.a}$$

This gives

$$v^* J v = |\lambda_v|^2 (1 + c^2 - |\lambda|^2) u^* J u,$$

and

$$\mu = \frac{v^* J \overline{v}}{v^* J v} = -\frac{\overline{\lambda \lambda_v}}{\lambda_v}.$$

Now, we can deduce u from (3.55.a) and get

$$u = \lambda_u[(1+ic)\overline{v} - \mu v] \tag{3.55.b}$$

with $1/\lambda_u = \lambda_v(1 + c^2 - |\lambda|^2)$. If u has this form, it is now obvious that u is proportional to $\Theta_2(\omega)\overline{v}$. □

Note: Assume that Θ is real and $u^* J \Theta(\omega) = 0$. Then $\overline{u}^* J \Theta(\overline{\omega}) = 0$. Let $\Theta = \Theta_1 \Theta'$. We know that Θ' is J-inner. The condition of the lemma is $v^* J \Theta'(\overline{\omega}) = 0$. We know that this condition implies $v^* J v > 0$. It says that $\Theta_2^{-1}\Theta'$ is J-inner. In the case where J is the identity matrix, the equation $v^* J v > 0$ holds in any case, because $|\lambda| \le 1$, $c > 0$ (because ω is not real), hence $1 + c^2 - |\lambda|^2 > 0$.

Lemma 47
Conditions (3.52) *and* (3.55) *are equivalent.*

Proof. 0. If (3.55) is true, then an easy check gives (3.52). Hence we shall assume that (3.52) is true.

1. Assume that w is real and orthogonal to u. Then $w^* U = 0$, and (3.52) implies $w^* V = 0$, hence $w^* v = 0$ (recall that c is non-zero). Assume now that w is orthogonal to u and $\overline{u}$. Then $w + \overline{w}$ and $i(w - \overline{w})$ are real and orthogonal to u, hence to v. In other words, v is a linear combination of u and $\overline{u}$.

2. Introduce $U_0 = \overline{u} u^* J / u^* J u$. If

$$v = \alpha u + \beta \overline{u} \tag{3.56}$$

we get

$$V = \frac{|\alpha|^2 U + \alpha \overline{\beta U_0} + \beta \overline{\alpha} U_0 + |\beta|^2 \overline{U}}{|\alpha|^2 + |\beta|^2 + \overline{\alpha}\beta\lambda + \alpha\overline{\beta\lambda}}, \tag{3.57}$$

so that (3.52.a) becomes

$$(2|\alpha|^2 + \overline{\alpha}\beta\lambda + \alpha\overline{\beta\lambda})(U - \overline{U}) = 0. \tag{3.58}$$

3. Assume that u and $\overline{u}$ are linearly dependent. Then $u = \gamma\overline{u}$ for some γ. Since $|\gamma| = 1$, we can write $\gamma = e^{2i\phi}$ and $ue^{-i\phi}$ is real. Hence U is real. But v is proportional to u, so that $V = U$. Now equations (3.52) are satisfied. From now on, we shall assume that u and $\overline{u}$ are linearly independent. Then $U \ne \overline{U}$. Now, if $\alpha = 0$, v is proportional to $\overline{u}$ so that $V = \overline{U}$, $U + V$ and $ic(U - V)$ are real. Hence (3.52) is equivalent to $\Im UV = 0$. But

$$U\overline{U} = \lambda \frac{u \overline{u}^* J}{u^* J u}.$$

If $\lambda \ne 0$, this cannot be real (it would give a linear dependency relation). Note that $\lambda = 0$ implies $\alpha = 0$.

4. From now on, we assume $U \ne \overline{U}$, $\alpha \ne 0$ and $\lambda \ne 0$. Write $t = -\lambda/\alpha$, $\gamma = \beta t$. Now (3.58) says $\gamma + \overline{\gamma} = 2$, so that $\gamma = 1 + id$ for some real d. We have now

$$V = \frac{|\lambda|^2 U + (1+d^2)\overline{U} - 2\Re[\lambda(1-id)\overline{U_0}]}{1 + d^2 - |\lambda|^2}, \tag{3.59.a}$$

$$UV = \frac{id}{1 + d^2 - |\lambda|^2}[-|\lambda|^2 U + \lambda(1-id)\overline{U_0}]. \tag{3.59.b}$$

Now (3.52.b) says

$$\frac{2(c-d)}{1+d^2-|\lambda|^2}\Re[\lambda(1-id)\overline{U_0}-|\lambda|^2U]=0.$$

If we multiply this equation by u^*J we get $c=d$ or

$$\lambda(1-id-|\lambda|^2)\overline{u}^*+|\lambda|^2idu^*=0.$$

Now, since $\lambda\neq 0$, this gives a linear relation between u and $\overline{u}$ (unless $d=0$, $\lambda=1$, case where V is undefined). Hence $c=d$. □

Corollary 8
Assume that Θ is a J-inner rational matrix. Then $\det\Theta=\epsilon q/\tilde{q}$ for some stable polynomial q. If Θ is not constant, then q is not constant. Assume that $q(\omega)=0$.

There exists a matrix Θ_1 such that Θ_1 and $\Theta_1^{-1}\Theta$ are J-inner, Θ_1 has McMillan degree one, and $\det\Theta_1=\beta_\omega$. Assume that Θ and J are real. The same result is true, for some real Θ_1 if ω is real; if ω is not real, we can find Θ_1 real, of McMillan degree two, and $\det\Theta_1=\beta_\omega\beta_{\overline{\omega}}$, with $\Theta_1^{-1}\Theta$ J-inner.

Proof. Equation (3.27) says that $\det\Theta(z)$ is in $\mathbb{T}$ if $z\in\mathbb{T}$, so that it is inner, and has the form $\epsilon\tilde{q}/q$ for some stable q. If the determinant is constant, equation (3.45) says that Θ is constant.

Assume now that $q(\omega)=0$. Then $\Theta(\omega)$ is not invertible. There exists u such that $u^*\Theta(\omega)=0$. The result is now obvious from lemma 44. Note that if Θ and J are real, if ω is also real, we can chose u real, so that Θ_1 will be real. Assume ω not real. Then if v is defined by (3.55.a), the matrix $\Theta_1\Theta_2$ (defined by (3.50)) is real, and $(\Theta_1\Theta_2)^{-1}\Theta$ is J-unitary. □

Theorem 26 (Potapov)
Let Θ be a J-inner rational matrix. Its determinant has the form $\epsilon q/\tilde{q}$, where q is stable and monic. Write it as $q(\Theta)$.

If $q=q_1q_2\cdots q_n$, $\alpha_i\in\mathbb{T}$, A_i is J-unitary and constant ($0\leq i\leq n+1$), there exits n J-inner matrices Θ_i such that

$$\Theta=A_0\Theta_1\Theta_2\cdots\Theta_nA_{n+1}\tag{3.60}$$

and $\Theta_i(\alpha_i)=A_i$, $q(\Theta_i)=q_i$. Among the $n+2$ matrices A_i, $n+1$ can be chosen arbitrary, the last one is defined by the previous equation.

If q_i, A_i, α_i, Θ and J are real, then Θ_i can be chosen real.

Proof. 1. Assume first that q_i is irreducible (in the complex case, it is of degree one, in the real case, it is of degree one or two). Assume that

$$\Theta=\Theta_1\Theta_2\cdots\Theta_kX,\tag{3.61.a}$$

where Θ_i is J-inner, $q(\Theta_i)=q_i$, and X is J-inner. Then $q(X)=q_{k+1}\cdots q_n$. The corollary says that we can obtain a similar formula with $k+1$ instead of k provided that $k<n$. It says that X is constant if $k=n$. Thus, by induction, we obtain (3.60).

2. Assume that q_i is any polynomial. Let's factor it. Since the product AB of two J inner matrices is J-inner and $q(AB)=q(A)q(B)$, we obtain (3.60) in this case also. Note that, if q_i is constant, its associated matrix is the identity matrix.

3. We have

$$\Theta=B_1^{-1}(B_1\Theta_1B_2^{-1})(B_2\Theta_2B_3^{-1})\cdots(B_n\Theta_nB_{n+1}^{-1})(B_{n+1}X).\tag{3.61.b}$$

If we want $B_i\Theta_i(\alpha_i)B_{i+1}^{-1}=A_i$, we can write

$$B_{i+1}=A_i^{-1}B_i\Theta_i(\alpha_i)\text{ or }B_i=A_iB_{i+1}\Theta_i(\alpha_i)^{-1}.$$

Assume that A_0, A_1, …, A_{k-1}, A_{k+1}, …, A_{n+1} are fixed. Then we take $B_1=A_0^{-1}$, and use the first relation to determine B_2, B_3, up to B_k. After that, we determine $B_{n+1}=A_{n+1}X^{-1}$, and use the second relation to get B_n, B_{n-1}, down to B_{k+1}. All these matrices are obviously J-unitary. □

3.5 The Schur algorithm

In this section, unless stated otherwise, are matrices will be rational, A and B will be of size $p \times p$, while Θ has size $2p \times 2p$ and partitioned as

$$\Theta = \begin{pmatrix} \Theta_{11} & \Theta_{12} \\ \Theta_{21} & \Theta_{22} \end{pmatrix}. \tag{3.62}$$

The matrix J will be of size $2p \times 2p$, $J = \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}$.

The Schur formulas (3.3)

$$B = T_\Theta(A) = (\Theta_{11}A + \Theta_{12})(\Theta_{21}A + \Theta_{22})^{-1}$$

are written as

$$V = \Theta_{11}A + \Theta_{12}, \quad U = \Theta_{21}A + \Theta_{22}, \quad U' = \Theta_{11} - B\Theta_{21}, \quad V' = B\Theta_{22} - \Theta_{12}, \tag{3.63}$$

$$T_\Theta(A) = VU^{-1} \qquad T_\Theta^{-1}(B) = U'^{-1}V'. \tag{3.64}$$

Let $\tau_B = (I \quad -B)$. If $D_B(\Theta)$ is the matrix U' above, then the previous formulas can be written as

$$\tau_B \Theta = D_B(\Theta)\tau_A \Leftrightarrow B = T_\Theta(A). \tag{3.65}$$

Introduce the following kernels:

$$K_B(z,\nu) = \frac{I - B(z)B(\nu)^*}{1 - \overline{z}\nu}, \quad K_\Theta(z,\nu) = \frac{J - \Theta(z)J\Theta(\nu)^*}{1 - z\overline{\nu}}. \tag{3.66}$$

If K_A is defined like K_B, then (3.63) and (3.64) say

$$U'K_AU'^* = K_B - \tau_B K_\Theta \tau_B^*. \tag{3.67}$$

Introduce also

$$K_A^*(z,\nu) = \frac{I - A(z)^*A(\nu)}{1 - \overline{z}\nu}, \qquad K_\Theta^*(z,\nu) = \frac{J - \Theta(z)^*J\Theta(\nu)}{1 - \overline{z}\nu}. \tag{3.68}$$

An easy computation says that $B = T_\Theta(A)$ is equivalent to

$$X(\mu,\nu) = \frac{U(\mu)^*U(\nu) - V(\mu)^*V(\nu)}{1 - \overline{\mu}\nu} = (\, A(\mu)^* \quad I \,)\, K_\Theta^*(\mu,\nu) \begin{pmatrix} A(\nu) \\ I \end{pmatrix} + K_A^*(\mu,\nu). \tag{3.69.a}$$

$$Y(\mu,\nu) = \frac{U'(\mu)U'(\nu)^* - V'(\mu)V'(\nu)^*}{1 - \mu\overline{\nu}} = K_B(\mu,\nu) - (\, I \quad -B(\mu) \,)\, K_\Theta(\mu,\nu) \begin{pmatrix} I \\ -B(\nu)^* \end{pmatrix}. \tag{3.69.b}$$

Note that (3.69.b) is nothing else than (3.67).

Lemma 48
Assume that Θ is J-inner. If A is inner, X is positive, then VU^{-1} is inner. If B is inner, Y is positive, then $U'^{-1}V'$ is inner.

Proof. The proof is the same for both claims. Consider the quantity

$$c^*[U(\mu)^*U(\nu) - V(\mu)^*V(\nu)]c. \tag{$*$}$$

If Θ is J-inner and A is inner, this quantity is zero if $\mu = \nu \in \mathbb{T}$.

We assume now that this quantity is ≥ 0 whenever $\mu = \nu \in \mathbb{U}$. This is weaker than just saying that X is positive. Hence

$$\|U(\mu)c\| \geq \|V(\mu)c\|. \tag{3.70}$$

Recall that

$$\Theta(\mu)\begin{pmatrix}A(\mu)c\\ c\end{pmatrix}=\begin{pmatrix}U(\mu)c\\ V(\mu)c\end{pmatrix}.$$

If $U(\mu)c=0$, then $V(\mu)c=0$ by (3.70). The previous equation then says that $\Theta(\mu)$ is not invertible. Since $\Theta(\mu)$ has not a constant zero determinant (it is invertible for $\mu\in\mathbb{T}$), we deduce that the determinant of U is not identically zero, so that $B=T_\Theta(A)$ is defined for almost every μ. But equation (3.70) can be written as

$$\|B(\mu)c\|\le\|c\|.$$

This says that B is bounded. Since B is rational, it cannot have a pole in $\mathbb{U}$ (nor can it have a pole on $\mathbb{T}$). Hence B is in H^2. $\square$

If we assume that Θ is J-inner, then the conditions $AA^*=I$ and $BB^*=I$ are equivalent. What this lemma says is just that, if some kernels are positive, then matrices A or B are analytic in $\mathbb{U}$. The easy case is to show that if A is inner then B is inner. We already know this, and X is positive. The non trivial point is the reverse: under which condition is T_Θ^{-1} inner.

Theorem 27
Assume Θ J-inner, $B=T_\Theta(A)$. The following conditions are equivalent

- *A is inner*
- *B is inner and $\tau_B x\in H(B)$ whenever $x\in H(\Theta)$.*

Moreover, the McMillan degree of B is the sum of the McMillan degrees of A and Θ.

Proof. 1. Assume first that B is inner, and $\tau_B x\in H(B)$ whenever $x\in H(\Theta)$. Note that, if $x=\begin{pmatrix}x_1\\ x_2\end{pmatrix}$, then $\tau_B x=x_1-Bx_2$. This is orthogonal to Bx_2, since it is in $H(B)$. But $\langle Bx_2\,|\,Bx_2\rangle=\langle x_2\,|\,x_2\rangle$, so that

$$\|\tau_B x\|^2=\|x_1\|^2-\|x_2\|^2=\|x\|_J^2. \tag{3.71}$$

Remember that $\|x\|_J$ is the norm in $H(\Theta)$.

2. Let τ be the mapping from $H(\Theta)$ into $H(B)$ that associates $\tau_B x$ to x. This is an isometry, according to (3.71). Let τ^* be its dual map. If x and y are in $H(B)$ we have

$$\langle\tau\tau^*x\,|\,\tau\tau^*y\rangle=\langle\tau^*x\,|\,\tau^*y\rangle_J=\langle\tau\tau^*x\,|\,y\rangle. \tag{2.72.a}$$

Hence

$$\langle\tau\tau^*x\,|\,(I-\tau\tau^*)y\rangle=0, \tag{3.72.b}$$

and

$$\langle x\,|\,(I-\tau\tau^*)x\rangle=\|(I-\tau\tau^*x)\|^2\ge 0. \tag{3.72.c}$$

Equation (3.72.b) says that the images of $\tau\tau^*$ and $I-\tau\tau^*$ are direct and orthogonal. In particular

$$H(B)=\mathrm{Im}\tau\tau^*\oplus\mathrm{Im}(I-\tau\tau^*). \tag{3.73}$$

Since all our spaces are finite dimensional, the injectivity of τ implies that τ^* is surjective, so that τ and $\tau\tau^*$ have the same image. We pretend that

$$H(B)=\tau_B H(\Theta)\oplus D_B(\Theta).H(A). \tag{3.74}$$

3. Take now $x=K_B(\cdot,\nu)c_1$ and $y\in H(\Theta)$. The properties of K_B and K_Θ say

$$\langle x\,|\,\tau y\rangle=\langle K_B(\cdot,\nu)c_1\,|\,\tau_B y\rangle=c_1^*\tau_B(\nu)y(\nu)=\langle K_\Theta(\cdot,\nu)\tau_B(\nu)^*c_1\,|\,y\rangle_J,$$

hence

$$\tau^* K_B(\cdot,\nu)c_1 = K_\Theta(\cdot,\nu)\tau_B(\nu)^* c_1.$$

Now equation (3.67) gives

$$(I - \tau\tau^*)x = U'(z)K_A(z,\nu)U'(\nu)^* c_1. \tag{3.75}$$

Let y be this quantity. It is in $H(\Theta)$, so that $\langle x \,|\, y\rangle = c_1^* y(\nu)$. Now (3.72.$c$) gives

$$c_1^* U'(\nu)K_A(\nu,\nu)U'(\nu)^* c_1 \geq 0.$$

4. This condition says that A is inner. Since U' is almost everywhere invertible, equation (3.75) says that $\mathrm{Im}(I - \tau\tau^*) = U' H(A)$, hence (3.74).

From (3.74), we get: the dimension of $H(B)$ is the sum of the dimensions of $H(\Theta)$ and $H(A)$, thus the McMillan degree of B is the sum of the McMillan degrees of A and Θ.

5. We use induction to show the converse, namely that if A is inner, then $B = T_\Theta(A)$ is inner (we already know this) and τ maps $H(\Theta)$ into $H(B)$. There is nothing to show if Θ has McMillan degree zero, because K_Θ is zero, and $H(\Theta)$ has dimension zero.

Otherwise, we can always write $\Theta = \Theta_1\Theta_2$, where Θ_1 has the form (3.5) and Θ_2 has smaller degree than Θ. Let $C = T_{\Theta_2}(A)$ and $B = T_{\Theta_1}(C)$. An easy computation says that $B = T_\Theta(A)$.

According to (3.46) we have

$$K_\Theta(z,\omega) = K_{\Theta_1}(z,\omega) + \Theta_1(z)K_{\Theta_2}(z,\omega)\Theta_1(\omega)^*.$$

Now (3.65) gives

$$\tau_B(z)K_\Theta(z,\omega) = \tau_B(z)K_{\Theta_1}(z,\omega) + D_B(\Theta_1)\tau_C(z)K_{\Theta_2}(z,\omega)\Theta_1(\omega)^*. \tag{3.76}$$

By induction

$$\tau_C K_{\Theta_2}\Theta_1(\omega)^* \in H(C).$$

The proof is complete if we show

$$H(B) = \tau_B H(\Theta_1) \oplus D_B(\Theta_1)H(C). \tag{3.77}$$

If we compare with (3.74), it is enough to show that $\tau_B x \in H(B)$, whenever $x \in H(\Theta_1)$ (i.e. prove the claim in the case of dimension one). But $H(\Theta_1)$ has dimension one, and $\tau_B x$ is

$$\lambda \frac{u - B(z)y}{1 - z\overline{\omega}}$$

for some complex λ, and

$$\langle \frac{u - B(z)y}{1 - z\overline{\omega}} \,|\, Bf\rangle = (u^* B(\omega) - y^*)f(\omega).$$

Now, this is zero, according to Theorem 20. □

We shall demonstrate three lemmas, in order to prove the next theorem, that was used in the previous chapter.

Lemma 49
Assume $|\alpha| < 1$. The mapping $\omega \to \beta_\omega(\alpha)$ is a bijection from $\mathbb{U}$ to itself, and its inverse is a C^∞ function.

Proof. We have $x = \beta_\omega(\alpha)$ if and only if

$$\frac{1 - x\alpha}{1 - x}|\omega|^2 - \omega - \alpha\overline{\omega} + \frac{\alpha - x}{1 - x} = 0, \qquad \omega \neq 1. \tag{3.78}$$

Introduce $\gamma = (x-\alpha)/(1-x)$ and $X = (1+\alpha+\gamma)|\omega|^2 - \omega - \alpha\overline{\omega} - \gamma$. The previous equation is $X = 0$. Note that $x \in \mathbb{U}$ if and only if $\Re(1+x)/(1-x) > 0$. Let $A = (\gamma - \overline{\gamma}\alpha)/(1-|\alpha|^2)$. Then

$$A = \frac{x - |x|^2 - \alpha + \alpha|x|^2 + |\alpha|^2 - |\alpha|^2 x}{(1-x)(1-\overline{x})(1-|\alpha|^2)}. \tag{3.79}$$

Hence

$$\Re(A+1/2) = \frac{(1-|x|^2)(1-\alpha)(1-\overline{\alpha})}{2(1-x)(1-\overline{x})(1-|\alpha|^2)} = \frac{1}{2}\Re\frac{1+x}{1-x}/\Re\frac{1+\alpha}{1-\alpha}. \tag{3.80}$$

Since $|\alpha| < 1$, we have $x \in \mathbb{U}$ if and only if $\Re(A+1/2) > 0$. Now

$$X - \alpha\overline{X} = (1-|\alpha|^2)[(A+1)|\omega|^2 - A - \omega]. \tag{3.81}$$

Since $|\alpha| \neq 1$, we have $X = 0$ if and only if $X - \alpha\overline{X} = 0$, hence $x = \beta_\omega(\alpha)$ is equivalent to

$$(A+1)|\omega|^2 = A + \omega \qquad \omega \neq 1. \tag{3.82}$$

If we write $\omega = u + iv$, it is easy to check that the previous equation has two solutions in ω, namely $\omega = 1$ and $\omega = -A/(\overline{A}+1)$. Moreover, for the second solution, we have $\omega \in \mathbb{U}$ if and only if $\Re A > -1/2$. □

Lemma 50
If Q_0 is inner, of McMillan degree $n > 0$, there exists Q near Q_0, Θ and A such that $Q = T_\Theta(A)$. We can chose A such that there exists $\alpha \in \mathbb{U}$, such that $A(\alpha)$ is invertible, $Q(\alpha)$ is not invertible.

Proof. We can write $Q_0 = T_{\Theta_0}(A)$, where Θ_0 is defined by u, ω_0, and y, and ω_0 is real. Let Q be defined as $T_\Theta(A)$, where Θ is defined by y, ω and u. If ω is near ω_0, then Q is near Q_0.

Take ω real. The Schur formulas give

$$q = (1-\omega)[q_A(z - \omega - (1 - z\omega)\|y\|^2) + y^*\tilde{D}_A u(1 - z\omega - z + \omega)]. \tag{3.83}$$

We want to find α such that $q_A(\alpha)$ is not zero, and $q(\alpha)$ is zero. Since q is of degree n and q_A of degree $n-1$, this is possible, unless every root of q_A is a root of q, said otherwise, q divides q_A^n. Now, the remainder R of the division of q_A^n by q is a rational function of ω, because of equation (3.83). If this is not identically zero, we can chose ω arbitrarily near to ω_0 for which this condition is true.

Finally, $q/(1-\omega)$ has a limit as $\omega \to 1$, and this limit has 1 as a root, which is not a root of q_A. Hence R is not identically zero. □

Lemma 51
If $Q = T_\Theta(A)$, $V_0 \in H^2$, $V_0Q^{-1} \in H^2$, $v = A^{-1}(\alpha)u - y$, then $V_0(\alpha)v = 0$, provided that $A(\alpha)$ is invertible, and $Q(\alpha)$ is not invertible.

Proof. If $V_0 = VQ$, the Schur formulas say

$$(\tilde{b} - b)\frac{q_A}{\tilde{q}_A}V_0(A^{-1}u - y)(u^* - y^*A^{-1})D_A = (VA - V_0)q. \tag{3.84}$$

Evaluate this at $z = \alpha$. One assumption is $q(\alpha) = 0$, so that this is zero. We also assume that $\alpha \in \mathbb{U}$, so that $\tilde{q}_A$ is not zero at $z = \alpha$. The second assumption is that $q_A(\alpha) \neq 0$. Hence D_A is invertible, because $D_A\tilde{D}_A = q_A\tilde{q}_A$. We get $(\tilde{b} - b)V_0(A^{-1}u - y)(u^* - y^*A^{-1}) = 0$. It is easy to check that $\tilde{b} - b$ and $u^* - y^*A^{-1}$ cannot be zero when evaluated at $z = \alpha$. □

Theorem 28
Let Q_0 be an inner matrix of McMillan degree $n > 0$ and V_0 a matrix with entries in H^2 such that, whenever Q is near Q_0, then V_0Q^{-1} has entries in H^2. Then $V_0 = 0$.

Proof. Chose Q_1 near Q_0, u_0, y_0, ω_0 such that $Q_1 = T_{\Theta_0}(A)$, where Θ_0 is defined by (u_0, y_0, ω_0), and α_0 such that A is invertible at α_0 and Q_1 is not. Let $v_0 = A^{-1}(\alpha_0)u_0 - y_0$.

Chose v near v_0, and α near α_0, and $q_A(\alpha) \neq 0$. Let $y = A^{-1}(\alpha)u_0 - v$. Consider $Q = T_\Theta(A)$, where Θ is defined by u_0, y and ω. Now α is a root of q provided that

$$\beta_\omega(\alpha) = \frac{\|y\|^2 - y^* A^{-1}(\alpha)u_0}{1 - y^* A^{-1}(\alpha)u_0}. \tag{3.85}$$

We know that the right hand side of this is near the same quantity with indices 0 on y and α, which is zero, because $Q_1(\alpha_0)$ is not invertible. One of the previous lemmas says now that there exists ω near ω_0 for which this equation is true.

Finally, we know that Q is near Q_1, hence near Q_0, and we can apply the last lemma, which says $V_0(\alpha)v = 0$. Since this is true for almost all α near α_0, and all v near v_0, it follows that V_0 is identically zero. □

3.6 The manifold of inner functions

Let $\mathcal{I}_n^p$ be the set of rational inner matrices of degree n and size p. This is a C^∞ manifold, see for instance [1]. We consider also the set $\mathcal{I}_n^p(1)$ formed of matrices Q such that $Q(1) = I$.

3.6.1 Case of dimension one

Let S_n be the set of monic stable polynomials. Any inner function is of the form $cq/\tilde{q}$, where $q \in S_n$ and $c \in U$. This gives us three ways to parameterise inner functions, namely, by c and the coefficients of q, or by the Schur parameters (see introduction of this Chapter), or by specialising the algorithm described below to the case $n = 1$.

Let $\mathcal{I}_{n0}$ be the set of inner functions of the form $q/\tilde{q}$. Topologically, S_n and $\mathcal{I}_{n0}$ are equivalent. Let's first state the following lemma:

Lemma 52
S_n is an open subset of $\mathbb{C}^n$, its closure is the (compact) set of polynomials with roots in $\overline{\mathbb{U}}$.

Note that, if $q = \sum q_i z^i$, and $q_n = 1$ we can consider q as an element of $\mathbb{C}^n$, namely the vector with coordinates $(q_0, q_1, \dots, q_{n-1})$, which has norm $\|q\|_1 = \sqrt{\sum_{i=0}^{n-1} |q_i|^2}$. Since $\|q - q'\|_1 = \|q - q'\|$, the topology of S_n is the same, whether we consider S_n as a subset of $\mathbb{C}^n$ or a subset of H^2 (of course, S_n is not open in H^2). The lemma is just an application of a more general result that says that the roots of q are continuous functions of the coefficients of q.

Theorem 29
The mapping $q \to q/\tilde{q}$ is continuous, with continuous inverse, from S_n onto $\mathcal{I}_{n0}$.

Proof. A little computation show that, if p and q are in S_n, we have

$$\|\frac{p}{\tilde{p}} - \frac{q}{\tilde{q}}\|^2 = 2\Re\langle \frac{p}{\tilde{p}} \,|\, \frac{(p-q)(\tilde{p}-\tilde{q})}{\tilde{p}\tilde{q}} \rangle.$$

This is essentially because

$$\langle \frac{p}{\tilde{p}} \,|\, \frac{p}{\tilde{q}} \rangle = \langle \frac{p}{\tilde{p}} \,|\, \frac{q}{\tilde{p}} \rangle = 1.$$

There exists a polynomial X_p, such that, for every polynomial A of degree at most $2n$ we have

$$\langle \frac{p}{\tilde{p}} \,|\, \frac{A}{\tilde{p}^2} \rangle = \langle X_p \,|\, A \rangle.$$

Moreover, there exists X_{pq} such that

$$\langle \frac{p}{\tilde{p}} \,|\, \frac{A}{\tilde{p}^2} \rangle - \langle \frac{p}{\tilde{p}} \,|\, \frac{A}{\tilde{p}\tilde{q}} \rangle = \langle X_{pq} \,|\, A \rangle$$

and this is bounded by $\|X_{pq}\| \|A\|$. Since X_{pq} is a continuous function of q at $q = p$, and vanishes for $q = p$, it is small for q near p. Take now $A = (p - q)(\tilde{p} - \tilde{q})$. We have $\|A\| = \|p - q\|^2$. Hence

$$\| \frac{p}{\tilde{p}} - \frac{q}{\tilde{q}} \|^2 \leq 2\|X_p\| \|p - q\|^2 + o(\|p - q\|^2).$$

This shows that the mapping $p \to p/\tilde{p}$ is continuous.

The converse is not so easy. Assume that q_k is a sequence of elements of S_n such that $q_k/\tilde{q}_k$ converges to $p/\tilde{p}$. We want to show that q_k converges to p. Since q_k is the closure of S_n which is compact, it is enough to show that, if q_k converges, the limit is p. However, if the limit is q, and q is stable, we know that $q_k/\tilde{q}_k$ converges to $q/\tilde{q}$, hence $p = q$. But if q is not stable, we do know nothing. Hence the following argument.

Let α be a root of p, of multiplicity n_α. Let

$$g_i = \frac{i! z^i}{(1 - z\overline{\alpha})^{i+1}}.$$

The Cauchy formula says that $\langle g_i \,|\, f \rangle$ is the i-th derivative of f at α. This is a continuous function of f. Assume $f = s/\tilde{s}$. It is easy to show by induction on i that we have

$$f^{(i)} = \sum_{j=0}^{i} a_j \frac{s^{(j)}}{\tilde{s}^{i+1-j}}$$

for some quantities a_i, which are polynomial functions of s and $\tilde{s}$, and the derivatives of $\tilde{s}$. Moreover $a_i = 1$. If we take $s = p$, and evaluate at α, we get zero if $i < n_\alpha$. Hence

$$\lim_{k\to\infty} \sum_{j=0}^{i} a_{jk}(\alpha) \frac{q_k^{(j)}(\alpha)}{\tilde{q}_k^{i+1-j}(\alpha)} = 0.$$

By induction, $\lim_k q_k^{(i)}(\alpha) = 0$. This is true because quantities $a_{jk}(\alpha)$ have a limit, $\tilde{q}_k(\alpha)$ has a non-zero limit, and $a_i = 1$.

Now $\lim_k q_k^{(i)}(\alpha) = q^{(i)}(\alpha) = 0$, so that $(z - \alpha)^{n_\alpha}$ divides q. Hence p divides q. Since p and q have the same degree we get $p = q$. ☐

It is now interesting to analyse the boundary of S_n. Obviously, q is in the boundary of S_n if q is the product of $q_1 \in S_k$ and q_2, where the roots of q_2 are in $\mathbb{T}$. Hence, this boundary is the union of objects homeomorphic to $\mathbb{C}^{n-k} \times \mathbb{T}^k$, $1 \leq k \leq n$. If we consider only real polynomials, the situation is a bit more complex: we may have two roots, which are complex conjugate. Hence the boundary is the union of quantities homeomorphic to $S_{n-k-2p} \times I_k \times I^p$. Here I_k is the set of polynomials of degree k with roots in $\mathbb{U}$, this is ± 1, so that it is a finite set of $k + 1$ elements. Moreover I is the set of polynomials with two complex conjugate roots of modulus one, hence the set of polynomials of the form $z^2 - az + 1$, $|a| < 2$. It is homeomorphic to $\mathbb{R}$. Thus, the boundary of S_n is the union of copies of $\mathbb{R}^k$ $0 \leq k < n$.

For instance, if $n = 2$, the boundary is $3\mathbb{R}^0 + 3\mathbb{R}^1$ (recall that S_2 is the triangle defined by $u < 1$, $v < u + 1$ and $-v < u + 1$). If $n = 3$, the boundary is $4\mathbb{R}^0 + 5\mathbb{R}^1 + 3\mathbb{R}^2$. In fact, if $q = z^3 + az^2 + bz + c$, then q is stable if

$$S_1(a, b, c) = 1 + a + b + c > 0$$

$$S_2(a, b, c) = 1 - a + b - c > 0$$

$$S_3(a, b, c) = 1 + ac - b - c^2 > 0.$$

Note that $S_1 = 0$ and $S_2 = 0$ are planes. In the decomposition $\partial S_3 = 4\mathbb{R}^0 + 5\mathbb{R}^1 + 3\mathbb{R}^2$, the three quantities $\mathbb{R}^2$ are parts of the surfaces $S_i = 0$, the four quantities $4\mathbb{R}^0$ are the polynomials $z^3 \pm 1$, $(z \pm 1)^3$, and the fives quantities $\mathbb{R}^1$ are the intersections $S_i \cup S_j$. These are pieces of straight lines. In fact, the intersection of S_1 and S_2 is the set of polynomials of the form $(z^2-1)(z+a)$, while the intersection of S_1 and S_3 is the set of polynomials of the form $(z-c)(z-1)^2$ or of the form $(z-1)(z^2+(1-a)z+1)$.

The non trivial point in this decomposition is how these piece are glued together (recall that the boundary of S_n is compact).

3.6.2 The case of dimension 2

Let Q be a 2×2 inner matrix. It can be written as $Q = D/\tilde{q}$, where D is a matrix of polynomials, and q is a stable polynomial. Consider

$$D = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{3.86}$$

Since $D\tilde{D} = q\tilde{q}$, we get

$$a\tilde{a} + b\tilde{b} = q\tilde{q}. \tag{3.87.a}$$

$$a\tilde{c} + b\tilde{d} = 0. \tag{3.87.b}$$

$$c\tilde{a} + d\tilde{b} = 0. \tag{3.87.c}$$

$$c\tilde{c} + d\tilde{d} = q\tilde{q}. \tag{3.87.d}$$

If we merge these equations, we get $a\tilde{a} = d\tilde{d}$ and $b\tilde{b} = c\tilde{c}$. Let μ be a root of a. Write

$$a = a_1(z-\mu)^n(1-z\overline{\mu})^m,$$

$$b = b_1(z-\mu)^\alpha(1-z\overline{\mu})^\beta,$$

$$c = c_1(z-\mu)^\gamma(1-z\overline{\mu})^\delta,$$

$$d = d_1(z-\mu)^p(1-z\overline{\mu})^s,$$

where a_1, b_1, c_1 and d_1 are coprime to $(z-\mu)(1-z\overline{\mu})$. We get $a_1\tilde{a}_1 = d_1\tilde{d}_1$, and $b_1\tilde{b}_1 = c_1\tilde{c}_1$. Now, equation (3.87.$c$) says $c_1\tilde{a}_1 + d_1\tilde{b}_1 = 0$. We have also (comparing multiplicities of μ and $\overline{\mu}$)

$$n+m = p+s, \quad \alpha+\beta = \gamma+\delta, \quad m+\gamma = \beta+p, \quad n+\delta = \alpha+s. \tag{3.88}$$

Write $A = \min(n,s)$, $B = \min(n,p)$, $C = \min(\alpha,\delta)$, $D = \min(\beta,\gamma)$. Let $n = A + n_1$, $s = A + s_1$, and define n_1, p_1, α_1, β_1, γ_1, δ_1 accordingly. Now, equation (3.88) holds with indices 1 everywhere.

Now, $n_1 s_1 = 0$ and $m_1 p_1 = 0$. From $n_1 + m_1 = p_1 + s_1$, we get: $n_1 = p_1 = 0$ and $m_1 = s_1$, or $m_1 = s_1 = 0$, $p_1 = n_1$. The relation $\alpha_1 + \beta_1 = \gamma_1 + \delta_1$ gives a similar condition. If we add the other conditions of (3.88), we get

$$n_1 = p_1 = \alpha_1 = \gamma_1 \qquad m_1 = s_1 = \beta_1 = \delta_1.$$

Moreover, one of these quantities is zero. Thus, if

$$X = (z-\mu)^A(1-z\overline{\mu})^B, \quad Y = (z-\mu)^C(1-z\overline{\mu})^D, \quad Z = (z-\mu)^{n_1}(1-z\overline{\mu})^{m_1} \tag{3.89}$$

we have

$$D = \begin{pmatrix} a_1XZ & b_1YZ \\ c_1\tilde{Y}Z & d_1\tilde{X}Z \end{pmatrix}.$$

If we do this for all roots, we get a similar relation, where a_1, b_1, c_1 and d_1 are constant. We can put a_1 in X and b_1 in Y. Now $a_1\tilde{a}_1 = d_1\tilde{d}_1$ says $|d_1| = 1$, and (3.87.c) says $c_1 = -d_1$. Hence

$$Q = \frac{Z}{\tilde{q}} \begin{pmatrix} X & Y \\ -\lambda\tilde{Y} & \lambda\tilde{X} \end{pmatrix}.$$

Write $Z = AB$, where A is stable, and the roots of B are not in $\mathbb{U}$. Since we have $Z\tilde{Z}(X\tilde{X} + Y\tilde{Y}) = q\tilde{q}$, the roots of $\tilde{B}$ are in fact in $\mathbb{U}$ (i.e. Z has no roots on $\mathbb{T}$). Moreover A divides q and B divides $\tilde{q}$. Write $\tilde{q} = \tilde{q}_1 B$, $q_2 = q_1 A$. If $X_1 = XA\tilde{A}$ and $Y_1 = YA\tilde{A}$, we get

$$Q = \frac{1}{\tilde{q}_2}\begin{pmatrix} X_1 & Y_1 \\ -\lambda\tilde{Y}_1 & \lambda\tilde{X}_1 \end{pmatrix}. \tag{3.90}$$

Note: if Q has McMillan degree n, we can write (3.86), (3.87) with polynomials of degree n (without the implicit condition that $\tilde{q}$ is coprime to the elements in D). We know that $\det Q = \epsilon q/\tilde{q}$. This gives $ad - bc = \epsilon q\tilde{q}$, hence $a\tilde{a}d - \tilde{a}bc = \tilde{a}\epsilon q\tilde{q}$. From (3.87.$c$) we get $(a\tilde{a} + b\tilde{b})d = \tilde{a}\epsilon q\tilde{q}$. Now (3.87.$a$) gives $d = \epsilon\tilde{a}$.

Let's write (3.90) as

$$Q = \frac{1}{\tilde{q}}\begin{pmatrix} p_1 & p_2 \\ -\lambda\tilde{p}_2 & \lambda\tilde{p}_1 \end{pmatrix}. \tag{3.91.a}$$

We have

$$\tilde{p}_1 p_1 + \tilde{p}_2 p_2 = \tilde{q}q. \tag{3.91.b}$$

This gives another way of parameterising inner matrices of size 2. The first implementation of the algorithm was done by M. Cardelli in the scalar case and in the case $p = 2$ using this parameterisation (cf. [4]), but the the algorithm did not work well in the matrix case, because it is uneasy and numerically unstable to find q and its derivative as a function from p_1 and p_2 as defined by (3.91.b). The Schur algorithm was implemented using Ψlab by P. Fulcheri [8] in the real case. A complete study in the general case can be found in [9].

Let μ be such that $\mu\tilde{q}(1) = 1$. If $q' = \overline{\mu}q$, $p_1' = \mu p_1$, $p_2' = \mu p_2$ and $\lambda' = \lambda\mu/\overline{\mu}$, then equations (3.91) hold with primes everywhere. This means that we can assume $q(1) = \tilde{q}(1) = 1$. Assume now that $Q(1)$ is the identity matrix. Then $p_1(1) = 1$ and $p_2(1) = 0$, so that

$$p_1 = (1 - z)p_4 + 1, \qquad p_2 = (1 - z)p_3 \tag{3.91.c}$$

for some polynomials p_2 and p_4 of degree $< n$. Note that (3.91.b) implies, for $|z| = 1$,

$$|p_1(z)|^2 + |p_2(z)|^2 = |q(z)|^2.$$

Since q has no roots on $\mathbb{T}$, this means that p_1 and p_2 cannot have a common root on $\mathbb{T}$. If this condition is satisfied, then $X = p_1\tilde{p}_1 + p_2\tilde{p}_2$ can be factored as

$$X = \prod \frac{(z - \alpha_i)(1 - \overline{\alpha}_i z)}{(1 - \alpha_i)(1 - \overline{\alpha}_i)}, \qquad |\alpha_i| < 1.$$

If $q = \prod(z - \alpha_i)/(1 - \alpha_i)$, then q is stable and (3.91.b) holds.

Theorem 30
The set $\mathcal{I}_n^2(1)$ of inner matrices Q of size 2×2, of McMillan degree n with $Q(1) = I$ is topologically equivalent to the set of polynomials (p_3, p_4) of degree $< n$, such that p_1 and p_2 have no common root on $\mathbb{T}$.

Let's study in details the case $n = 1$. Define $\mathcal{I}_n'^p$ to be the set of all inner matrices of McMillan degree of size p, having degree at most n, and $\mathcal{I}_n'^p(1)$ be the quotient of this set by the equivalence relation: Q and Q' are equivalent if their quotient is a constant matrix. We have to minimise ψ over this set.

In the case $p = 2$, and $n = 1$, the polynomials p_3 and p_4 are constant. In the case where p_3 is not zero, then p_3 and p_4 are coprime. Otherwise, p_2 is zero, and the condition becomes $\Re p_4 \neq -1/2$. Thus, in the real case, $\mathcal{I}_1^2(1)$ is equivalent to the plane minus a point, i.e. $\mathbb{T} \times \mathbb{R}$. In the complex case, we have to remove a real line, so that the manifold is $\mathbb{T}_2 \times \mathbb{R}^2$ ($\mathbb{T}_2$ is the sphere in $\mathbb{R}^3$). Let's study the boundary of this manifold.

For simplicity, introduce $p_5 = p_4 + 1/2$. Then equation (3.91) becomes

$$D = I + (1-z)\begin{pmatrix} p_5 - 1/2 & p_3 \\ \overline{p}_3 & -\overline{p}_5 - 1/2 \end{pmatrix}. \tag{3.92}$$

Let $p_5 = u + iv$, $\alpha = 1/4 - u^2 - v^2 - iv - |p_3|^2$. Then (3.91.$b$) says

$$q\tilde{q} = \alpha z^2 + (1 - \alpha - \overline{\alpha})z + \overline{\alpha}. \tag{3.93}$$

The roots of $q\tilde{q}$ are

$$z = \frac{-1 + 2u' \pm \sqrt{1 - 4u' - 4v'^2}}{2\alpha} \tag{3.94}$$

if $\alpha = u' + iv'$. Note that if $1 - 4u' - 4v'^2 \le 0$, both roots are in $\mathbb{T}$. Otherwise, $2u' - 1 < 0$, and the root with a plus sign is in $\mathbb{U}$. Thus, the root of q is

$$\mu = \frac{-1/4 - |p_5|^2 - |p_3|^2 + \sqrt{(\Re p_5)^2 + |p_3|^2}}{1/4 - |p_5|^2 - |p_3|^2 - i\Im p_5}. \tag{3.95.a}$$

We can also write this as

$$\frac{1}{1-\mu} = \frac{1}{2} + \sqrt{(\Re p_5)^2 + |p_3|^2} - i\Im p_5. \tag{3.95.b}$$

If the square root is zero, then the real part of this expression is 1/2, so that $|\mu| = 1$; otherwise it is $> 1/2$, hence $|\mu| < 1$.

Let v be a real number. Define θ by $2v = \tan(\theta/2)$. Then, if p_5 is near iv, and p_3 is near zero, the root of (3.95) is near $-e^{i\theta}$. This gives $q = (z + e^{i\theta})/(1 + e^{i\theta})$. An easy computation says that Q is now near the identity matrix.

Note: Assume $p_3 = 0$. Equation (3.91) says that $q = p_1$ or $q = \tilde{p}_1$. Which one depends on the sign of the real part of p_5. We have

$$Q = \begin{pmatrix} \beta_\mu & 0 \\ 0 & 1 \end{pmatrix}, \Re p_5 > 0, \qquad Q = \begin{pmatrix} 1 & 0 \\ 0 & \beta_\mu \end{pmatrix}, \Re p_5 < 0. \tag{3.96}$$

Consider now what happens at infinity. We fix (p_3, p_5), and consider $p'_3 = \lambda p_3$, $p'_5 = \lambda p_5$, where $\lambda \to \infty$. Let μ' be the solution of (3.95.b) for these values, and define μ by

$$\frac{1}{1-\mu} = \sqrt{(\Re p_5)^2 + |p_3|^2} - i\Im p_5. \tag{3.97}$$

Then $1/(1-\mu') - 1/2 = \lambda/(1-\mu)$, so that the limit of μ' is 1. We have

$$Q = \frac{1 - \overline{\mu}'}{1 - z\overline{\mu}'} + \frac{1-z}{1 - z\overline{\mu}'}\begin{pmatrix} (1-\overline{\mu}')(p'_5 - 1/2) & (1-\overline{\mu}')p'_3 \\ (1-\overline{\mu}')\overline{p}'_3 & (1-\overline{\mu}')(-1/2 - \overline{p}'_5) \end{pmatrix}.$$

This has a limit

$$Q_0 = \begin{pmatrix} (1-\overline{\mu})p_5 & (1-\overline{\mu})p_3 \\ (1-\overline{\mu})\overline{p}_3 & -(1-\overline{\mu})\overline{p}_5 \end{pmatrix}. \tag{3.98}$$

This is an unitary matrix. In fact, we can always normalise p_3 and p_5 as

$$p_5 = \cos\theta e^{i\phi} \qquad p_3 = \sin\theta e^{i\psi}.$$

Now, μ is in $\mathbb{T}$, hence is e^{it}, and (3.96) says

$$\cos t = \sqrt{\cos^2\theta\cos^2\phi + \sin^2\theta}, \quad \sin t = \cos\theta\sin\phi, \tag{3.99.a}$$

and we get

$$Q_0 = \begin{pmatrix} \cos\theta e^{i(\psi-t)} & \sin\theta e^{i(\psi-t)} \\ \sin\theta e^{i(-\psi-t)} & -\cos\theta e^{i(-\phi-t)} \end{pmatrix}. \tag{3.99.b}$$

Note that Q_0 is not an arbitrary unitary matrix. In the real case, have $\sin\phi = 0$, hence $\sin t = 0$. Since $\cos t > 0$, we get $t = 0$. This implies that the determinant of Q_0 is -1. The converse is obvious: any orthogonal matrix of determinant -1, can be written in the form (3.99.b) with $\phi = \psi = t = 0$, and (3.99.a) holds.

In the complex case, the situation is a bit different: the set of unitary matrices is of dimension 4, and we have only three parameters. Note the special case where $\sin\theta = 0$. We can assume $\theta = 0$, so that we get $t = \phi$ if $\cos\phi > 0$, $t = \pi - \phi$ otherwise, hence one of

$$Q_0 = \begin{pmatrix} 1 & 0 \\ 0 & -e^{-2i\phi} \end{pmatrix}, \qquad Q_0 = \begin{pmatrix} -e^{2i\phi} & 0 \\ 0 & 1 \end{pmatrix}.$$

Note that the case $\cos\phi = 0$ is excluded. If we take the limit, we get the identity matrix in every case.

Now the structure of the manifold is a follows: in the real case, $\mathcal{I}_1^2(1)$ with its boundary is just a plane. However $\mathcal{I}_n'^p$ is this manifold where all unitary matrices are identified. Thus, ∞ and the point $p_3 = 0$, $p_5 = 0$, have to be identified. In the complex case, we have to identify ∞ and the real line $p_3 = 0$, $\Re p_5 = 0$ (by construction, $\mathcal{I}_1^2(1)$ is just a part of $\mathcal{I}_n'^p$).

One can wonder how this parameterisation is connected to the Schur algorithm. Equations (3.10) are

$$\tilde{q} = \tilde{b} - b\|y\|^2 - (\tilde{b} - b)u^*y. \tag{3.100.a}$$

$$D = \tilde{b} - b\|y\|^2 + (\tilde{b} - b)[yu^* - u^*y + uy^* - yy^* - uu^*]. \tag{3.100.b}$$

Divide everything by $(1-\omega)(1-\overline{\omega})(1-\|y\|^2)$ so as to obtain $q(1) = 1$. Introduce

$$\beta = \frac{1-|\omega|^2}{(1-\omega)(1-\overline{\omega})}, \quad \gamma = \frac{\overline{\omega}}{1-\overline{\omega}}, \quad T = \|y\|^2 + yu^* - u^*y + uy^* - yy^* - uu^*. \tag{3.101}$$

Then

$$D = I + (1-z)(\gamma + \beta\frac{T}{1-\|y\|^2}).$$

Write now $y = \begin{pmatrix} a \\ b \end{pmatrix}$, $u = \begin{pmatrix} c \\ d \end{pmatrix}$. Then

$$\frac{T}{1-\|y\|^2} = \begin{pmatrix} p_4 & p_3 \\ \overline{p_3} & 1-\overline{p_4} \end{pmatrix}, \quad p_3 = \frac{(a-c)(\overline{d}-\overline{b})}{1-|a|^2-|b|^2} \qquad p_4 = \frac{c(\overline{a}-\overline{c}) + b(\overline{b}-\overline{d})}{1-|a|^2-|b|^2}. \tag{3.102}$$

Define $p_3' = \beta p_3$, $p_4' = \beta p_4 + \gamma$. The relation $\gamma + \overline{\gamma} = \beta - 1$ says

$$D = I + (1-z)\begin{pmatrix} p_4' & p_3' \\ \overline{p_3'} & 1-\overline{p_4'} \end{pmatrix}. \tag{3.103}$$

We have shown on figure 3.1 this mapping. We have chosen $\omega = 0$, $c = 1$, and $d = 0$, so that

$$p_3 = \frac{(1-a)\overline{b}}{1-|a|^2-|b|^2} \qquad p_4 = \frac{\overline{a} - 1 + |b|^2}{1-|a|^2-|b|^2}. \tag{3.104}$$

Getting the inverse mapping is not trivial. Consider first

$$a' = \overline{c}a + \overline{d}b \quad b' = -da + bc$$

$$p_3'' = -\overline{c}\overline{d}(p_4 + \overline{p}_4 + 1) - \overline{d}^2\overline{p}_3 + \overline{c}^2\overline{p}_3$$

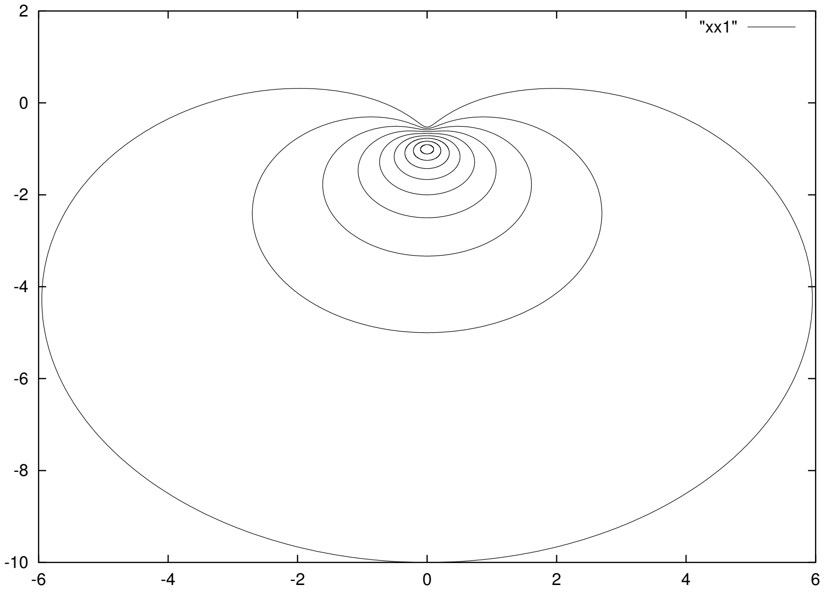

Figure 3.1: Expression of Q as a function of the Schur parameters. If Q is defined by equation (3.100), (3.101) and (3.102), we plotted the values of p_3 and p_4, for $\omega = 0$, $u = (1,0)^t$; on each curve $\|y\|$ is constant, and is $i/10$ for $1 \le i \le 9$. The point $(0,0)$ is unreachable; the line $(0,y)$ for $y > 0$ is unreachable in this chart.

$$p_4'' = c\overline{c}p_4 + c\overline{d}\overline{p}_3 + \overline{c}p_3 - d\overline{d} - d\overline{d}\overline{p}_4.$$

In fact, since $\|u\| = 1$, there exists a unitary matrix A such that $Au = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. What we do is multiply y by A, and replace Q by AQA^*. This gives (compare with (3.104)):

$$p_3'' = \frac{(1-a')\overline{b}'}{1-|a'|^2-|b'|^2} \qquad p_4'' = \frac{\overline{a}'-1+|b'|^2}{1-|a'|^2-|b'|^2}.$$

Now the inverse mapping is defined as follows: we have p_2' and p_4'. We deduce p_3 and p_4 from $p_3' = \beta p_3$, $p_4' = \beta p_4 + \gamma$. We compute p_3'' and p_4'', thus a' and b', and then get a and b.

All we have to do now is express (a,b) as a function of (p_3,p_4) in case (3.104) is true. Let $x = p_3$ and $y = p_4 + 1/2$, $a' = a/(1-2\overline{y})$, $b' = b/(2\overline{x})$. The magic is that $a' = b'$. If $T = 1/a'$, we have a second relation

$$|T|^2 = 2T - 1 + 2y + 2\overline{y} + 4|x|^2 + 4|y|^2. \tag{3.105}$$

The solutions are

$$T = 1 - 2iv \pm \sqrt{|x|^2+u^2}, \quad y = u + iv.$$

We deduce

$$\frac{1}{|a|^2+|b|^2} = 1 + 4\frac{\pm\sqrt{|x|^2+u^2}-u}{(1+2u)^2+4v^2+4|x|^2}. \tag{3.106}$$

In order for $|a|^2+|b|^2 < 1$, we have to chose the plus sign.

$$a = \frac{2(\overline{p}_4+1)}{T}, \quad b = \frac{2\overline{p}_3}{T}, \quad T = 1 - 2i\Im p_4 + \sqrt{|p_3|^2 + \Re(p_4+1/2)^2}.$$

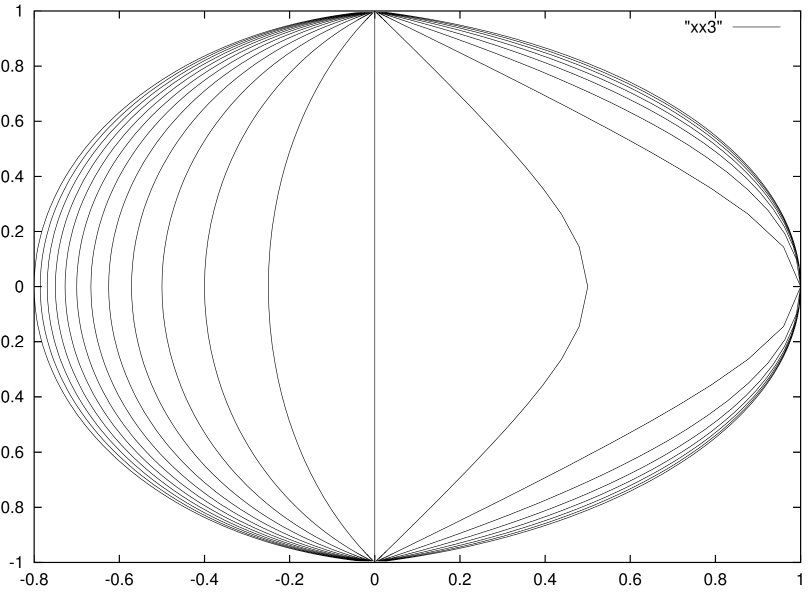

Figure 3.2: Inverse mapping of the previous figure. We use the same chart and show the value of y, which is in the unit circle. On each curve, p_4 is constant, and p_3 varies.

We shown the result on figure 3.2. Note the dissymmetry between the curves on the left and the curves on the right. On each curve, p_4 is fixed. For (3.106), if y is large, we have

$$\frac{1}{|a|^2+|b|^2}-1=\frac{2}{|u|}, u<0, \qquad \frac{1}{|a|^2+|b|^2}-1=\frac{x^2}{u^3}, u>0.$$

Note also that if $x=0$, $u>0$, we have $|a|^2+|b|^2=1$. This means that our application is not a bijection: With a and b we obtain every (p_3,p_5), except those for which $p_3=0$, and $\Re p_5>0$. The matrix Q we cannot obtain are those defined by (3.96). In fact, since $Q(\omega)^*u=y$, we cannot obtain the matrices Q such that $Q(\omega)^*u$ have unit norm.

3.6.3 General case

Let's introduce some notations. The set $\mathcal{B}_p$ denotes the set of vectors y of size p with $\|y\|<1$. Given Q, u_i and ω_i, we can define $Q^{(n)}=Q$, and $Q^{(k)}=T_{\Theta_k}(Q^{(k-1)})$, provided that $y_k=Q^{(k)}(\omega_k)^*u_k$, $\|y_k\|<1$, and Θ_k is defined by ω_k, u_k and y_k.

Let $\mathcal{V}_{(\mathbf{w},\mathbf{u})}$ be the set of all inner matrices Q for which this is possible ($\mathbf{u}=(u_1,u_2,\dots,u_p)$, and $\mathbf{w}=(\omega_1,\dots,\omega_n)$), and $\varphi_{(\mathbf{w},\mathbf{u})}$ be the list $(y_1,\dots,y_n)$. This is an element of $\mathcal{B}_p^n$. If Θ_i is defined by (ω_i,u_i,y_i) and $\Theta=\Theta_1\Theta_2\dots\Theta_n$, then

$$y=\varphi_{(\mathbf{w},\mathbf{u})}(Q)\iff T_\Theta(I)=Q.$$

Lemma 53
The family $(\mathcal{V},\varphi)$ defines a C^∞ atlas on $\mathcal{I}_n^p(1)$ which is compatible with its natural structure of embedded submanifold of $H_2^{p\times p}$.

For the proof of this lemma, see [9].

Theorem 31
Assume $|\omega_i| < 1$, $\|u_i\| = 1$ and $\|y_i\| \leq 1$. Define $q^{(i)}$ and $D^{(i)}$ via formulas (3.10). This is possible under the assumption that no $\tilde{q}^{(i)}$ is identically zero. This condition is the same as $u_k^* Q^{(k-1)} y_k - 1$ not identically zero.

Then $q^{(i)}$ and $D^{(i)}$ are polynomials of degree $\leq i$, $q^{(i)}$ has degree i. The matrix $Q = D^{(n)}/\tilde{q}^{(n)}$ is inner, its McMillan degree is n, minus the number of k such that $\|y_k\| = 1$, minus the number of points $\xi \in \mathbb{T}$ such that $y_k = Q^{(k-1)}(\xi)^* u_k$. There exists a neighbourhood $\mathcal{W}$ of $\mathbf{y}$ such that $\varphi^{-1}_{(\mathbf{w},\mathbf{u})}$ extents smoothly to $\mathcal{W}$.

Proof. The important point here is that the coefficients of $D^{(k)}$ and $q^{(k)}$ are polynomial functions of y_i. This explains that φ^{-1} can be computed for every y. Moreover, $D^{(k)}\tilde{D}^{(k)} = q^{(k)}\tilde{q}^{(k)}$, and the roots of $q^{(k)}$ are in $\overline{\mathbb{U}}$. Thus $Q^{(k)}$ is inner, provided that it is defined, i.e. that $\tilde{q}^{(k)}$ is not identically zero.

The proof is by induction on k. Assume first that $\|y_k\| < 1$. Then we are in the case studied above, and $Q^{(k)}$ is inner, its McMillan degree is one more than the McMillan degree of $Q^{(k-1)}$. Note that the equation $y_k = Q^{(k-1)}(\xi)^* u_k$ has no solution $\xi \in \mathbb{T}$, since $Q^{(k-1)}(\xi)^*$ is unitary, $\|u_k\| = 1$, $\|y_k\| < 1$.

Assume now $\|y_k\| = 1$. Then

$$\tilde{q}_B = (1-z)(1-|\omega|^2)(\tilde{q}_A - u^* D_A y).$$

$$\tilde{q}_B = (1-z)(1-|\omega|^2)\tilde{q}_A(1 - u^* A y). \tag{3.107.a}$$

Let's now compute the roots of $\tilde{q}_B/(z-1)$ in $\overline{\mathbb{U}}$. If ξ is such a root, then $u^* A(\xi) y = 1$. Write $u^* A(\xi) = \lambda y^* + w^*$, where w is orthogonal to y. Then $\lambda = 1$, because $\|y\| = 1$. But $\|u^* A(\xi)\| \leq 1$, so that $w = 0$, and $y^* = u^* A(\xi)$. This is $A(\xi)^* u = y$, and implies $u^* A(\xi) y = 1$. Since $|u^* A(\xi) y| \leq 1$ on $\overline{\mathbb{U}}$, if equality holds on a point of $\mathbb{U}$, then equality holds everywhere (maximum modulus principle).

This means that either $\tilde{q}_B$ is identically zero, or does not vanish on $\mathbb{U}$. We exclude the first case. Thus, $\tilde{q}_B(0) \neq 0$, and this says that q_B has degree n.

Now,

$$D_B = (1-z)(1-|\omega|^2)[D_A + \frac{D_A y u^* D_A - u^* D_A y D_A}{\tilde{q}_A} + \tilde{q}_A u y^* - D_A y y^* - u u^* D_A].$$

$$D_B = (1-z)(1-|\omega|^2) q_A [A + A y u^* A - u^* A y A + u y^* - A y y^* - u u^* A]. \tag{3.107.b}$$

This is zero if $z = 1$, or if $y^* = u^* A(\xi)$. Thus in the quotient $D_B/\tilde{q}_B$, we can simplify by a factor T, which is $(z-1)\prod(z-\xi_i)$, where ξ_i is defined by $u^* A(\xi_i) y = 1$. Then we obtain a stable polynomial q. □

Lemma 54
Assume that Q is inner, of McMillan degree $d < n$. There exists a unitary matrix $\mathcal{U}$, and quantities $(\mathbf{w}, \mathbf{u}, \mathbf{y})$, such that $\mathcal{U}Q$ has the form of the previous theorem.

Proof. Consider $n-d$ elements u_i, y_i and ω_i. Let's assume that $\|y_i\| = 1$, and $y_i \neq u_i$. Let's apply the Schur algorithm, at order n. Equations (3.107) say that the first matrix we obtain is constant, and is

$$\mathcal{U}_1 = I - \frac{(y_1 - u_1)(y_1^* - u_1^*)}{1 - u_1^* y_1}. \tag{3.108}$$

The d-th matrix is also constant and unitary. Let $\mathcal{U}$ be this matrix. Assume that Q is obtained via the Schur algorithm with parameters ω_{i+n-d}, u'_{i+n-d} and y_{i+n-d}. Define $u_{i+n-d} = \mathcal{U}u'_{i+n-d}$. Since Q is of degree d, we have $\|y_i\| < 1$ for $i > n-d$. The key relation is that the Schur algorithm satisfies

$$S_{(\omega,\mathcal{U}u)}(\mathcal{U}A, y) = \mathcal{U}S_{(\omega,u)}(A, y), \tag{3.109}$$

provided that $\mathcal{U}$ is unitary and constant. This relation shows that the n-matrix we obtain is $\mathcal{U}Q$. □

Lemma 55
Assume that $y_i(t)$ are C^∞ functions of t, $0 \le t \le 1$. Assume that for $t > 0$, $\|y_i\| < 1$, and for $t = 1$, $\|y_i\| \le 1$. The Schur algorithm applied to $(\omega_i, u_i, y_i(t))$ gives an inner matrix $Q(t)$. This matrix is of degree n in case $\|y_i(0)\| < 1$, of degree $< n$, if $\tilde{q}^{(n)}(0)$ is not identically zero, and anything can happen otherwise.

The only thing we have to show is the behaviour when $\tilde{q}$ vanishes identically. Consider the scalar case first. Here we have

$$\tilde{q}_B = (\tilde{b} - b\|y\|^2)\tilde{q}_A - (\tilde{b} - b)u^* D_A y$$
$$D_B = (b - \tilde{b}\|y\|^2)D_A + (\tilde{b} - b)\tilde{q}_A u y^*.$$

A little computation shows that

$$\frac{\beta}{Bu^* - y^*} = \frac{1}{Au^* - y^*} - \frac{(1-\beta)y}{1 - \|y\|^2}. \tag{3.110}$$

Now if $\lim \|y\| = 1$, the limit of the last term is ∞, so that the limit of $B^*u - y^*$ is zero. Note that the condition $\tilde{q}_B$ is identically zero at $t = 0$ says that $uy^* - A = 0$, so that A must be constant. In this case the limit of $1/(Au^* - y*)$ is also infinity. Since $1 - \beta$ takes an infinite number of values, the quantity $\beta/(Bu^* - y^*)$ is infinite for almost every value of z, so that B is still constant.

Now, in the case $p \ge 2$, anything can happen: Q has no limit, Q has a limit of degree n, Q has a limit of degree $< n$. It suffices in fact to consider the case $p = 2$. We can assume $n = 1$, and everything real. Consider the case $u = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ and $\omega = 0$. Then

$$\tilde{q}_B = 1 - z(y_1^2 + y_2^2) - (1 - z)y_1. \tag{3.111}$$

Let z_0 be the root of $\tilde{q}$. Let

$$\mathcal{U} = \begin{pmatrix} y_1 & y_2 \\ y_2 & 1 - y_1 - 1/z_0 \end{pmatrix}.$$

Then

$$B = \frac{(1 - y_1)(1 - z)\mathcal{U} + z(1 - y_1^2 - y_2^2)I}{(1 - y_1)(1 - z) + z(1 - y_1^2 - y_2^2)}. \tag{3.112.a}$$

This can be written as

$$B = \frac{z_0(1 - z)\mathcal{U} + z(z_0 - 1)}{z_0 - z}. \tag{3.112.b}$$

Assume now that the limit of y_1 is not one. Then (3.112.a) says that the limit of B is the identity matrix. Assume now that the limit of y_1 is one. We have shown on figure 3.3 the sets (y_1, y_2) that give z_0 as a root of q_B. On can see that, if y has u as limit, but if this limit is not tangential, then the limit of the root is $z_0 = 1$. Otherwise, any value value outside $[-1, 1]$ can be the limit.

In fact, if z_0 has a limit, then $\mathcal{U}$ has a limit, and B has a limit. If $z_0 \ne 1$, this limit has McMillan degree n, otherwise, it is constant (note that $z_0 = \infty$ is valid, this gives $q = 0$). We have

$$1 - 1/z_0 = \frac{1 - y_1^2 - y_2^2}{1 - y_1}.$$

Assume that $y_2(t) \sim \alpha t^n$, and $y_1(t) \sim 1 - \beta t^m$. The previous quantity is approximatively

$$\frac{2\beta t^m - \beta^2 t^{2m} - \alpha^2 t^{2n}}{\beta t^m}$$

We must have $2n \ge m$, so that we get the asymptotic result:

$$2 - \beta t^m - \alpha^2 t^{2n-m}/\beta.$$

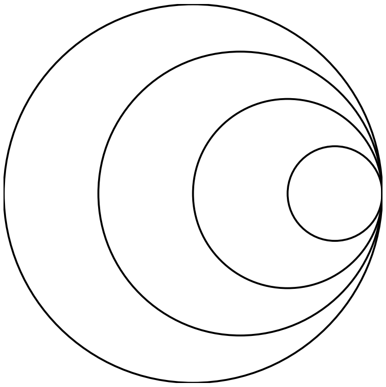

Figure 3.3: Roots of (3.111). To each (y_1, y_2) we can associate a root z. If z is fixed, we get this root if (y_1, y_2) is on a circle, symmetric w.r.t. the x-axis, and that passes through $(1, 0)$. We have shown these circle for $z = -2$, $z = \infty$ and $z = -2$. The outer circle is the unit circle ($z = 1$).

Now the case that causes trouble is when all derivatives of y_1 and y_2 are zero. An example is given by

$$y_1(t) = 1 - \frac{\sin(1/t)^2 + 1}{2} e^{-1/t^2}. \qquad y_2(t) = e^{-1/t^2}.$$

Here

$$\frac{1 - y_1^2 - y_2^2}{1 - y_1} = \frac{2\sin(1/t)^2}{1 + 2\sin(1/t)^2}$$

and this has no limit.

3.6.4 Minimisation of ψ

Let $\psi_F^n(Q)$ be the function defined in the previous chapter. It is the minimum of $\|F - Q^{-1}C\|^2$, where $Q^{-1}C$ is rational, strictly, proper, stable, of McMillan degree n. For simplicity, we shall omit the subscript F. Define

$$\psi^n_{(\mathbf{w},\mathbf{u})}(\mathbf{y}) = \psi^n(\varphi^{-1}_{(\mathbf{w},\mathbf{u})}(\mathbf{y})). \tag{3.113}$$

In fact, if $\mathbf{y}$ is as above, we let y be the vector with same components as $\mathbf{y}$. This is an element of $\mathbb{C}^{np}$. What φ^{-1} computes is not Q, but the numerator and denominator. Let x be the vector whose entries are the coefficients of $\tilde{D}$ and q. Then $x \in \mathbb{C}^k$, with $k = (n + 1)(p^2 + 1)$. Write $x = X(y)$, and $\psi^n(Q) = \Psi(x)$. Hence

$$\psi^n_{(\mathbf{w},\mathbf{u})}(y) = \psi(X(y)). \tag{3.114}$$

What we do is integrate the differential equation

$$\frac{dy}{dt} = -\nabla\psi^n_{(\mathbf{w},\mathbf{u})}(y). \tag{3.115}$$

In fact, if A is a function of $(a_1, \dots, a_m)$ with components $(A_1, \dots, A_p)$, then ∇A is the matrix whose (i, j) component is $\partial A_i/\partial a_j$. The chain rule says $\nabla(A \circ B) = \nabla A \nabla B$. Note that $\nabla\Psi$ is a $1 \times np$ vector while dy/dt in (3.115) is a $np \times 1$ vector. It is of course possible to identify these vectors. The important point is that $\nabla\psi$ is not a tangent vector on the manifold: it is just a linear form. If we use a chart, then we can define $\nabla\Psi$, and identify this to a tangent vector. In any case, we get

$$\frac{dx}{dt} = -\nabla X \nabla X^t \nabla\Psi, \tag{3.116.a}$$

$$\frac{d\psi}{dt} = -(\nabla\Psi\nabla X)(\nabla\Psi\nabla X)^t. \tag{3.116.b}$$

Lemma 56
The function $\psi(t)$ is either constant or decreasing. If it is constant, then Q is a critical point of ψ^n, and y is a critical point of $\psi^n_{(\mathbf{w},\mathbf{u})}$. Otherwise, the solution of the differential equation (3.116.a) depends not only on the initial condition Q, but also on u and ω.

Note that the equation $\nabla\Psi\nabla X = 0$ does not imply $\nabla\Psi = 0$. This is because the mapping X is not surjective (there are much more parameters in x, i.e. in D and q then in y). What we say is just that if x' is near to x, and if x' defines an inner matrix, then $\Psi(x') - \Psi(x) = o(x' - x)$. This is $\nabla\psi^n = 0$.

Assume now that one component of y satisfies $\|y_i\| = 1$. Assume moreover that we are in a generic case, i.e. that the matrix Q_0 we obtain is of McMillan degree $n-1$. Assume that this is a critical point of ψ^{n-1}. Let h be a vector, and consider what happens if we replace y by $y + th$. If $Q_0 = Q_1 \dots Q_{n-1}A$ is a Potapov factorisation of Q_0, and if Q_0 has only simple roots, then any matrix Q'_0 near Q_0 can be factored as as $Q'_0 = Q'_1 \dots Q'_{n-1}A'$, where Q'_i is near Q_i. This is because the interpolation condition $u_1^* Q_0(\omega_1) = 0$, gives $u_1'^* Q'_0(\omega'_1) = 0$. There exists, for small t, a unique pole ω'_1 of Q'_0 near ω_1, and since this pole is simple, the set of vectors u with $u_1'^* Q'_0(\omega'_1) = 0$ has dimension one. If we chose in this space a vector with unit norm, such that the phase of the k-th component is the same as the phase of the k-component of u_1 (for some k), then u'_1 is near u_1.

Now, if Q_t is the result of the Schur algorithm for $y + th$, what we get is a matrix A' near A, and if we chose h such that $\|y + th\| < 1$, the matrix A' will be of McMillan degree one.

Consider now equations (2.23) and (2.24). If $Q(t) = Q_0(t)Q_1(t)$, where $Q_0(0) = 0$, and $Q_1(0) = A$ is constant, these equations say $\psi(t) = \psi(Q_0) - \|R_2/q_2\|^2 + o(t)$. Recall that Q_0 is a critical point of ψ. Note that $R_2(0) = 0$. Assume that y_k is of norm one. In the case where $h_k = 0$, then $\psi(Q(t)) - \psi(Q(0)) = o(t)$, because $Q_1(t)$ is very near to a constant. The same is true if we chose h such that $h_i = 0$ for $i \neq k$, and h_k orthogonal to y_k. If however h_k is proportional to y_k, say $h_k = -y_k$, then generically, the derivative of $\|R_2/q_2\|$ is not zero (compare with (2.40): the condition is $V_1(\alpha) \neq 0$, here it is a bit more complicated). As a conclusion, $\nabla\psi$ is parallel to y_k. We can compare this with the scalar case.

Theorem 32
Assume that the solution of (3.115) *is such that $r(t) = \max \|y_i(t)\|$ satisfies $r(t) < 1$ for $t < t_0$ and $r(t_0) = 1$. Then $Q = \varphi^{-1}(y)$ is not a local minimum of ψ.*

Assume that, for $t = 0$, $r(0) = 1$, but only one vector y_i is of norm one. Then generically, $r(t) < 1$ for t positive and small.

Of course, all problems that happen in the scalar case can happen in the matrix case. We are interested in the second claim of the theorem. We assume that Q is inner, of McMillan degree $n-1$ with Schur parameters (ω_i, u_i, y_i), $i = 2, \dots, n$. We consider y_1 and u_1 be two vectors of norm one, $y_1 \neq u_1$, $\omega_1 \in \mathbb{U}$. Let $\mathcal{U}$ be the matrix defined by (3.108). Let $\mathbf{y}' = (y_1, \dots, y_n)$, $\mathbf{u} = (u_1, \mathcal{U}u_2, \dots, \mathcal{U}u_n)$ and $\mathbf{w}' = (\omega_1, \omega_2, \dots, \omega_n)$. This gives us a starting condition. What we need is $u_1^* y_1 \neq 1$. In the complex case, we can chose u_1 such that all components but the first one is zero, the first one being 1, and y_1 such that all components are zero, the first one being $e^{2im\pi/k}$, for $0 < k < m$. In the real case, the same can be done if $p > 1$: we put the sine and cosine of $2m\pi/k$ into the first two components of y. But if $p = 1$, we have no choice: we must have $u = \pm 1$ and $y = -u$. According to (3.5), multiplying u and y by an element of $\mathbb{T}$ does not change anything: in other words, there is only one possible initial condition.

We do not know the exact conditions on which $r'(0) < 0$. Because the equations are badly conditioned if $\|y\| = 1$, we do not use the quantities defined above, but change them a little bit. In some cases, we have $r'(0) > 0$. This worries us a little bit, but in all our test cases r remains less than one.

We explained above that $\|y\|$ should be small. This means that, from time to time we chose a better chart. This is impossible for $t = 0$ (of course, we assume that we have a good chart for the matrix that is of degree $n-1$, but the chart for the degree n is never good in the sense given above). The reason why we want y to be small is the following: assume that $n = 1$. Then on the boundary Q is constant and $\psi = 1$. If we are near a minimum, then ψ' is small. If we are near a minimum and near the boundary, then ψ'' has to be huge somewhere.

Assume that ω is fixed. We are interested in finding y with smallest norm, with $y = Q(\omega)^* u$.

Consider first the case $p = 2$. Assume that $Q(\omega)^* = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. We may consider $u = \begin{pmatrix} \cos\theta \\ \sin\theta e^{i\phi} \end{pmatrix}$, since multiplying u by a constant number of modulus one does not change the norm of y. The square of the norm of y is now $A\sin^2\theta + B\sin\theta\cos\theta + C\cos^2\theta$ where

$$A = |a|^2 + |c|^2, \qquad C = |b|^2 + |d|^2, \qquad B = 2\Re(\overline{a}be^{i\phi} + \overline{c}de^{i\phi}).$$

We have

$$\|y\|^2 = \frac{A + C + (C - A)\cos 2\theta + B\sin 2\theta}{2}.$$

This expression is minimal or maximal if $\tan 2\theta = B/(C - A)$. Define $\Delta = \sqrt{(A-C)^2 + B^2}$. Then $\cos 2\theta = (C - A)/\Delta$. Then $\|y\|^2 = (A + C)/2 \pm \Delta/2$. The minimum is found if we take a minus sign, and if we maximise B. In fact, we chose $e^{i\phi}$ in such a way that $\overline{a}be^{i\phi} + \overline{c}de^{i\phi}$ is real: if $w = \overline{a}b + \overline{c}d$, we take $e^{i\phi} = -\overline{w}/|w|$. This gives $B < 0$, so that we have $\sin\theta = \sqrt{(1 - (C - A)/\Delta)/2}$ and $\cos\theta = \sqrt{(1 + (C - A)/\Delta)/2}$, where both square roots are positive.

In the general case, we are looking for u such that $Q(\omega)^*u$ is small. Let $B = Q(\omega)^*$, and $C = B^*B$. If $y = Q(\omega)^*u$, then $u^*Cu = \|y\|^2$. There exists an orthonormal basis e_i such that $Ce_i = \lambda_i e_i$. If $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, then $\|y\|^2 \geq \lambda_1$, and equality holds if $u = e_1$. Thus, we have to find the eigenvector of C with smallest eigenvalue. Since $\lambda_n \leq 1$, if $D = I - B^*B$, we have to find the eigenvector of D with largest eigenvalue. This can be found via following iteration $u_{k+1} = Du_k/\|Du_k\|$. Unless we start with an eigenvector, this will converge to the desired result. If ϵ_i is the i-th base vector, we start with ϵ_i, unless this is an eigenvector of D. In case every base vector is an eigenvector of D, then D is diagonal, and it is obvious to find u.

Note that if ω is not well chosen, this may produce bad values for y. In the complex case, we can chose for ω a pole of Q, i.e. a zero of q. The same can be done in the real case, provided that q has a real zero. If this is not the case, we chose a lot of values of ω (in fact, we chose $\omega = i/8$, where $-8 < i < 8$).

Chapter 4

Automatic differentiation

In this chapter we give the C++ code that is used to implement the formulas defined in the previous chapter. In the scalar case the code is rather easy, but we have different equivalent formulas. Only one implementation will be given. Others are studied in the next chapter, where we give the complexity of different variants. In the matrix case, the situation is quite more complicated. We shall discuss here what is automatic differentiation, why we use it, how we use it, and why we do not use generic software.

4.1 Introduction

In the multivariate case, the big problem is to implement the Schur formulas, and the derivatives of them. A general problem, when optimising a function f, is to compute the derivatives of f, and, maybe, the second derivatives. A good approximation of the derivative of f, in the direction d, is obtained by *finite differences*, namely $[f(x+hd)-f(x)]/h$. We have to chose h small, so that this is a good approximation of f', but not too small, otherwise we have too many rounding errors. In general, using divided differences again for the Hessian gives very bad results.

In the case where f is defined by simple formulas, the best thing to do is implement the equations that give f'. In general, this is slower than finite differences, but gives f' with a good precision. Obtaining the exact code of f' can be done by hand, using a computer algebra system like Maple, or using an automatic differentiator. In general, the hand-written code is more efficient, but could be wrong (there could be an error in the formula that gives f', or an error in the translation of the formula into a C program). Since a good optimiser manages to minimise f, even if f' is not precise, such errors may remain undetected for a long time (we once found that the sign of the Hessian was wrong; after correction, the program run much faster). Both computer algebra systems and automatic differentiators can compute the derivative of simple expressions, they have both their advantages and their limits.

In our case, we have to implement the Schur formulas. A first attempt was done by P. Fulcheri, using the Scilab program (see [8]). What we have to compute is a function that depends on (u_i, ω_i, y_i), and find the derivatives with respect to y_i. In the case $n = 3$, the code has the following structure

$$Q_1 = f(y_1, I, p_1), \tag{4.1.1}$$

$$Q_2 = f(y_2, Q_1, p_2), \tag{4.1.2}$$

$$Q_3 = f(y_3, Q_2, p_3), \tag{4.1.3}$$

where $p_i = (u_i, \omega_i)$, and I is the identity matrix. If we differentiate these functions, and denote by f_i the derivative of f with respect to the i-th argument, we get

$$\frac{\partial Q_i}{\partial y_j} = 0 \qquad (j > i) \tag{4.2.1}$$

$$\frac{\partial Q_i}{\partial y_j} = f_1(y_i, Q_{i-1}, p_i) \qquad (j = i) \tag{4.2.2}$$

$$\frac{\partial Q_i}{\partial y_j} = f_2(y_i, Q_{i-1}, p_i)\frac{\partial Q_{i-1}}{\partial y_j} \qquad (j < i). \tag{4.2.3}$$

If we differentiate again, we get

$$\frac{\partial^2 Q_i}{\partial y_j \partial y_k} = f_{22}(y_i, Q_{i-1}, p_i)\frac{\partial Q_{i-1}}{\partial y_j}\frac{\partial Q_{i-1}}{\partial y_j} + f_2(y_i, Q_{i-1}, p_i)\frac{\partial^2 Q_{i-1}}{\partial y_j \partial y_k} \tag{4.3}$$

in the case $k < i$ and $j < i$, zero in the case $j > i$ or $k > i$, and a different formula otherwise. Note that y_i is a vector, so that the formula for $j = i$ is a bit more complicated than given above. This is in particular the case for the second derivative (this explains why we did not give here the formula).

In the first implementation, the objective was to show that the Schur algorithm can be applied to real cases. No attempt was made to get an efficient code. In particular, some subexpressions where computed more than once. In the case of the Hessian, all first derivatives were recomputed wherever needed. If we look closely at the Schur formulas,

$$q_B = (b - \tilde{b}\|y\|^2)q_A + (\tilde{b} - b)y^*\tilde{D}_A u, \tag{4.4.1}$$

$$\tilde{D}_B = (b - \tilde{b}\|y\|^2)\tilde{D}_A + (\tilde{b} - b)[\tilde{D}_A uu^* + yy^*\tilde{D}_A - y^*uq_A] + \frac{b - \tilde{b}}{q_A}[\tilde{D}_A uy^*\tilde{D}_A - y^*\tilde{D}_A u\tilde{D}_A], \tag{4.4.2}$$

we see that $y^*\tilde{D}_A u$ is shared. Now, $x_1 = y^*\tilde{D}_A$ and $x_2 = \tilde{D}_A u$ appear also in the formulas, as subexpressions. We can compute x_1 and x_2, and deduce $y^*\tilde{D}_A u$ from one of them. Using one or the other is irrelevant. What we do is use x_1, and compute $\tilde{D}_A uu^*$ by multiplying $\tilde{D}_A$ by uu^*; we do this because the second factor is constant. Note that constant means that it has zero as derivative; but, since it is independent of y, it can be precomputed (note however that the cost of uu^* is small compared to the other formulas, so that this is a small optimisation).

As a conclusion, it appears that we have to split our big formulas into a set of smaller ones, and store in memory every result, and never compute twice the same quantity. The main objection against differentiation in reverse mode is memory usage. It happens that, in this case, even in direct mode, we have to store a lot of things: memory usage is the same in reverse mode, and in optimised direct mode. The main advantage of reverse mode is its speed: this is why we adopted it. In the next chapter, we give the complexity in both direct and reverse mode.

There are systems like Odyssée (see [16, 6]) that take as input a Fortran program, and return another Fortran program that computes the derivative of the code. The list of all these systems can be found on Internet[1]. Other systems, like Adolc, do the same with C programs. There are also some computer algebra systems that offer automatic differentiation modules, but none of these systems is really adapted to our needs. This is the main reason why we wrote a small automatic differentiator.

4.2 Straight line programs

A straight line program is a basic and simplified model of a computer program. What we exclude explicitly is input/output routines, transfer of control, and calls to other programs. Given these constraints, one gets a nice theory, that can be applied to no real program. It is always possible to relax these constraints. The main difficulty however is that, if the code is defined by: replace x by $f_p(x)$ until x converges with precision ϵ, the differentiated code computes x' which is, in general, a very bad approximation to the derivative of the fixed point of f_p (if x_p is the fixed point of f_p, there are conditions under which x_p is unique and is differentiable with respect to p). In our case, we have only explicitly loops (associated to matrix or polynomial products). For this reason, our differentiator is very small.

---

[1] `http://www.mcs.anl.gov/Projects/autodiff/AD_Tools/index.html`

4.2.1 Definition

A straight line program is the basic model for computer programs. It is given by

- n input variables $x_1, \dots, x_n$,
- m output variables $y_1, \dots, y_m$,
- p local variables $z_1, \dots, z_p$,
- q function calls $F_1, \dots, F_q$.

Let $N = n+m+p$, $w = (x_1, \dots, x_n, y_1, \dots, y_m, z_1, \dots, z_p)$. This is called the state vector. Each function call has the form $(f, t_0, t_1, \dots, t_k)$ where f is a function from $\mathbb{R}^k$ to $\mathbb{R}$, and each t_i is the index of a variable, an integer between 1 and N. To the function call F we associate a function

$$\begin{array}{rcl} \check{f} : \mathbb{R}^N & \rightarrow & \mathbb{R}^N \\ w & \rightarrow & w' \end{array}$$

defined by

$$\begin{array}{rcll} w'_i & = & w_i & (i \neq t_0) \\ w'_{t_0} & = & f(w_{t_1}, \dots, w_{t_k}). & \end{array} \tag{4.5.a}$$

If there is no confusion, we just write f instead of $\check{f}$.

We consider also

$$\begin{array}{rcl} f_0 : \mathbb{R}^n & \rightarrow & \mathbb{R}^N \\ (x_1, \dots, x_n) & \rightarrow & (x_1, \dots, x_n, 0, \dots, 0) \end{array} \tag{4.5.b}$$

and

$$\begin{array}{rcl} f_{q+1} : \mathbb{R}^N & \rightarrow & \mathbb{R}^m \\ (x_1, \dots, x_n, y_1, \dots, y_m, z_1, \dots, z_p) & \rightarrow & (y_1, \dots, y_m). \end{array} \tag{4.5.c}$$

The function computed by the SLP is

$$\begin{array}{rcl} \mathbb{R}^n & \rightarrow & \mathbb{R}^m \\ f & = & f_{q+1} \circ f_q \circ \cdots \circ f_1 \circ f_0. \end{array} \tag{4.6}$$

4.2.2 Rational SLP

An SLP is called rational if it computes a rational function. In fact, we may assume that each f_i is a rational function. We can decompose f_i into elementary (binary) operations, hence get one of the following forms: $y = a$, $y = -a$, $y = 1/a$, $y = -1/a$, $y = a + b$, $y = a - b$, $y = ab$, $y = -ab$, $y = a/b$, $y = -a/b$. We allow also ternary operations of the form $y = a + bc$. Hence the general form is

$$y = a \pm bc, \qquad y = \pm a/b, \tag{4.7}$$

where a, b, c are variables or constants. In fact, the case $y = a + bc$ will only be used in the case where a is y. Moreover, $y = y + x$ will be written as $y \mathrel{+}= x$ (in the same fashion, $y = y - x$ will be written as $y \mathrel{-}= x$). This is standard C notation, and is very useful for differentiation in reverse mode.

Example: If A, B and C are $n \times n$ matrices, the SLP formed by the n^2 instructions $C_{ij} = 0$ and the n^3 instructions $C_{ij} \mathrel{+}= A_{ik} B_{kj}$ computes the matrix product $C = AB$.

4.2.3 Differentiation in direct mode

Let ∇g be the gradient of g, the matrix with entries $\partial g_i/\partial x_j$ if g is a function of $(x_1,\dots,x_n)$ with components $(g_1,\dots,g_p)$. The chain rule says

$$\nabla f = \nabla f_{q+1}\nabla f_q \cdots \nabla f_1 \nabla f_0. \tag{4.8}$$

Differentiating equations (4.5) gives a new set of function calls, hence a new SLP. It has $2n$ inputs variables $(x_1,\dots,x_n,x'_1,\dots,x'_n)$, $2m$ output variables $(y_1,\dots,y_m,y'_1,\dots,y'_m)$, $2p$ local variables $(z_1,\dots,z_p,z'_1,\dots,z'_p)$ and $2q$ function calls $G_1,\dots,G_{2q}$, described below.

According to the previous definition, the state vector of the new SLP is (x,x',y,y',z,z'). We prefer changing the order of variables and use $W=(x,y,z,x',y',z')$ instead. In particular we have $G_{2i}=F_i$. Moreover, the equivalent of (4.5.b) and (4.5.c) is

$$\begin{array}{rcl} g_0:\mathbb{R}^{2n} & \to & \mathbb{R}^{2N} \\ (x,x') & \to & (x,0,0,x',0,0) \end{array} \tag{4.9.b}$$

and

$$\begin{array}{rcl} g_{2q+1}:\mathbb{R}^{2N} & \to & \mathbb{R}^{2m} \\ (x,y,z,x',y',z') & \to & (y,y') \end{array} \tag{4.9.c}$$

Finally, G_{2i-1} is the derivative of f_i. The equivalent of (4.5.a) is

$$\begin{array}{rcl} w'_i & = & w_i \qquad (i\neq t_0+N) \\ w'_{t_0+N} & = & \displaystyle\sum_i \frac{\partial f}{\partial x_i}(w_{t_1},\dots,w_{t_k})w_{t_i+N}. \end{array} \tag{4.9.a}$$

Theorem 33
The SLP defined by equations (4.9) computes a function $(x,x')\to(y,y')$, where $y=f(x)$ and $y'=\nabla f(x).x'$ (the derivative of f at x in the direction x'). If T multiplications are required for computing f, then at most $3T$ multiplications are required for this SLP.

Note: In the case of division, $y=a/b$, the derivative is $y'=a'/b-ab'/b^2$. Instead of computing y', then y, we compute y, and then y' with the formula $y'=(a'-b'y)/b$. This needs one less division.

4.2.4 Reverse mode

Transposing Equation (4.8) gives

$$\nabla f^t = \nabla f_0^t\,\nabla f_1^t \cdots \nabla f_q^t\,\nabla f_{q+1}^t. \tag{4.10}$$

Hence, we get a new SLP, that takes x and y' as input, and computes $y=f(x)$ and $x'=\nabla f^t(x).y'$. In the special case where f is a scalar function (one output), we can take $y'=1$, and this gives the derivative of f at x.

The important point to notice is that the product in (4.10) is done from right to left, i.e. in the reverse order of original computation. Assume that $i<j$, f_i and f_j depend on a, but a is modified by some call f_k $(i<k<j)$. In direct mode, nothing special happens, but in reverse mode, we call f_i, then f_k, then f_j. We compute derivatives of f_j, then f_k, and finally f_i. At this moment, the value of a is wrong. We have to reset the old value. A good moment is when we compute the derivative of f_k. The old value can be found on the stack, if we push it before evaluating f_k. In some cases, it is possible to get the old value from the new one. The typical case is when we compute a sum $\sum_i f(x_i)$. In the code of the loop, we increment i. In reverse mode we decrement i; note that, in this case, the order of evaluation is irrelevant (in the case $g\tilde{q}=Vq+R$, the order of the loop in the product $g\tilde{q}$ is irrelevant, the order in

the division loop is imposed). In general, the total memory used in reverse mode can be large, because of this. There are techniques that reduce memory usage, trading time against space, in other words, we recompute some quantities instead of saving them. In HYPERION, this is not needed.

We assume here that the code has been modified in such a way as no variable is set more than once. We construct here an SLP that computes the derivative. As in the direct case, we consider variables x', y' and z' with the same size as x, y and z, and we still define $W = (x, y, z, x', y', z')$. The SLP will have $n + m$ input variables x, y', $n + m$ output variables y, x' and $2p$ local variables (z, z'). It has a certain number of function calls.

By definition of the input and output variables, the equivalent of (4.5.b) and (4.5.c) is

$$\begin{aligned} g_0 : \mathbb{R}^{n+m} &\rightarrow \mathbb{R}^{2N} \\ (x, y') &\rightarrow (x, 0, 0, 0, y', 0) \end{aligned} \tag{4.11.1}$$

and

$$\begin{aligned} g_l : \mathbb{R}^{2N} &\rightarrow \mathbb{R}^{n+m} \\ (x, y, z, x', y', z') &\rightarrow (y, x'). \end{aligned} \tag{4.11.2}$$

The first q function calls are

$$G_i = F_i \qquad (1 \le i \le q). \tag{4.11.3}$$

There are some additional function calls. For each i, for q, $q - 1$, down to 1, if the function f associated to f_i has k input variables, we have $k + 1$ function calls G_{ij}. Each function call has the form

$$W'_{t_j+N} = (1 - \delta_{t_0 t_j}) W_{t_j+N} + \frac{\partial f}{\partial x_j}(W).W_{t_0+N}. \tag{4.11.4}$$

$$W'_i = W_i \text{ if } i \notin \{t_0 + N, t_1 + N, \ldots, t_k + N\}. \tag{4.11.5}$$

If we compose, we get

$$W' = G_{i,0} G_{i,1} G_{i,2} \cdots G_{i,k}(W). \tag{4.11.6}$$

We added a function call G_{i0} that corresponds to $j = 0$, the output of f. The partial derivative of f with respect to this variable is zero, so that (4.11.4) gives $W'_{t_0+N} = 0$. Note: in most of the cases, this equation is useless, and will be omitted (it is useless in case $W'_{t_0+N} = 0$ is not used).

Assume first that t_0 is not an input variable. Then $\delta_{t_0 t_j}$ is zero, so that (4.11.4) just increments W'_{t_j+N} by the partial derivative of f, with respect to x_j, multiplied by W_{t_0+N}. On the other hand, in the case where t_0 is an input variable, if $t_0 = t_j$, equation (4.11.4) just means: replace W'_{t_0+N} by the partial derivative of f, multiplied by W_{t_0+N}. It can happen that t_0 is used more than once as an input variable. Then we replace W_{t_0+N} by the sum of all partial derivatives, multiplied by W_{t_0+N}. In these cases, we do not replace W_{t_0+N} by zero at the end. Note: one difficulty when trying to differentiate a function call, like `call f(x,y,z)` in Fortran, is that we do not know which variables are input variables, and in the case of arrays, which part of the array is an input.

Example: the derivative, in reverse mode of $y = ab$ is $a' \mathrel{+}= by'$, $b' \mathrel{+}= ay'$ and $y' = 0$. The derivative of $y \mathrel{+}= ab$ is $a' \mathrel{+}= by'$, $b' \mathrel{+}= ay'$. (We assume here that a, b and y are three different variables).

Note: the derivative of $y = a/b$ is a priori $a' \mathrel{+}= y'/b$, $b' \mathrel{-}= ay'/b^2$, $y' = 0$. We can write this as $t = y'/b$, $a' \mathrel{+}= t$, $b' \mathrel{-}= yt$. Hence one additional multiplication and one additional division are required. An additional variable t is used. In the code, it will be `tmp`.

Theorem 34
The SLP defined by equations (4.11) *takes as input* (x, y') *and computes* $y = f(x)$ *and* $x' = \nabla f^t(x).y'$. *If* T *multiplications are required for computing* f*, then this SLP uses* $3T$ *multiplications.*

This theorem was first stated in [14] as: if $f(x_1, \dots, x_n)$ is a polynomial function of n variables, and can be computed in time T, then f and all its partial derivatives can be computed in times less than $3T$. This remains true in the case of division (see results at the end of the next chapter; the ratio is bit larger than 3, because we recompute some quantities, instead of saving them).

Caveat: In the case where a variable is modified more than once in the original code, you can either use the same technique, storing what is necessary (this costs memory) or recomputing it (this costs time). Note that, in the code of the matrix multiplication $C = AB$ given above, we can pretend that C_{ij} is only modified once.

4.2.5 Complex numbers

Nothing special happens in direct mode if we replace real numbers by complex numbers. However, in reverse mode, if $y = ab$, b is constant, then the mapping $a \to y$ is linear. Its matrix is the matrix $\begin{pmatrix} u & -v \\ v & u \end{pmatrix}$, in case $b = u + iv$. If we transpose this matrix, we obtain the matrix of the multiplication by the complex conjugate of b. This means that the code of the derivative contains functions calls that do not appear in the initial code. There is another technique, in which the derivative of the product ab does not involve complex conjugates. These appear nevertheless, when a couple of real numbers is transformed into a complex number.

In the following table we give the derivatives in reverse mode of some usual cases. Here $R(z)$ and $I(z)$ are the real and imaginary parts of z, while $C(a, b)$ is the function that computes $a + ib$.

$$y = C(a,b) \qquad a' \mathrel{+}= R(y'), b' \mathrel{+}= I(y'), y' = 0 \tag{4.12.1}$$

$$a = R(y) \qquad y' \mathrel{+}= C(a', 0), a' = 0 \tag{4.12.2}$$

$$b = I(y) \qquad y' \mathrel{+}= C(0, b'), b' = 0 \tag{4.12.3}$$

$$y = \overline{z} \qquad z' \mathrel{+}= \overline{y'}, y' = 0 \tag{4.12.4}$$

$$y = a + b \qquad a' \mathrel{+}= y', b' \mathrel{+}= y, y' = 0 \tag{4.12.5}$$

$$y = ab \qquad a' \mathrel{+}= \bar{b}y', b' \mathrel{+}= \overline{a}y', y' = 0 \tag{4.12.6}$$

$$y = \overline{a}b \qquad a' \mathrel{+}= b\overline{y}', b' \mathrel{+}= ay', y' = 0 \tag{4.12.7}$$

$$y \mathrel{+}= |a|^2 \qquad a' \mathrel{+}= 2a\Re(y') \tag{4.12.9}$$

$$y = a/b \qquad t = a/b, t' = y'/\bar{b}, a' \mathrel{+}= t', b' \mathrel{-}= \bar{t}t', y' = 0 \tag{4.12.10}$$

4.2.6 Matrices

Nothing special happens here, essentially because we shall replace operations on matrices by operations on the entries of the matrices. It is however noteworthy to see that, in reverse mode, the derivative of the matrix product $Y \mathrel{+}= AB$ is

$$A' \mathrel{+}= Y'B^* \qquad B' \mathrel{+}= A^*Y'. \tag{4.13}$$

Recall that A^* is the transpose conjugate of A.

4.2.7 The case of polynomials

One can notice that, if $t_{nm}(X)$ is the remainder by z^{n+1} of the quotient of X by z^m, then the derivative in reverse mode of the polynomial product $Y \mathrel{+}= AB$, where A has degree n and B has degree m, is

$$A' \mathrel{+}= t_{nm}(\tilde{B}Y') \qquad B' \mathrel{+}= t_{mn}(\tilde{A}Y'). \tag{4.14}$$

This relation is not used (essentially because $\tilde{B}Y'$ contains useless terms).

The case of division of polynomials is interesting. We leave it as an exercise to the reader to show that the following pseudo C code

```
void division(A,q,B,R)
{
   for(i=0;i<=n+m;i++)
       R[i] = A[i];
   for(i=0;i<=m;i++) {
      t = R[n+m-i] / q[n];
      B[m-i] = t;
      for(j=0;j<n;j++)
          R[m-i+j] -= t*q[j];
   }
}
```

computes in B and R the quotient and the remainder of the division of A by q, assuming that q is of degree n, and B of degree m (i.e. A is of degree $n+m$). The exact C code can be found later, by instantiation of the algorithm shown later in this chapter. Differentiating in reverse mode gives something like

```
void division_prime(...)
{
   for(i=m;i>=0;i--) {
      t = B[m-i];
      t' = B'[m-i];
      for(j=0;j<n;j++) {
          t' = q[j]+ R'[m-i+j]; /* H */
          q'[j] -= t*R'[m-i+j]; /* H */
      }
      B'[m-i] = t';
      t' = t'/q[n];       /* H */
      R'[n+m-i] -= t';
      q'[n] -= t*t';     /* H */
   }
}
```

In the complex case, we have to replace the factors of `t'` and `R'[m-i+j]` by their complex conjugates in the lines with the comment ‘H’. For the exact code, see table 4.12.

If we differentiate $A = Bq + R$ in direct mode, we get

$$\delta A - B\delta q = q\delta B + \delta R.$$

This means that δB and δR are the quotients and remainder in the division by q of $\delta A - B\delta q$.

If we differentiate in reverse mode, strange things happen. Assume $A = \sum a_k z^k$ $B = \sum b_k z^k$ and $q = z - \beta$. Define c_i by $c_0 =$ and $c_{i+1} = b_i$. Then

$$c_i = \sum_k a_{i+k}\beta^k.$$

If we differentiate, we obtain

$$da_{i+k} = \beta^k dc_i, \quad d\beta = \sum k a_{i+k}\beta^{k-1} dc_i.$$

The first equation can be written as

$$dA = \frac{dR + zdB}{1 - z\beta} + o(z^n),$$

where the notation $o(z^n)$ means that dA is the truncation to n terms of the long division of the numerator by the denominator.

If we denote by $A^{(k)}(\beta)$ the derivative, evaluated at β of the quotient of A by z^k, then

$$d\beta = \sum A^{(k)}(\beta) dc_k.$$

There is no general and simple formula for dq. However dA is defined by

$$\tilde{q} dA \bmod d_A = \tilde{q} dR \bmod d_A + z^n dB.$$

In this expression, $\tilde{q} dA \bmod d_A$ is the truncation of the product $\tilde{d}A$ with as many terms as A.

Note that the situation is easier in the case of matrix inversion: the derivative in direct mode of $Y = X^{-1}$ is $\delta Y = -Y \delta X Y$, and the derivative in reverse mode is $dX = -Y^* dY Y^*$.

4.3 The WEB system

Assume that we have a function f. For instance $y = f(x)$, where $z = (x+1)^2$ and $y = (z+2)^2$. Our differentiator generates the code of the function and its derivative. In the reverse mode of differentiation, we have to generate the following

```
z = (x+1)*(x+1);      /* (E1) */
y = (z+2)*(z+2);      /* (E2) */
dz = 2*(z+2)*dy;      /* (E3) */
dx = 2*(x+1)*dz;      /* (E4) */
```

The differentiator handles the first assignment $z = (x+1)^2$, then the second $y = (z+2)^2$, and generates the four lines (E_i) in the order (E_1), (E_4), (E_2) and (E_3), and we have to put them in the right order. There are three possible strategies:

- reordering done by the differentiator,
- reordering done by the C code,
- reordering done by an external program.

In general, automatic differentiators do the re-ordering. This means that they have to memorise the whole code. Reordering can also be done by the C program, as shown here (`push` and `pop` are macros that push and pop a value on a stack).

```
push(0);
z = (x+1)*(x+1);      /* (E1) */
push(1);
y = (z+2)*(z+2);      /* (E2) */
push(2);
for(;;) {
   switch(pop()) {
     case 1:
dx = 2*(x+1)*dz;      /* (E4) */
     break;
     case 2:
dz = 2*(z+2)*dy;      /* (E3) */
     break;
   default:
     goto done;
 }
}
done:;
```

Note that in this example, the differentiator still has to memorise the whole code. However, this is the easiest way to handle the case of wild transfer of control. We do not know the loss of performance of such a scheme for a real program. See for instance [7]; in this test case the overhead of memory management (execution stack and data stack) is so great that comparison with other methods is uneasy.

We have chosen the last solution. The external program is called `ctangle`, which is a part of the `CWEB` system. The description of CWEB can be found for instance in [12, 15]. Note that the `WEB` system was designed by Knuth for TEX, and the HYPERION software is entirely written in `WEB`. The only feature we use here is the macro expansion system: the differentiator generates some macros (called *sections*), and `tangle` re-arranges the code. An associated program, `weave`, pretty prints the source code, expanding all LATEX macros.

This is the code of our function, in `WEB` format.

```
@ This is the code of the function $f$, defined by equation \ref{the-equation}.
@u double f (double x)
{
   double y,z;
   @<First part of f@>
   @<Second part of f@>
   return y;
}
```

And this is now the code of the derivative.

```
@ This is the code of the derivative of $f$ in reverse mode.
@u double fprime (double x,double dy)
{
   double y,dx,z,dz;
   @<First part of f@>
   @<Second part of f@>
   @<Second part of derivative of f@>
   @<First part of derivative of f@>
   return y;
}
```

Our differentiator generates now, for the part $z = (x+1)^2$:

```
@ @<First part of f@>=
  z = (x+1)*(x+1);
@ @<First part of derivative of f@>=
  dx = 2*(x+1)*dz;
  dz = 0;
```

and for the part $y = (z+2)^2$:

```
@ @<Second part of f@>=
  y = (z+2)*(z+2);
@ @<Second part of derivative of f@>=
  dz = 2*(z+2)*dy;
  dy = 0;
```

If we put these sections together, `tangle` gives us the following C code. (note the `#line` commands and the `//:N` comments: they are very useful for debugging; we removed them in the code of `fprime`).

```
//1:
```

```
#line 2 "toto.web"
double f(double x)
{
double y,z;
//3:

#line 18 "toto.web"

#line 19 "toto.web"
z= (x+1)*(x+1);
//:3

#line 5 "toto.web"

//5:

#line 23 "toto.web"

#line 24 "toto.web"
y= (z+2)*(z+2);
//:5

#line 6 "toto.web"

return y;
}
//:1
//2:

double fprime(double x,double dy)
{
double y,dx,z,dz;
z= (x+1)*(x+1);
y= (z+2)*(z+2);
dz= 2*(z+2)*dy;
dy= 0;//:6
dx= 2*(x+1)*dz;
dz= 0;
return y;
}
```

4.4 Naming scheme

A very important point in automatic differentiation is the naming scheme that will be used. We have to give a name to functions, variables, and sections. If the name of the section that computes $z = (x+1)^2$ is 'First part of f', then we must generate automatically the name associated to the derivative of the section. Getting 'First part of derivative of f' is not trivial. For this reason, we shall use 'First part of f, diff' instead, i.e. we add the *mode name*, as defined in table 4.1. We have four modes, because we compute ψ, ψ' and ψ'', and the derivative of ψ has to be computed twice (see later). The name 'diff' is for differentiation in reverse mode, the name 'delta' for differentiation in direct mode.

Table 4.1: Table of modes

	mode name	variable prefix	name
0	direct	(no prefix)	x
1	diff	`d`	dx
2	delta	`delta_`	δx
3	delta diff	`delta_d`	δdx

Table 4.2: Table of precisions

C type	precision name	precision	suffix
`double`	double	8 bytes	`D`
`SLD`	SLD	20 bytes	`S`
`complex`	complex	16 bytes	`C`
`SLD_complex`	SLD complex	40 bytes	`SC`

The derivative of a variable x cannot be called x', because this is an illegal name in C. For this reason, we add a prefix, and use dx. When we differentiate in direct mode, we add another prefix, and use δx. Finally, if we differentiate twice, we add a double prefix δdx. Of course, δ is not a valid letter in C, so that the real name will be `delta_dx`.

Consider the simple equation:

$$c \mathrel{+}= ab. \tag{4.15.a}$$

The derivative in reverse mode is

$$a' \mathrel{+}= bc', \qquad b' \mathrel{+}= ac'. \tag{4.15.b}$$

Differentiating again gives

$$c'' \mathrel{+}= ba'', \quad a' \mathrel{+}= c'b'', \quad b' \mathrel{+}= c'a'', \quad c'' \mathrel{+}= ab''. \tag{4.15.c}$$

No naming scheme is applied here, so that we have two different variables with the same name a' (and the same for b'). In fact, one a' is $\partial a/\partial x_i$, and the other is $\partial a/\partial x_j$ if we compute row i, column j of the Hessian. Note that the Hessian is computed by differentiating in direct mode the code of ψ', which is computed by differentiating in reverse mode the code of ψ. Differentiating (4.15.b) in direct mode gives

$$a'' \mathrel{+}= b'c' + bc'', \qquad b'' \mathrel{+}= a'c' + ac''. \tag{4.15.d}$$

If we use the naming scheme shown above, we get

$$da \mathrel{+}= b\, dc, \qquad db \mathrel{+}= a\, dc \tag{4.15.e}$$

$$\delta da \mathrel{+}= \delta b\, dc + b\, \delta dc, \qquad \delta db \mathrel{+}= \delta a\, dc + a\, \delta dc. \tag{4.15.f}$$

$$\delta c \mathrel{+}= \delta a\, b + a\, \delta b. \tag{4.15.g}$$

There is no more ambiguity in these equations.

For some reasons, computation in double precision is not precise enough. This means that we need quadruple precision. Moreover, we want our algorithms to work in the real and complex case. This means that each formula is implemented four times. In table 4.2, we give for each precision its name (this name will be added to section names), the suffix to add to the functions we generate or call, and also to the name of some global variables, and the the number of bytes of memory used by each type. Each entry in this table is in one of the Lisp arrays `precision_names`, `prec_types`, `types-vector`.

4.5 Web interface

From now on, we shall give the source of the code of the differentiator, and everything else that is needed. The code uses a number of global variables that are not shown here (for instance the tables shown before are implemented as some vectors). We have also some variables, `at-start`, `at-equal` and `at-semi` that contain the quantities `@<`, `@>=` and `@>@;`.

Each piece of code we generate exists for each mode, each precision. In general, each section has the form: `@<`some title, current mode, current precision `@>`. The next function generates a section definition, it takes the current mode and precision from global variables. The `show-usage` function prints a dot on the terminal after every ten calls.

```
(defun :decl-sec (T comm)
    (show-usage)
    (print "@ " comm)
    (print at-start T (vref mode_names mode) (vref precision_names prec) at-equal)
```

The same, without comments.

```
(defun decl-sec (T)
  (show-usage)
  (print "@ @<" T (vref mode_names mode) (vref precision_names prec)
         at-equal))
```

This one is when we use a section.

```
(defun use-sec (T)
    (print at-start T (vref mode_names mode) (vref precision_names prec)
  at-semi))
```

These two functions are used in case the section is mode-independent (depend only on the current precision).

```
(defun decl-sec0 (T prec)
  (show-usage)
  (print "@ @<" T (vref precision_names prec) at-equal))

(defun use-sec0 (T prec)
  (print at-start T (vref precision_names prec) at-semi))
```

Case where the name is independent of the precision. We add a dot at the end of the name, because it is not possible to have two sections A and B such that A is a prefix of B.

```
(defun decl-sec1 (T mode)
  (show-usage)
  (print "@ @<" T (vref mode_names mode) "." at-equal))

(defun use-sec1 (T mode)
  (print at-start T (vref mode_names mode) "." at-semi))
```

According to (4.12.6), in the complex case, the first equation of (4.15.f) is

$$\delta da \mathrel{+}= \overline{\delta b}\, dc + \overline{b}\, \delta dc. \tag{4.16.a}$$

We replace this by

$$\delta da \mathrel{+}= \overline{\delta b}\, dc, \qquad \delta da \mathrel{+}= \overline{b}\, \delta dc. \tag{4.16.b}$$

Let $f(x, y, z)$ be the function (or macro) that replaces z by $z + \overline{x}y$. Then we have

$$f(\delta b, dc, \delta da), \qquad f(b, \delta dc, \delta da). \tag{4.16.c}$$

The C code we generate is:

Table 4.3: Operator table

opcode	arguments	meaning	num	db
`+`	(a, b, c)	$c \mathrel{+}= ab$	0, 1	15
`-`	(a, b, c)	$c \mathrel{-}= ab$	1, 0	15
`_+`	(a, b, c)	$c \mathrel{+}= \overline{a}b$	2, 3	15
`_-`	(a, b, c)	$c \mathrel{-}= \overline{a}b$	3, 2	15
`/`	(a, b, c)	$c = a/b$	4	15
`_/`	(a, b, c)	$c = a/\overline{b}$	5	15
`=`	(a, b)	$b = a$	6	
`1*`	(a, b)	$b \mathrel{+}= \lvert a\rvert^2$	7, 12	16
`2*`	(a, b, c)	$c \mathrel{+}= 2\Re(\overline{a}b)$, c real	8, 13	17
`3*`	(a, b, c)	$c \mathrel{+}= 2ab$, b real	9, 14	18
`++`	(a, b, c)	$c = a + b$	10	
`--`	(a, b, c)	$c = a - b$	11	

```
c_add_mult_conj_mac(delta_b,dc,delta_da);
c_add_mult_conj_mac(b,delta_dc,delta_da);
```

In `C++`, this becomes

```
delta_da.add_mul_conj(delta_b,dc);
delta_da.add_mul_conj(b,delta_dc);
```

Instead of using a C macro, we can use a Lisp macro, or a Web macro. It happens that every function call generated by the differentiator has this form (maybe with two arguments). The complete list (using the `C` names) is given in table 4.4. As seen in the previous case, we can use a `C++` method, instead of a function call.

Each function in this table is identified by its row and column index. The column index is the current precision. The row index is specified by Table 4.3. In this one, 'opcode' is a 1 or 2 character opcode that identifies the function. We show the arguments of the function, the meaning, and a number. For instance, the function f above has opcode `_+`. Its number is two. There is a second number, 3, that corresponds to the same operation, with $+=$ and $-=$ exchanged (this is needed in order to implement equation (2.58)). Last column was only used for debugging.

4.6 Parsing arguments

What we have to differentiate is

$$G\tilde{D} = Vq + R, \qquad \psi = \|V\|^2 \tag{4.17}$$

or the modified version of it

$$G = V_1 q + R_1 \qquad R_1\tilde{D} = V_2 q + R_2, \tag{4.18.a}$$

$$\psi = \|F\|^2 - \|R_1\|^2 + \|V_2\|^2, \tag{4.18.b}$$

and the Schur formulas (4.4) on page 102.

The operations that are used in these formulas are addition, multiplication and division of matrices. We also need the tilde operator. Note that an expression like $\tilde{x}$ appears in four cases

- In (4.4), $\tilde{D}_A$ and $\tilde{D}_B$ are just funny names, no tilde operation is required.
- We have $b = (z - \omega)(1 - \overline{\omega})$ and $\tilde{b} = (1 - z\overline{\omega})(1 - \omega)$. These quantities are constant. We could just say that $\tilde{b}$ is another funny name. As explained below, $\tilde{b}$ is not used in the code. [Note: constant means independent of y, not of z.]

Table 4.4: Operators

double	SLD	complex	SLD complex
`double_add_mul`	`SLD_add_mul`	`c_add_mult_mac`	`SLD_add_mul_c`
`double_sub_mul`	`SLD_sub_mul`	`c_sub_mult_mac`	`SLD_sub_mul_c`
`double_add_mul`	`SLD_add_mul`	`c_add_mult_conj_mac`	`SLD_add_mult_conj`
`double_sub_mul`	`SLD_sub_mul`	`c_sub_mult_conj_mac`	`SLD_sub_mult_conj`
`double_div`	`SLD_div`	`c_div_mac`	`SLD_cmplx_div`
`double_div`	`SLD_div`	`c_div_conj_mac`	`SLD_div_conj`
`double_set`	`SLD_copy`	`double_set`	`SLD_cmplx_copy`
`double_square`	`SLD_square`	`cmplx_square`	`SLD_cmplx_square`
`double_2times`	`SLD_2times`	`cmplx_2times`	`SLD_cmplx_2times`
`double_2times`	`SLD_2times`	`cmplx_2times_r`	`SLD_cmplx_2times_r`
`double_add_mac`	`SLD_add`	`cmplx_add_mac`	`SLD_cmplx_add`
`double_sub_mac`	`SLD_sub`	`cmplx_sub_mac`	`SLD_cmplx_sub`
`double_square_neg`	`SLD_square_neg`	`cmplx_square_neg`	`SLD_cmplx_square_neg`
`double_2times_neg`	`SLD_2times_neg`	`cmplx_2times_neg`	`SLD_cmplx_2times_neg`
`double_2times_neg`	`SLD_2times_neg`	`cmplx_2times_r_neg`	`SLD_cmplx_2times_r_neg`

- In the scalar case, we have $D = q$. Thus, we have to compute $\tilde q$. We do not use automatic differentiation in the scalar case.
- Lemma 36 shows that the division in (4.4) is exact. In the case $p = 2$, we have $Y/q_A = \tilde q_A(uy^* - y^*u)$. Thus, we have to compute the product of the scalar $\tilde q$ by the matrix $uy^* - y^*u$.

Since b is a polynomial of degree one, it has the form $b_0 + b_1 z$, so that $\tilde b = \overline{b}_1 + \overline{b}_0 z$. Let $b_2 = b - \tilde b$ and $b_4 = b - \tilde b \|y\|^2$. Computing b_2 and b_4 is done using a special piece of code (see section 4.11.7). Note that b and $\tilde b$ are used only through b_2 and b_4 (we shall see later that the array B holds b_0 and b_1 at locations 0 and 1, the coefficients of b_2 at location 2 and 3, the coefficients of b_4 at locations 4 and 5, $\|y\|^2$ at location 6, and y^*u at location 7; this explains the names b_2 and b_4).

Now, a careful examination shows that the only operations that are needed are those define in table 4.5, namely

- $c \mathrel{+}= \|a\|^2$. This operation is called ψ, because it is only used to compute ψ. We take advantage of the fact that $d\psi = 1$ when we differentiate this in reverse mode.
- $c \mathrel{+}= ab$, where a is a scalar and b a matrix. This appears in y^*DuD, where the scalar is y^*Du and the matrix is D.
- $c \mathrel{+}= \tilde a b$, where a is a scalar and b a matrix (with $a = q$, $b = uy^* - y^*u$).
- $c = a/b$. This operation replaces a by the remainder of the Euclidean division of a by b, and puts the quotient in c. There are three divisions, in one case we use the remainder, otherwise the quotient.
- $c \mathrel{+}= a\tilde b$ or $c \mathrel{+}= \tilde a b$. It happens that $\tilde a$ and $\tilde b$ are independent of z, so that only $c \mathrel{+}= ab^*$ and $c \mathrel{+}= a^*b$ are needed.
- $c \mathrel{+}= ab$. Here and in the previous case, a and b are matrices or vectors (we exclude the case: scalar times matrix). Moreover, c has the same size and degree as the product ab.
- The same with $\mathrel{-}=$ instead of $\mathrel{+}=$.
- Initialisation (in general to zero, but the initial value of ψ is $\|G\|^2$, i.e. one). This operation is not in the table.

Note that (4.4.1) is

$$q_B = b_4 q_A - b_2 y^* D_A u$$

using the notations b_2 and b_4 defined above. We compute first $y_D = y^* D$, then $E = y_D u$ (we explained in the introduction why we compute it this way, and not the other way). We initialise q_B to zero, add $b_4 q_A$ and subtract $b_2 E$. The first computation is done by

$$\mathrm{diff}([y, T], D, y_D) \tag{4.19.a}$$

and the last by

$$\mathrm{diff}([b_2, -], E, q_A). \tag{4.19.b}$$

Here 'diff' is our differentiator. It takes four arguments, the last being the name of the section to generate (not indicated here). The third argument is a name. The first two arguments are a bit special: they consists of lists of 4 or more elements: name of a variable, first dimension, second dimension, degree. We have some flags: in (4.19.a) we have a flag that says that we use y^*, and in (4.19.b) a flag that says that subtraction should be used instead of addition. In table 4.17 on page 162, the first row is equation (4.19.a), while equation (4.19.b) is '`new q, 2`' (the quantity $b_4 q_A$ is computed by '`new q, 1`'). The parser puts the flags into global variables. The next function clears these globals.

```
(defun :initialise-diff ()
  (setq :transpose-a ())
  (setq :transpose-b ())
  (setq :neg-a ())
  (setq :invert-b ())
  (setq :compute-psi ())
  (setq :constant-flag 0)
  (setq :the-case 0))
```

This is the differentiator. The code is very simple, because split into two parts. The first part checks that the operation is well-defined and finds what to do (function `:check-mat-prod`). The second part (`:print-code`) differentiates the code, essentially by replacing in a table every name by its value.

As said above, the first arguments x and y are lists that contain at least four elements. These will be put into two tables `:param_a` and `:param_b`, remaining stuff will be in global variables. The third argument z is a name. We put it in `:param_c`. The first part will fill completely this vector and return it as a list. Thus the result of the function is a valid argument to itself.

```
(defun differentiate (x y z title)
  (let (res)
    (vset :param_c 0 (catenate z))
    (setq res (:check-mat-prod x y))
    (print "@ Code of \\verb!" x "! and  \\verb!" y "!")
    (print "into \\verb!" res "!.")
    (:print-code :the-case title (vref :patterns-name :the-case))
    res))
```

Each element A and B is a list of 4 mandatory parameters, and some optional flags. The mandatory parameters are: the name of the variable, put in A_0, the first and second dimensions (put in A_1 and A_2), and the degree put in A_3.

This is the start of the function `check-mat-prod`: fetch mandatory parameters, and put them into global variables. For simplicity of use, if y is a list that represents a variable, and T is the 'transpose' flag, we accept $[y, T]$ (a list of two elements) as argument to the function (see equation (4.19.a)). This explains the test `if (consp (car x))`.

```
(defun :check-mat-prod (x y)
  (:initialise-diff)
  (if (consp (car x)) (setq x (append (car x) (cdr x))))
```

```
    (if (consp (car y)) (setq y (append (car y) (cdr y))))
    (vset :param_a 0 (catenate (car x)))
    (setq x (cdr x))
    (vset :param_a 1 (catenate (car x)))
    (setq x (cdr x))
    (vset :param_a 2 (catenate (car x)))
    (setq x (cdr x))
    (vset :param_a 3 (catenate (car x)))
    (setq x (cdr x))
    (vset :param_b 0 (catenate (car y)))
    (setq y (cdr y))
    (vset :param_b 1 (catenate (car y)))
    (setq y (cdr y))
    (vset :param_b 2 (catenate (car y)))
    (setq y (cdr y))
    (vset :param_b 3 (catenate (car y)))
    (setq y (cdr y))
```

Second part of `check-mat-prod`: fetch optional parameters.

```
    (when (and (consp x) (equal (catenate (car x)) "transpose"))
          (setq :transpose-a true x (cdr x)))
    (when (and (consp x) (equal (catenate (car x)) "neg"))
          (setq :neg-a true x (cdr x)))
    (when (and (consp x) (equal (catenate (car x)) "constant"))
          (setq :constant-flag 1 x (cdr x)))
    (when (and (consp y) (equal (catenate (car y)) "transpose"))
          (setq :transpose-b true y (cdr y)))
    (when (and (consp y) (equal (catenate (car y)) "div"))
          (setq :invert-b true y (cdr y)))
    (when (and (consp y) (equal (catenate (car y)) "psi"))
          (setq :compute-psi true y (cdr y)))
    (when (and (consp y) (equal (catenate (car y)) "constant"))
          (setq :constant-flag (+ 2 :constant-flag) y (cdr y)))
```

Last part of `check-mat-prod`: check and construct the result.

```
    (:construct-deg)
    (:construct-result)
    (list (vref :param_c 0)(vref :param_c 1)(vref :param_c 2)(vref :param_c 3)))
```

The flags are used in the following way: first, we put in `constant-flag` an integer: 0 means that A and B are variable, 1 means that A is constant, 2 means that B is constant, and 3 means that A and B are constant (this happens rarely). Other flags (transpose, div, psi) are denoted by T, D and ψ in Table 4.5. These define the operation to be applied, which is characterised by a pattern name (and a pattern number). If `neg-a` is true, then $+=$ should be replaced by $-=$.

In the case D, we assume that a is a matrix of polynomials or a polynomial, and b a polynomial. We compute $a = bq + r$, by Euclidean division, where the quotient q is put in c, and the remainder in a. In the case ψ, we compute $\psi \mathrel{+}= \|a\|^2$, where ψ is a real number. In all other cases, the operation is $c \mathrel{+}= f(a, b)$, or $c \mathrel{-}= f(a, b)$. The minus sign is chosen if the flag N appears in A. After the type of the operation is fetched, we compute the size and dimension of the result, and check that the operation is valid: if we multiply matrix a by matrix b, then the second dimension of a must be the first dimension of b. Exception: we allow the product of a scalar by a square matrix. The next function computes the size of the result (and the degree in the case ψ), and fills the variable `the-case`, which is the pattern to apply. Note: we cannot compute $uy^* - y^*u$ (scalar plus matrix); we explain later how this can be done.

```
  (defun :construct-result ()
    (let ((dim1a (vref :param_a 1))
```

Table 4.5: How flags determine the operation

Flags for A	flags for B	operation	pattern	comments
none	ψ	$c \mathrel{+}= \|a\|^2$	ψ	name and dimensions of B ignored
none	none	$c \mathrel{+}= ab$	S	if a is scalar, b a square matrix
T	none	$c \mathrel{+}= \tilde{a}b$	ST	if a is scalar, b a square matrix
none	D	$c = a/b$	D	b must be a scalar
none	T	$c \mathrel{+}= \widetilde{ab}$	T_b	a and b are vectors
T	none	$c \mathrel{+}= \tilde{a}b$	T_a	
none	none	$c \mathrel{+}= ab$	N	

```
        (dim2a (vref :param_a 2))
        (dim1b (vref :param_b 1))
        (dim2b (vref :param_b 2)))
    (vset :param_c 1 dim1a) ; values OK for pattern N
    (vset :param_c 2 dim2b)
    (cond (:compute-psi
           (vset :param_c 1 "1")
           (vset :param_c 2 "1")
           (vset :param_c 3 "0")
           (setq :the-case 5))
          ((and (equal dim1a "1") (equal dim2a "1") (equal dim1b dim2b))
           (vset :param_c 1 dim1b)
           (setq :the-case (if :transpose-a 4 3)))
          (:invert-b
           (vset :param_c 2 dim2a)
           (setq :the-case 6))
          (:transpose-b
           (if (or (not (equal dim2b "1"))(not (equal dim2a "1")))
               (error 'diff "bad transpose" ()))
           (vset :param_c 1 dim1a)
           (vset :param_c 2 dim1b)
           (setq :the-case 2))
          (:transpose-a
           (if (not (equal dim1a dim1b))
               (error 'diff "bad" "dim"))
           (vset :param_c 1 dim2a)
           (setq :the-case 1))
          (true
           (if (not (equal dim2a dim1b))
               (error 'diff "bad" "dim"))
           (setq :the-case 0)))))
```

The next function computes the degree of the result. It is called before the preceding function, and does not compute the degree in the case of ψ. Moreover, in the case $c = a/b$ (Euclidean division of a by b), it checks that b is a scalar, of non-zero degree.

```
(defun :construct-deg ()
  (let ((dega (vref :param_a 3)) (degb (vref :param_b 3)))
    (if :invert-b
         (progn
          (when (or (not (equal (vref :param_b 1) "1"))
                    (not (equal (vref :param_b 2) "1"))
                    (equal degb "0"))
```

```
        (error ’diff "illegal parameters" 0))
     (vset :param_c 3 (:deg-sub dega degb)))
   (vset :param_c 3 (:deg-add dega degb)))))
```

In some cases, it is important to know that the result has zero degree. Hence the following function thats computes the sum of two degrees.

```
(defun :deg-add (a b)
 (cond ((equal a "0") b)
    ((equal b "0") a)
    (true (catenate a "+" b))))
```

In the case of division, the degree of the result (the quotient) is $n-m$, and we know that $m>0$. This function gives p in case n has the form $p+m$ (typical use: we compute the remainder of gd by q where d and q have the same degree).

```
(defun :deg-sub (a b)
 (let ((test true))
  (let ((i (strlen a)) (j (strlen b)) k)
   (setq k (- i j))
   (if (and (> k 0) (= (sref a (1- k)) #/+) (substring-eq a k b 0 j))
     (substring a 0 (1- k))
    (catenate a "-" b)))))
```

4.7 The patterns

The only function to explain now is `print-code`. This function takes a pattern, instantiates it, and prints the result. We start with the list of patterns.

If we look at table 4.5, we see that we have three kinds of operations. The case D is a bit special, since it takes two arguments a and b, computes the quotient and remainder of the Euclidean division of a by b, puts the quotient in c, and the remainder in a. It is the only operation that modifies its input parameter (thus, if we use (4.18.a), we have to copy G somewhere). Other differences with the general case will be explained later. The case ψ is also special, because it computes ψ, the result of the function. Since $d\psi = 1$, we could simplify the code (this is not yet done). In fact, the main difference with the general case is that it takes only one argument.

Consider a general pattern, for instance T_b. We compute $C = A\tilde{B}$, where A and B are vectors of polynomials. Let A_{ij} and B_{ij} be the coefficient of z^j of entry i of A or B. Let C_{ijk} be the coefficient of z^k of entry (i,j) of C (if A has size n_a and B has size n_b, then C is an $n_a \times n_b$ matrix). Equation $C = A\tilde{B}$ can be written as

$$C_{klw} = \sum_{i+j=w} \overline{B}_{l,m-j} A_{ki}.$$

We can write this as a sequence of assignments: $C_{klw} = 0$, followed by

$$\forall i,j,k,l \quad C_{kl,i+j} \mathrel{+}= \overline{B}_{l,m-j} A_{ki}. \tag{4.20}$$

This equation is translated into the first five rows of the pattern for T_b. For instance, the first line is `k<1a`. This line just says that $0 \le k < n_a$, where n_a is the number of rows of A. The third line `i<=3a` says $0 \le i \le d_a$, where d_a is the degree of A. Thus the first four lines of the pattern give the range of the variables that are quantified in equation (4.20). The body of (4.20) itself is coded as `_+ bt a c`. This is a short-hand for

$$c \mathrel{+}= \overline{bt}a.$$

Here a, bt and c are abbreviations for A_{ki}, $B_{l,m-j}$ and $C_{kl,i+j}$, see table 4.13 on page 126.

If we differentiate equation (4.20), in direct mode, reverse mode, or twice, we get something like $\forall i,j,k,l,E$, where E is a list of assignments, and i, j, k, and l take the same values as before. For this

Table 4.6: The patterns for N

k < 1a w < 2a l < 2b i <= 3a j <= 3b
+ a b c
+ delta_a b delta_c + a delta_b delta_c
_+ b dc da _+ a dc db
_+ delta_b dc delta_da _+ b delta_dc delta_da _+ delta_a dc delta_db _+ a delta_dc delta_db

Table 4.7: The patterns for T_a

k < 2a w < 1a l < 2b i <= 3a j <= 3b
_+ at b ct
_+ delta_at b delta_ct _+ at delta_b delta_ct
_+ dct b dat + at dct db
_+ dct delta_b delta_dat _+ delta_dct b delta_dat + delta_at dct delta_db + at delta_dct delta_db

Table 4.8: The patterns for T_b

k < 1a l < 1b i <= 3a j <= 3b
_+ bt a c
_+ delta_bt a delta_c _+ bt delta_a delta_c
_+ dc a dbt + bt dc da
_+ dc delta_a delta_dbt _+ delta_dc a delta_dbt + delta_bt dc delta_da + bt delta_dc delta_da

Table 4.9: The patterns for S

l < 1b * 2b i <= 3a j <= 3b
Other lines like N.

Table 4.10: The patterns for ST

l < 1b * 2b i <= 3a j <= 3b
_+ at b c
_+ delta_at b delta_c _+ at delta_b delta_c
_+ dc b dat + at dc db
_+ dc delta_b delta_dat _+ delta_dc b delta_dat + delta_at dc delta_db + at delta_dc delta_db

Table 4.11: The patterns for ψ

k < 1a * 2a i <= 3a
1* a cp
2* delta_a a delta_cp
3* a dcp da
3* delta_a dcp delta_da 3* a delta_dcp delta_da

Table 4.12: The patterns for D

k < 1a * 2a
for(i=0;i<=, degc, ;i++) / a1 b1 tmp = tmp c2 for(j=0;j<, degb, ;j++) - tmp b2 a2

for(i=0;i<=, degc, ;i++)
= c2 tmp
= delta_a1 delta_tmp
- tmp delta_b1 delta_tmp
/ delta_tmp b1 delta_tmp
= delta_tmp delta_c2
for(j=0;j<, degb,;j++)
{
- delta_tmp b2 delta_a2
- tmp delta_b2 delta_a2
}

for(i=, degc, ;i>=0;i--)
= c2 tmp
= dc2 dtmp
for(j=0;j<, degb, ;j++)
{
_- b2 da2 dtmp
_- tmp da2 db2
}
= dtmp dc2
_/ dtmp b1 dtmp
++ dtmp da1 da1
_- tmp dtmp db1

for(i=, degc, ;i>=0;i--)
= c2 tmp
= dc2 dtmp
= delta_c2 delta_tmp
= delta_dc2 delta_dtmp
for(j=0;j<, degb, ;j++)
{
_- delta_b2 da2 delta_dtmp
_- b2 delta_da2 delta_dtmp
_- delta_tmp da2 delta_db2
_- tmp delta_da2 delta_db2
}
= delta_dtmp delta_dc2
_/ dtmp b1 dtmp
_- delta_b1 dtmp delta_dtmp
_/ delta_dtmp b1 delta_dtmp
++ delta_dtmp delta_da1 delta_da1
_- delta_tmp dtmp delta_db1
_- tmp delta_dtmp delta_db1

reason, the pattern is split into five parts C_i. There is a Lisp file that contains a list of 7 patterns, one for each row of table 4.5. These are shown in the tables 4.6 to 4.12. Each pattern is formed by 5 chunks of lines. In the Lisp file, each chunk is a list, with a name (for instance, `pat-N-head`, `pat-N-code`, etc.) and a pattern is a list of 5 lists. Chunk C_0 is the header (describing the loops), chunk C_i is the code for the ith mode of derivation (direct, delta, diff, delta-diff).

The next function takes as argument the content of the whole Lisp file and interprets it.

```
(defun read-patterns1 (l)
  (let (x)
    (setq :patterns (makevector 7 ()))
    (setq :patterns-name (makevector 7 ()))
    (for (i 0 1 6)
         (setq x (car l) l (cdr l))
         (vset :patterns i (:read-one-pattern x i)))
    l))
```

If the chunk has more than one instruction, curly braces are added. In the special case of division, the opening brace is added after the first ‘for’ instruction (braces after the second ‘for’ are explicit, see table 4.12). The constants `open-brace` and `close-brace` contain these braces.

This is the code that adds braces around a chunk.

```
(defun :add-a-brace (l)
 (if (not (cdr l))
     l
   (if (and (consp (car l)) (stringp (car (car l)))
            (substring-eq (caar l) 0 "for" 0 3))
       (cons (car l) (cons open-brace (append1 (cdr l) close-brace)))
     (cons open-brace (append1 l close-brace)))))
```

This function converts the list of 5 lists, into a single list, and adds ‘nil’ markers between chunks. It also adds a brace wherever needed.

```
(defun :add-braces (l)
  (let (LL a b c d e)
    (setq LL (list ())) ; chunk separator
    (setq a (car l)
          b (:add-a-brace (cadr l))
          c (:add-a-brace (caddr l))
          d (:add-a-brace (cadddr l))
          e (:add-a-brace (car (cddddr l))))
    (append a LL b LL c LL d LL e LL)))
```

The next function reads a single pattern. The first element is the name of the pattern, stored in the table.

```
(defun :read-one-pattern (l i)
  (let (res)
    (setq l (:add-braces l))
    (vset :patterns-name i (car l))
    (setq l (cdr l))
    (while (car l)
      (setp p (makevector 5 ()))
      (newl res p)
      (:fetch-for-pattern (car l) p)
      (setq l (cdr l)))
    (while l
      (setq p (makevector 5 ()))
      (vset p 1 0)
      (newl res p)
      (:fetch-normal-pattern (car l) p)
      (setq l (cdr l)))
     (nreverse res)))
```

Each line in a pattern is translated into a vector of size 5. The meaning of these fields depend on whether this is chunk C_0, or chunk C_i. The first chunk C_0 encodes the 'for' loops. For instance, if we compute the product $C = AB$, where A, B and C are matrices of polynomials, A_{ijk} is the coefficient of z^k in row i, column j of A, we compute

$$C_{kl,i+j} = \sum_{w,i,j} A_{kwi} B_{wlj} \tag{4.21}$$

for all k and l. This gives five loops. If A has degree n, then i is between 0 and n. If the first dimension is N, then k ranges between 0 and $N-1$. On the other hand, matrices are stored as in Fortran. Hence, if A is an $n \times m$ matrix, $\sum_{i=0}^{nm-1} A_i^2$ is the square of the norm of A (see table 4.11).

This example shows that we need up to five index variables. The variable n is reserved for the McMillan degree of the result. The variable m is the degree of G. The variable p is the dimension of the space (i.e. the dimension of D). For this reason, our index variables are called i, j, k, l and w. For some strange reason, they are ordered as kwlij. This piece of code puts the value on the property list of the symbol.

```
(putprop 'k 0 ':idx-val)
(putprop 'w 1 ':idx-val)
(putprop 'l 2 ':idx-val)
(putprop 'i 3 ':idx-val)
(putprop 'j 4 ':idx-val)
```

A loop that prints as

`for(i=0;i<n*m;i++)`

may be input as `i < 1a * 2b` (note the spaces) and is internally coded as a list of five quantities p_i. Field p_2 holds `<` or `<=`, and the field p_0 is 16 and 17 respectively (`<` is used when the upper bound is a dimension, and `<=` is used when the upper bound is a degree. This could be deduced automatically from the expression). Fields p_4, p_3 and p_1 encode the quantities i, n and m.

Quantities n, m that appear in the 'for' loop are dimensions and degrees. For each variable A, B, C of (4.21), we define n_a, m_a, d_a, n_b, m_b, etc., to be the first dimension, second dimension and degree. In the input, we shall use `1a`, `2a`, `3a` instead. This is because the vector `:param_a` contains at location i ($1 \le i \le 3$) this information. To each variable, we associate a value, formed of the index of the name, and the dimension.

```
(putprop '1a '(0 . 1) ':dblx-val)
(putprop '2a '(0 . 2) ':dblx-val)
(putprop '3a '(0 . 3) ':dblx-val)
(putprop '1b '(1 . 1) ':dblx-val)
(putprop '2b '(1 . 2) ':dblx-val)
(putprop '3b '(1 . 3) ':dblx-val)
(putprop '1c '(2 . 1) ':dblx-val)
(putprop '2c '(2 . 2) ':dblx-val)
(putprop '3c '(2 . 3) ':dblx-val)
```

This is the magic function, it splits the string into a list of symbols.

```
(defun :str-to-list (s)
  (read-from-string (catenate "(" s ")")))
```

A 'for' loop is given in the pattern list as `i<1a*2b` or `j<=3c`. The next function parses the given string. In the second case, the value of m is one, it is encoded by the integer 1, instead of a dotted pair.

```
(defun :fetch-for-pattern (s p)
  (let ((aux (:str-to-list s)))
    (vset p 4 (getprop (car aux) ':idx-val))
    (setq aux (cdr aux))
    (vset p 2 (car aux))
    (if (eq (car aux) '<=)
        (vset p 0 17)
        (vset p 0 16))
    (setq aux (cdr aux))
    (vset p 3 (getprop (car aux) ':dblx-val))
    (setq aux (cddr aux))
    (if aux
        (vset p 1 (getprop (car aux) ':dblx-val))
        (vset p 1 1))))
```

Consider again the product $C = AB$. Assume now that A is a zero-degree matrix, while B is a vector. Instead of

$$C_{kl,i+j} = \sum_{w,i,j} A_{kwi} B_{wlj}$$

we have now

$$C_{kj} = \sum_{w,j} A_{kw} B_{wj} \tag{4.22}$$

where A_{kw} is entry (k, w) of A, B_{wj} is the coefficient of z^j in entry w of B. In this example, A, B and C are objects with two indices. We get (4.22) from (4.21) by removing useless indices. Index i is useless because A has degree zero, index l is useless because the second dimension of B is one.

In general, a loop is useless if it is executed only once. Hence `for(i=0;i<=0;i++){body}` can be replaced by `i=0{body}`, and we can remove i from the body, if we wish so. What we have to do is find all useless loops and their associated index (because this index is also useless). In the case of `i<1a*2b` the loop is useless if $A_1 B_2$ is one; in the case of `j<=3c`, this is true if C_3 is zero. The next function checks whether a value n or m is zero or one. Here `val` is an integer, `str` the string that represents this number, and `i` the expression to test.

```
(defun :is-index01 (i val str)
  (if (eq i 1)
      (eq val 1)
      (equal (:dbl-index-val i) str)))

(dmd :dbl-index-val (x)
  `(vref (vref :all_param (car ,x)) (cdr ,x)))
```

If an index (i, j, etc.) appears in a useless loop the index is useless. On the other hand, if it appears in a useful loop, it is useful. We explain later what we do with the useful indices. The next function takes a 'for' loop, returns 5 if the loop is useless, and the index of the loop variable in case the loop is useful.

```
(defun :is-a-loop-useful (p)
    (if (= (vref p 0) 16) ; case <
        (if (and (:is-index01 (vref p 3) 1 "1")
                 (:is-index01 (vref p 1) 1 "1"))
            5
          (vref p 4))
    (if (= (vref p 0) 17)  ; case <=
        (if (or (:is-index01 (vref p 3) 0 "0")
                (:is-index01 (vref p 1) 0 "0"))
            5
          (vref p 4))
      5)))
```

Given a pattern name i, this returns the list of useful indices in the pattern. We look at every element of chunk C_0 (between C_0 and C_1, there is an element with no opcode, i.e. with null p_2). The return value is a vector, that holds true for a given slot if the variable is useless. Note that in the case $i = 6$ (division), there are two loops which are not in chunk C_0. For this reason we say that indices i and j are useful (in fact, they could be useless, but only if we divide a by b, where $\deg a < \deg b$ or $\deg b = 0$, both conditions are false in our case).

```
(defun :find-useless (i)
  (let ((p (vref :patterns i))
        (res (makevector 6 true))
        k)
    (while (vref (car p) 2)
      (vset res (:is-a-loop-useful (car p)) false)
      (setq p (cdr p)))
    (when (= i 6)       ; hack
          (vset res 3 false)
          (vset res 4 false))
    res))
```

Print the loops. Useless loops are not printed. There is some indentation, defined by the `indent` vector.

```
(defun :print-the-loops (p)
  (let (aux c j s (loop-number 0))
    (while (vref (car p) 2)
      (setq aux (car p))
      (setq p (cdr p))
      (unless (= 5 (:is-a-loop-useful aux))
            (prin1 (vref :indent loop-number))
            (setq loop-number (1+ loop-number))
            (setq c (vref :template (vref aux 4)))
            (prin1 "for(") (prin1 c) (prin1 "=0;") (prin1 c)
            (prin1 (vref aux 2))
            (setq j (vref aux 3))
            (setq s (:dbl-index-val j))
            (prin1 s)
            (setq j (vref aux 1))
            (when (consp j)
                  (setq s (:dbl-index-val j))
                  (prin1 "*") (prin1 s))
            (print ";" c "++)")))))
```

4.8 Differentiation

Consider again the simple operation $C \mathrel{+}= AB$. We have to generate

$$C_{kl,i+j} = \sum_{w,i,j} A_{kwi} B_{wlj}.$$

If A is an $n \times m$ matrix, we store A_{kw} in a vector at location $k + wn$, like in Fortran (but $0 \leq k < n$, $0 \leq w < m$ like in C), this avoids using 3 indirections for $A_{kw,i}$. The C code that refers to $A_{kw,i}$ is hence one of

```
a[k+w*n][i]
  a[k][i]
  a[w][i]
```

```
a[k+w*n]
  a[k]
  a[w]
  a[i]
```

(recall that some indices are useless; we assume that at least one of them is useful). Every such expression is obtained by instantiation of the string `X[k+w*Y][i]`, which is called an abbreviation. Table 4.13 gives, for each abbreviation, its name, for instance 'a', the name of the variable (for instance 'A'), the value of the first index, and the value of the second index. Each index is encrypted as either a list of four items, or as a single integer.

Table 4.13: Table of abbreviations

	name	index1	index2	enc1	enc2
a	a	$k + w * A_1$	i	(0 1 0 1)	4
at	a	$w + k * A_2$	$\deg(a) - i$	(0 2 1 0)	12
b	b	$w + l * B_1$	j	(1 1 1 2)	5
bt	b	$l + w * B_2$	$\deg(b) - j$	(1 2 2 1)	23
c	c	$k + l * A_1$	$i + j$	(0 1 0 2)	9
ct	c	$k + l * A_2$	$i + j$	(0 2 0 2)	9
a1	a	k	$\deg(a) - i$	1	12
a2	a	k	$\deg(c) - i + j$	1	35
b1	b		$\deg(b)$	0	20
b2	b		j	0	5
c2	c	k	$\deg(c) - i$	1	32
cp	c			0	0

The next function takes as argument a list of 4 elements, for instance `(0 1 0 1)` and evaluates it. The value is $k + w * A_1$, where A is the variable number 0, the subscript 1 on A_1 is the next element of the list, and variable k, w, have index 0 and 1 in the `template` string (`kwlij`). We return k if w is useless (or if A_1 is zero), w if k is useless (since this implies $A_1 = 1$), and nil, if both are useless.

```
(defun :instantiate-big (idx useless)
  (let (j1 j2 i1 i2 T)
    (setq j1 (car idx) j2 (cadr idx) i1 (caddr idx) i2 (cadddr idx))
    (setq T (vref (vref :all_param j1) j2) )
    (if (or (= (sref T 0) #/0) (vref useless i2))
        (if (vref useless i1)
            ()
          (vref :template i1))
      (if (vref useless i1)
          (vref :template i2)
        (catenate (vref :template i1) "+" (vref :template i2) "*" T)))))
```

The next function takes as argument an integer or a list. See table 4.13 for the meaning of the different codes. These codes are generated in the following way: the code for i, j and k is one more than the index in the 'kwlij' string. The codes for $-i$ and $-j$ are 2 and 3. The codes for the degrees of A, B and C are respectively 10, 20 and 30. The code of an expression is just the sum of the codes of the terms.

```
(defun :instantiate-idx1 (idx useless)
  (cond ((eq idx 0) ())
        ((or (eq idx 1) (eq idx 4) (eq idx 5))
         (setq idx (- idx 1))
         (if (vref useless idx)
             ()
             (vref :template idx)))
        ((or (eq idx 12) (eq idx 23) (eq idx 32))
         (let (a b res)
           (setq a (quomod idx 10) b #:ex:mod)
           (setq res (catenate (vref (vref :all_param (1- a)) 3)
                                (if (eq b 3) "-j" "-i")))
           (if (= (sref res 0) #/0)
               () res)))
        ((eq idx 9)
          (if (vref useless 3)
              (if (not (vref useless 4))
                 "j" ())
              (if (vref useless 4)
                  "i"
                  "i+j")))
        ((eq idx 35)
          (catenate (vref :param_c 3) "-i+j"))
        ((eq idx 20)
          (vref :param_b 3))
        (true (:instantiate-big idx useless))))
```

The next function does the same, but if the result is nil, it converts it to the empty string, otherwise, it adds square brackets around the expression.

```
(defun :instantiate-index (idx useless)
   (let ((res (:instantiate-idx1 idx useless)))
     (if res
         (catenate "[" res "]")
       "")))
```

Finally, we instantiate each of the 12 abbreviations, and put the result in a table. The table holds 19 elements, elements 12, 13 are the degrees of b and c. These quantities are needed by the code of the division. Element 14 holds 'tmp', used for the division. Remaining elements are prefixes that are added before the variables.

```
(defun :instantiate-subst (useless)
  (let ((res1 (makevector 19 ()))
        slot)
    (for (i 0 1 11)
         (setq slot (vref :abbrevs i))
         (vset res1 i  (catenate (vref (vref :all_param (vref slot 0)) 0)
                             (:instantiate-index (vref slot 1) useless)
                             (:instantiate-index (vref slot 2) useless))))
    (vset res1 12 (vref :param_b 3))
    (vset res1 13 (vref :param_c 3))
    (vset res1 14 "tmp")
    (vset res1 15 "")
    (vset res1 16 "delta_")
    (vset res1 17 "d")
    (vset res1 18 "delta_d")
    res1))
```

Now, the pattern lists holds quantities like `dc delta_b delta_dat`. The next piece of code associates to each variable that has this form three numbers: the first is 2 in general, it is 1 in case the variable is a derivative of a, and 1 if it is a derivative of b. The second quantity is the index in the abbreviation table. The last one is the prefix.

```
(let (L1 V1 l1 v1 L2 L a b i)
     (setq L1 '("" "delta_" "d" "delta_d"))
     (setq V1 '(15  16      17   18))
     (setq L2 '(a 0 at 0 b 1 bt 1 c 2  ct 2 a1 0 a2 0 b1 1
                b2 1 c2 2 cp 2 () 0 () 0 tmp 2))
     (while (consp L1)
        (setq l1 (car L1) v1 (car V1) L1 (cdr L1) V1 (cdr V1))
        (setq L L2 i 0)
        (while (consp L)
            (setq a (car L) b (cadr L) L (cddr L))
            (if (= v1 15) (setq b 2))
            (if a
                (putprop (concat l1 a) (mcons b v1 i) ':var-val))
            (setq i (+ i 1)))))
```

This reads now a prefix and a variable. If no variable is found, we hope it is OK, and act as if we found 'No variable found'. This will generate a syntax error when compiling.

```
(defun :split-pat-aux (str p loc)
  (let ((a (getprop str ':var-val)))
       (if a
           (progn (vset p loc (cdr a)) (car a))
           (vset p loc "No variable found")
           2)))
```

The variable `mask` is 1 if a derivative of the variable 'a' appears in the expression, 2 if a derivative of the variable 'b' appears, 0 if none, and 3 if both.

We shall ignore every operation for which the logical 'and' between `mask` and `:constant-flag` is not zero. It is to be noticed that the mask is zero for all special operations ('for' loops, braces, etc.) and for all operations that do not have the form $x \mathrel{+}= y$, where y is a product of two terms. If the mask is 1, one of the factors is a derivative of a, if the mask is 2, one of the factors is a derivative of b, and if the mask is 3, one factor is a derivative of a and the second a derivative of b.

The expression we parse is a function call with 2 or 3 arguments. The next function updates the mask after having examined one argument. The index returned by the previous function is the argument b, by construction it is 0 (resp. 1) if the variable 'a' (resp. 'b') appears with a prefix in the expression, i.e. if the derivative of the variable appears in the expression. It is 2 if no such derivative appears.

```
(defun :add-to-mask (mask b)
  (cond ((= b 2) mask)
        ((= b 0) (if (or (= mask 0) (= mask 2)) (+ mask 1) mask))
        ((= b 1) (if (or (= mask 0) (= mask 1)) (+ mask 2) mask))))
```

This function takes as argument a string L, and a vector p, splits L into 3 arguments, puts the arguments into p_2, p_3 and p_4. It computes also the mask, and puts it into p_1. The last field p_0 of p is the operator, it is read by another function.

```
(setq open-p #/( close-p #/)) ; for emacs
(defun :split-pattern (L p)
  (let ( (mask 0))
    (setq i (:split-pat-aux (car L) p  2))
    (setq mask (:add-to-mask mask i))
    (setq i (:split-pat-aux (cadr L) p  3))
    (setq mask (:add-to-mask mask i))
    (setq i (:split-pat-aux (caddr L) p  4))
    (setq mask (:add-to-mask mask i))
    (vset p 1 mask)))
```

The code of the division is a bit special: there are ‘for’ loops inside the code, for instance: `for(j=0;j<, degb, ;j++)`. The `find-useless` function knows about these loops: indices i and j are always useful in the case of division, even if the loop in which they appear is not in the C_0 chunk. The number of times the loop is executed has abbreviation number 12 and 13. Recall that `instantiate-subst` instantiates these quantities to the degree of B and C. The special ‘for’ loops are parsed via the next function. It fills the fields p_2, p_3 and p_4 (the field p_1 is left null).

```
(defun :special-pattern (L p)
  (vset p 2 (car L))
  (vset p 4 (caddr L))
  (cond ((eq (cadr L) 'degb) (vset p 3 12))
        ((eq (cadr L) 'degc) (vset p 3 13))
        (true (error "diff" "Bad pattern" L))))
```

Finally, a pattern line is converted using the following function. The argument may be nil (end-of-chunk marker), a string (uninterpreted), or a list. The first element of the list, if it is a symbol, is an operator, which is converted to some numbers, using `convert-to-nb`, and put in p_0, otherwise, there is no operator. After that, we should have an argument list, (a string that starts with an left parenthesis) or a special pattern, namely 3 strings.

```
(defun :fetch-normal-pattern (L p)
  (if (eq L ())
      (vset p 2 ())
    (if (consp L)
        (:special-pattern L p)
      (if (= (strlen L) 1)
          (vset p 2 L)
          (setq L (:str-to-list L))
          (vset p 0 (getprop (car L) ':op-val))
          (:split-pattern (cdr L) p)))))
```

This converts an operator, defined by the first two letters into a list of 3 items (see table 4.3).

```
(putprop '++ '(10 10)':op-val)
(putprop '+ '(0 1 15) ':op-val)
(putprop '-- '(11 11) ':op-val)
(putprop '- '(1 0 15) ':op-val)
(putprop '/  '(4 4 15) ':op-val)
(putprop '=  '(6 6) ':op-val)
(putprop '1* '(7 12 16) ':op-val)
(putprop '2* '(8 13 17) ':op-val)
(putprop '3* '(9 14 18) ':op-val)
(putprop '_+ '(2 3 15) ':op-val)
(putprop '_- '(3 2 15) ':op-val)
(putprop '_/ '(5 5 15) ':op-val)
```

This is the code of the differentiator: it just replaces each abbreviation by its value. Recall that an abbreviation is a number, or a cons of two numbers, and we take the value from the `subst` vector.

```
(defun :instantiate-string (L subst)
  (cond ((stringp L) L)
        ((integerp L) (vref subst L))
        ((not L) "")
        ((consp L) (catenate (vref subst (car L)) (vref subst (cdr L))))
        (true "Bad string in instantiate")))
```

It’s now time to print the result of the differentiator. We do not show the piece of code that prints debugging information. The ‘funcall’ that’s here may print the name of the operator followed by the arguments, or expand it.

```
(defun :print-an-instruction (p subst prec)
  (when (= 0 (land (vref p 1) :constant-flag))
        (let ((s1 (vref p 2))(s2 (vref p 3))(s3 (vref p 4))
              args res db db-op need-debug (slot (vref p 0)))
          (if :neg-a (setq op (cadr slot)) (setq op (car slot)))
          (setq s1 (:instantiate-string s1 subst))
          (setq s2 (:instantiate-string s2 subst))
          (setq s3 (:instantiate-string s3 subst))
          (prin1 "    ")
          (when op
                (if (= 15 op)
                    (setq op ())
                  (setq op (vref  :the-operators op))))
          (if (not op)
              (print s1 s2 s3)
            (setq args (list s1 s2 s3))
            (funcall op prec args)))))
```

Prints the whole code. For each mode i, each precision in the list of precisions, we print a section, that is formed of chunks C_0 and C_i.

```
(defun :print-code (i title name)
  (let (useless subst p p1 p2)
    (setq useless (:find-useless i))
    (setq subst (:instantiate-subst useless))
    (setq p (vref :patterns i))
    (setq p1 p)
    (while (vref (car p1) 2)
      (setq p1 (cdr p1)))
    (for (prec 0 1 3)
         (setq p2 p1)
         (for (mode 0 1 3)
              (:decl-sec title name)
              (:print-the-loops p)
              (setq p2 (cdr p2))
              (while (vref (car p2) 2)
                (:print-an-instruction (car p2) subst prec)
                (setq p2 (cdr p2)))))))
```

4.9 Merging code

The next function takes as argument a list, with at least 3 elements. It calls the differentiate function with the first 4 arguments, and returns the fourth. This is the section title, one is invented if none is given. If more than four arguments are given, remaining arguments are printed as `WEB` comments.

```
(defun :call-diff (y)
    (let (z (temp (cdddr y)))
       (if (not temp)
           (setq z (catenate "Untitled section " (gensym) "."))
           (setq z (car temp)))
       (setq temp (cdr temp))
       (when temp
             (print "@")
             (show-usage)
             (while temp
                    (print (car temp))
                    (setq temp (cdr temp))))
       (differentiate (car y) (cadr y) (caddr y) z)
       z))
```

The next function takes as argument a list like `(if test y)`. If y is a list, it calls `call-diff` on it. If this gives Y, we return the list with three items, the first is `iftest{`, the second is Y, the last is `}`. Note that the first argument must be the Lisp symbol `if`, and the second argument (called 'test' in the example), should be a valid C test, hence a parenthesised expression. We allow also the case of two arguments, `(ifsomething y)`, where the first argument is a string that starts with the two letters 'i' and 'f'. In the case where y is not a list, there is no call to `call-diff`, and Y is replaced by y.

Finally, the user may give y alone. The function returns y if it is not a list, the result Y of `call-diff` otherwise.

```
(defun :handle-if (x)
  (if (not (consp x))
      x
    (let ((test ()) y)
      (cond ((eq (car x) 'if)
             (setq test (catenate "if" (cadr x) open-brace) y (caddr x)))
            ((and (not (cddr x)) (substring-eq (car x) 0 "if" 0 2))
             (setq test (catenate (car x) open-brace) y (cadr x)))
            (true (setq y x)))
      (if (consp y) (setq y (:call-diff y)))
      (if test (list test y close-brace) y))))
```

This calls the previous function on each elements of the argument list. It calls `merge-code` on the result.

```
(defun differentiate1 (L)
  (let (res x)
    (newl res (car L))
    (setq L (cdr L))
    (while (consp L)
      (setq x (car L) L (cdr L))
      (newl res (:handle-if x)))
    (merge-code (nreverse res))))
```

For each mode, each precision in the precision list, the next function prints a section whose name is the first element of L. The list L is printed in reverse order in reverse mode. If an element L_i of L is a list, it has 3 elements, a string, a section title, and another string. Otherwise L_i is a section title. Typically, each L_i is the result of a call to `handle-if` (see example below). Each section title is printed via `use-sec`, that appends the mode and precision to it.

```
(defun merge-code (L)
  (for (prec 0 1 3)
    (for (mode 0 1 3)
      (let ((aux L) x)
         (decl-sec (car aux))
         (setq aux (cdr aux))
         (if (or (= mode 1) (= mode 3))
             (setq aux (reverse aux)))
         (while aux
           (setq x (car aux) aux (cdr aux))
           (if (consp x)
               (progn (print (car x)) (use-sec (cadr x)) (print (caddr x)))
             (use-sec x)))))))

```

4.9.1 Example

We consider the case of where a is `(y p 1 0)`, b is `(y p 1 0 transpose)`, and c is z.

If we run the `differentiate` function, we get `(z p p 0)` as a result, and HYPERION prints, among other things:

```
@ Patterns for $T_b$
@<Title, delta diff, complex case@>=
for(k=0;k<p;k++)
   for(l=0;l<p;l++)
    {
    c_add_mult_conj_mac(dz[k+l*p],delta_y[k],delta_dy[l]);
    c_add_mult_conj_mac(delta_dz[k+l*p],y[k],delta_dy[l]);
    c_add_mult_mac(delta_y[l],dz[k+l*p],delta_dy[k]);
    c_add_mult_mac(y[l],delta_dz[k+l*p],delta_dy[k]);
    }
```

In the case where we want macros to be expanded, the start of the previous code may be the following (complex multiplication may use Karatsuba or the usual algorithm). We show here only the first multiplication.

```
@<Title, delta diff, complex case@>=
for(k=0;k<p;k++)
   for(l=0;l<p;l++)
    {
    {
double Ctemp1,Ctemp2;
Ctemp1 = dz[k+l*p].r*delta_y[k].r;
Ctemp2 = -dz[k+l*p].i*delta_y[k].i;
delta_dy[l].i += (dz[k+l*p].r-dz[k+l*p].i)*(delta_y[k].r+delta_y[k].i)
- Ctemp1 - Ctemp2;
delta_dy[l].r += Ctemp1 - Ctemp2;
}
```

Example of `merge-code`. Consider the following input to HYPERION:

```
(differentiate1 '(
   "Prepare $X$ and $Y$"
   "Complete code of B"
   "Code of |yD|"
   "Code of $E$"
   "Code of N"
   "Code of $X$, 1"
   "Code of $X$, 2"
   "Code of $X$, 3"
   ("if(p==2)" "Complete code of $Z$")
   (if "(p>2)" "Code of $Y$")))
```

In this piece of code, `handle-if` does not call `call-diff`. We could change the `handle-if` function in such a way the the last two lines could be merged in a single one, namely `("if(p==2)" "Complete code of $Z$" "elseif(p>2)" "Code of $Y$")`, but we think this is not really needed (if the compiler is smart enough, it will compare p against 2 only once, but even if the test is done twice, this will not really slow down the algorithm).

One of the 16 sections printed is:

```
@ @<Prepare $X$ and $Y$, direct, complex case@>=
@<Complete code of B, direct, complex case@>@;
@<Code of |yD|, direct, complex case@>@;
@<Code of $E$, direct, complex case@>@;
@<Code of N, direct, complex case@>@;
@<Code of $X$, 1, direct, complex case@>@;
@<Code of $X$, 2, direct, complex case@>@;
@<Code of $X$, 3, direct, complex case@>@;
if(p==2){
@<Complete code of $Z$, direct, complex case@>@;
}
if(p>2){
@<Code of $Y$, direct, complex case@>@;
}
```

And this is another section.

```
@ @<Prepare $X$ and $Y$, delta diff, complex case@>=
if(p>2){
@<Code of $Y$, delta diff, complex case@>@;
}
if(p==2){
@<Complete code of $Z$, delta diff, complex case@>@;
}
@<Code of $X$, 3, delta diff, complex case@>@;
@<Code of $X$, 2, delta diff, complex case@>@;
@<Code of $X$, 1, delta diff, complex case@>@;
@<Code of N, delta diff, complex case@>@;
@<Code of $E$, delta diff, complex case@>@;
@<Code of |yD|, delta diff, complex case@>@;
@<Complete code of B, delta diff, complex case@>@;
```

4.10 Operators

In this section, we implement some functions that print something like $a \mathrel{+}= bc$. Recall that we have a C macro `double_add_mac`, that is not shown here, and that could be used. We find it more readable to just print `a += b*c`. On the other hand, in the complex case, we have a macro `c_add_mult_mac`, and using the function below makes the code unreadable. In fact, the main reason for doing so, is that we can switch between the standard macro, or the macro that uses Karatsuba, without changing the header files (recall the funcall in `print-an-instruction`).

The next function takes 2 arguments, a string s, and a list (x, y, z, t) or five arguments, s, x, y, z and t. It prints the string s, but 'X', 'Y', 'Z' and 'T' are replaced by x, y, z or t. Moreover, '#' is replaced by a newline character.

```
(defun :print-subst (str . l)
  (when (and (consp (car l)) (not (cdr l))) (setq l (car l)))
  (let ((X (car l)) (Y (cadr l)) (Z (caddr l)) (T (cadddr l)))
    (let ((i (strlen str)) (j 0) c)
      (while (< j i)
        (setq c (sref str j) j (+ j 1))
        (cond ((eq c #/X) (prin1 X))
              ((eq c #/Y) (prin1 Y))
              ((eq c #/Z) (prin1 Z))
              ((eq c #/T) (prin1 T))
              ((eq c #/#) (terpri))
              (true (princn c))))
      (terpri))))
```

This is the table of operators (cf table 4.4 on page 114).

We give here only the code of one function per slot in the table. (For instance, there is a function equivalent to `:op-add-mul-K` that does not use Karatsuba, and another one that calls a C macro instead of expanding the call). Switching between equivalent function is trivially done by changing the name in the next table.

```
(setq :the-operators #[ :op-add-mul-K :op-sub-mul-K :op-add-mul-conj-K
    :op-sub-mul-conj-K :op-div  :op-div-conj :op-set  :op-plus-norm
    :op-2times :op-2timesr :op-add  :op-sub :op-minus-norm
    :op-2times-neg :op-2timesr-neg])
```

This is used for `C++`. Instead of `f(a,b,c)`, we print `c.f(a,b)`.

```
(defun :pc++2 (op argl)
  (:print-subst "Z.X(Y);" (cons op args)))
(defun :pc++3 (op argl)
  (:print-subst "T.X(Y,Z);" (cons op args)))
(defun :pc++4 (op argl)
  (let (((x y z t) argl))
    (print t z "." op "(" x "," t y ");")))
```

This function computes $z = z + xy$. As will be the case for the following functions, the first argument `prec` is the precision, and the second argument is the list `(x y z)`. Some operators take only two arguments. We call them x and y. Note that `print-subst` uses X, Y and Z.

```
(defun :op-add-mul-K (prec args)
  (cond ((eq prec 0) (:print-subst "Z += X*Y;" args))
        ((eq prec 2)(:pc++3 "add_mul_K" args))
        (true (:pc++3 "add_mul" args))))
```

This function computes $z = z - xy$.

```
(defun :op-sub-mul-K (prec args)
  (cond  ((eq prec 0) (:print-subst "Z -= X*Y;" args))
         ((eq prec 2)(:pc++3 "sub_mul_K" args))
         (true (:pc++3 "sub_mul" args))))
```

This function computes $z = z + \overline{x}y$.

```
(defun :op-add-mul-conj-K (prec args)
  (cond ((eq prec 0)  (:print-subst "Z += X*Y;" args))
         ((eq prec 1) (:pc++3 "add_mul" args))
         ((eq prec 2) (:pc++3 "add_mul_conj_K" args))))
         ((eq prec 3) (:pc++3 "add_mul_conj" args))))
```

This function computes $z = z - \overline{x}y$.

```
(defun :op-sub-mul-conj-K (prec args)
  (cond ((eq prec 0)  (:print-subst "Z -= X*Y;" args))
         ((eq prec 1) (:pc++3 "sub_mul" args))
         ((eq prec 2)(:pc++3 "sub_mul_conj_K" args))
         ((eq prec 3) (:pc++3 "sub_mul_conj" args))))
```

This function computes $z = x/y$.

```
(defun :op-div (prec args)
  (cond ((eq prec 0) (:print-subst "Z = X/Y;" args))
         (true (:pc++3 "div" args))))
```

This function computes $z = x/\overline{y}$.

```
(defun :op-div-conj (prec args)
  (cond ((eq prec 0) (:print-subst "Z = X/Y;" args))
         ((eq prec 1) (:pc++3 "div" args))
         (true (:pc++3 "div_conj" args))))
```

This function computes $y = x$.

```
(defun :op-set (prec args)
  (:print-subst "Y = X;" args))
```

This function computes $y = y + |x|^2$.

```
(defun :op-plus-norm (prec args)
  (cond ((eq prec 0) (:print-subst "Y += X*X;" args))
         ((eq prec 2)
          (:print-subst "{#Complex cmt = X; Y += cmt.r*cmt.r+cmt.i*cmt.i;#}" args))
         (true (:pc++2 "square" args))))
```

This function computes $y = y - |x|^2$.

```
(defun :op-minus-norm (prec args)
  (cond ((eq prec 0) (:print-subst "Y -= X*X;" args))
         ((eq prec 2)
          (:print-subst "{#Complex cmt = X; Y -= cmt.r*cmt.r+cmt.i*cmt.i;#}" args))
         (true (:pc++2 "square_neg" args))))
```

The function computes $z = z + 2\Re(\overline{x}y)$. Note that z is real.

```
(defun :op-2times (prec args)
   (cond ((eq prec 0) (:print-subst "Z += 2*X*Y;" args))
         ((eq prec 2) (:print-subst "Z += 2*(X.r*Y.r +X.i*Y.i);" args))
         (true (:pc++3 "two_times" args))))
```

The function computes $z = z - 2\Re(\overline{x}y)$. Note that z is real.

```
(defun :op-2times-neg (prec args)
   (cond ((eq prec 0) (:print-subst "Z -= 2*X*Y;" args))
         ((eq prec 2) (:print-subst "Z -= 2*(X.r*Y.r +X.i*Y.i);" args))
         (true (:pc++3 "two_times_neg" args))))
```

This function computes $z = z + 2xy$. Here y is real, z and x may be complex.

```
(defun :op-2timesr (prec args)
  (cond ((eq prec 0) (:print-subst "Z += 2*X*Y;" args))
         ((eq prec 1)  (:pc++3 "two_times" args))
         (true (:pc++3 "two_times_r" args))))
```

This function computes $z = z - 2xy$. Here y is real, z and x may be complex.

```
(defun :op-2timesr-neg (prec args)
  (cond ((eq prec 0) (:print-subst "Z -= 2*X*Y;" args))
         ((eq prec 1) (:pc++3 "two_times_neg" args))
         (true (:pc++3 "two_times_r_neg" args))))
```

This function computes $z = x + y$.

```
(defun :op-add (prec args)
   (cond ((eq prec 0) (:print-subst "Z = X+Y;" args))
         (true (:pc++3 "add" args))))
```

This function computes $z = x - y$.

```
(defun :op-sub (prec args)
   (cond ((eq prec 0) (:print-subst "Z = X-Y;" args))
         (true (:pc++3 "sub" args))))
```

This prints the code associated to $b \mathrel{+}= |a|^2$.

```
(defun :print-add-norm (a b prec)
  (let ((s1 "Y += XT*XT;") (s2 "Y.add_mul(XT,XT);"))
    (cond ((= prec 0) (:print-subst s1 a b () ""))
          ((= prec 1) (:print-subst s2 a b () ""))
          ((= prec 2)
           (:print-subst s1 a b () ".r")
           (:print-subst s1 a b () ".i"))
          ((= prec 3)
           (:print-subst s2 a b () ".r")
           (:print-subst s2 a b () ".i")))))
```

This prints the code associated to $b = |a|^2$, given real and imaginary part of a.

```
(defun :print-add-norm1 (a1 a2 b prec)
  (cond ((= prec 0)
         (:print-subst "Z = X*X;" a1 a2 b))
        ((= prec 2)
         (:print-subst "Z = X*X+Y*Y;" a1 a2 b))
        ((= prec 1)
         (:print-subst "Z.mul(X,X);" a1 a2 b))
        ((= prec 3)
         (:print-subst "Z.mul(X,X);#Z.add_mul(Y,Y);" a1 a2 b))))
```

This prints the code associated to $b = -a$.

```
(defun :print-neg (a b prec)
  (cond ((= prec 0)
         (:print-subst "Y = -X;" a b))
        (true
         (:print-subst "Y.neg(X);" a b))))
```

This prints the code associated to $x = x - y$.

```
(defun :print-neg-equal (x y p)
  (print p x " -= " p y ";"))
```

This function sets a variable to zero.

```
(defun :call_clear_op (prec)
  (if (eq prec 0)
      (print " = 0;")
    (print ".kill();")))
```

4.11 Other functions

From now on, the code is specific to our application. We first define two functions `:real` and `:imag` that return the real and imaginary part of a complex number. Since a complex number x is represented as a structure with two fields, this is $x.r$ and $x.i$.

```
(dmd :spec-imag (prec) `(= ,prec 2))
(defun :imag (x prec)
     (catenate x ".i"))
(defun :real (x prec)
   (if (:prec-real prec) x
      (catenate x ".r")))
```

This returns true if precision is real or not. We have a function that returns the extension to use if we must take the real part of a complex number.

```
(dmd :prec-real (prec) `(< ,prec 2))
(dmd :prec-imag (prec) `(>= ,prec 2))
(defun :real-ext (prec) (if (:prec-real prec) "" ".r"))
```

What we have to differentiate is a function ψ that depends on Q, which is a function of n vectors y_i of size p. Note that y_i and Q can be complex. Thus the derivative is a set of n vectors y'_i of size p. Since our function is used by a generic optimiser, its input and output must be a single real vector Y and Y'. Thus we have to copy Y into y_i at the beginning, and y'_i to Y' at the end.

We shall explain later exactly how and where y_i and Y are stored. The next function converts a single element of Y into a single element of y_i (a single element of Y can be formed of two real numbers). It is assumed that s has the form `x[k]` and the destination d has the form `y[j]`. This piece of code modifies the index k, but not the index j. The argument `aux` must have the form `k++`.

```
(defun :gen-copy-cmplx2real (s d aux prec)
  (if (:prec-real prec)
      (:print-subst "Y = X;Z" s d aux)
      (:print-subst "Y = X.r;Z#Y = X.i;Z" s d aux)))
```

This does the copy the other way.

```
(defun :gen-copy-real2cmplx (s d aux prec)
  (if (:prec-real prec)
      (:print-subst "Y = X;Z" s d aux)
      (:print-subst "Y.r = X;Z#Y.i = X;Z" s d aux)))
```

This piece of code declares a real variable (even in the complex case) and initialises the variable to 1. Note that this must be after every other declaration and before any other executable C code (this restriction does not apply to `C++`).

```
(defun :initialise-psi (var prec)
  (if (or (= prec 0) (= prec 2))
      (print "  double " var " = 1;")
    (print "   SLD " var "; " var ".get_one();")))
```

Same code, but the variable is set to one for a different reason (in the previous code, we have $\|G\| = 1$, here we have $d\psi = 1$).

```
(defun :clear-a-div (var prec)
  (:initialise-psi var prec))
```

Sets a variable to one. If the mode is complex, the variable is still real (this can be used for instance to compute $1 - |\omega|^2$).

```
(defun :print-set-one-only (prec var)
  (if (or (= prec 1) (= prec 3))
      (print var ".get_one();")
      (print var " = 1;")))
```

Set the real part of the variable to 1, where the variable is defined by the concatenation of `var1`, `extp` and `var2`. This will be used for instance to initialise Q_0 to the identity matrix.

```
(defun set-cr-one (var1 var2 prec extp)
   (cond ((= prec 0) (print var1 extp var2 " = 1;"))
         ((= prec 1) (print var1 extp var2 ".get_one();"))
         ((= prec 2) (print  var1 extp var2 ".r = 1;"))
         ((= prec 3) (print  var1 extp var2 ".get_one();"))))
```

4.11.1 Main function

Recall the example given at the start of this chapter.

```
@u double f (double x)
{
   double y,z;
   @<First part of f@>
   @<Second part of f@>
   return y;
}
```

We explained how to construct the sections 'First part of f' and 'Second part of f', and now we have to explain everything else. For simplicity, the functions we construct are members of a class T, which is a huge data structure that contains everything (including the result of the function). There are objects in T which are not needed here (for instance, we always assume $\|G\| = 1$, by dividing G by its norm. The norm of G has to be kept somewhere, because we need it when computing the numerator $L_F(Q)$). Whenever a field of T is first used, we shall explain where it is stored.

The first implementation choice we do concerns the naming of the external functions. The main function `matrix_psi_fct` computes ψ. It just calls another function, that is precision dependent. Thus we have four medium size functions, instead of one big function (compilers prefer small functions, they are easier to optimise). The quantity `prec_type` is an integer that gives the current precision of T. In the Lisp code, the precision is an integer between 0 and 3. In the C code, it is an integer, for instance `ll_type_scpolynom` (the 'sc' part is variable). The next function prints a `case` statement. It is followed by some trivial functions.

```
(defun :print-switch-case (prec)
  (print "case ll_type_" (vref prefix_types prec) "polynom:"))

(defun :print-open-brace () (print "{"))
(defun :print-close-brace () (print "}"))
(defun :print-double-close-brace () (print "}}"))
```

In the case of ψ and ψ', we just call the function that is in T. In the case of ψ'', the function in T compute the second derivative in one direction. Thus we have to define two functions : one that takes -1 as argument, and computes the Hessian in one direction (found in T), and puts the result somewhere in T, and a function without arguments, that computes the derivative in every direction. The variable `mode` is 3 or 4. Note that n and p are represented by `a_n` and `a_p` in the data structure T.

```
(defun print_a_switch_call1 (title mode)
  (let ((args "(I);"))
    (show-usage)
    (print "@")
    (print "@u void " title "()")
    (:print-open-brace)
    (if (= mode 4)
        (setq args "(-1);")
        (print "int I, N=a_n*a_p;")
        (print "if(real_flag==0) N = 2*N;")
        (print "for(I=0;I<N;I++)"))
    (print "matrix_psi_delta_d" args)
    (:print-close-brace)))
```

Equations (4.17) or (4.18) have the form $Y = g_1(Q)$, $\psi = g_2(Y)$. In writing this, we just mean that we take Q (which is a pair (D, q)), compute some quantities, and use these to compute ψ. In the same fashion, equations (4.4) have the form $X = f_1(A)$ and $B = f_2(X)$. Now the code of ψ can be written as

```
$B = I$
for $i = 0$ to $n-1$ do
    $A = B$
    $X = f_1(A)$, $B = f_2(X)$
$Q = B$
$Y = g_1(Q)$, $\psi = g_2(Y)$
store $\psi$ in $T$.
```

or more generally as

```
Get variables, 1
for $i = 0$ to $n-1$ do
    Get variables, 2
    The main loop
Get variables, 3
Remaining of the code
Code before returning.
```

The parts '`The main loop`' and '`Remaining of the code`' are obtained from the differentiator. The other parts are explained below.

The next function prints the code of ψ, ψ' and ψ''. There is one C function per precision, hence 12 functions at all.

```
(defun print-fct-00 ()
 (for (prec 0 1 3)
     (let (
        (extp (vref prec_types prec))
        (casep (vref precision_names prec)))
```

This is the first part of the Lisp function. It defines the function ψ as explained above. We shall see that '`Get variables, 3`' is independent of the precision. We add a dot at the end of the section name so as to make `web` happy. Note that this piece of code is a member function of a sub-object of T, whose type gives the precision.

```
       (show-usage)
       (print "@")
       (print "@u double data2p" extp "::matrix_psi_fct()")
       (:print-open-brace)
       (print "@<Get variables, 1" case-direct casep at-semi)
       (print "@<Special initialisation, direct@>@;")
       (print "   for(it=0;it<n;it++){")
       (print "   @<Get variables, 2" case-direct casep at-semi)
       (print "   @<The main loop" case-direct casep at-semi)
       (print "    }")
       (print "@<Get variables, 3" case-direct "." at-semi)
       (print "   @<Remaining of the code" case-direct casep at-semi)
       (print "   @<Code before returning" case-direct casep at-semi)
       (:print-close-brace)
       (terpri)
```

This is the second part of the Lisp function. It defines the function ψ' which computes the derivative of ψ in reverse mode. The order of the sections is reversed, as is the main loop. The section '`Code before returning`' stores the derivative somewhere. Recall that, in reverse mode, since ψ is a function of y_i, the derivative will be in y_i', we copy this into a single vector.

```
       (show-usage)
       (print "@ @u  void data2p" extp "::matrix_psi_grad()")
       (:print-open-brace)
       (print "@<Get variables, 1" case-diff casep at-semi)
       (print "@<Get variables, 3" case-diff "." at-semi)
       (print "@<Remaining of the code" case-diff casep at-semi)
       (print "   for(it=n-1;it>=0;it--){")
       (print "   @<Get variables, 2" case-diff casep at-semi)
       (print "   @<The main loop" case-diff casep at-semi)
       (print "    }")
       (print "@<Code before returning" case-diff casep at-semi)
       (:print-close-brace)
       (terpri)
```

Last part. We compute ψ'' by differentiating, in direct mode, the code of ψ and ψ', applied to a vector v, which depends on the argument I. If I is not -1, the section 'Code before returning' copies column I of the Hessian (which is somewhere in the data structure) into the desired place.

```
       (show-usage)
       (print "@ @u void data2p" extp "::matrix_psi_delta_d(int I)")
       (:print-open-brace)
       (print "@<Get variables, 1" case-delta casep at-semi)
       (print "@<Special initialisation, delta@>@;")
       (print "   for(it=0;it<n;it++){")
       (print "   @<Get variables, 2" case-delta casep at-semi)
       (print "   @<The main loop" case-delta casep at-semi)
       (:print-close-brace)
       (print "@<Get variables, 3" case-delta "." at-semi)
       (print "@<Remaining of the code" case-delta casep at-semi)
       (print "@<Get variables, 3" case-deltadiff "." at-semi)
       (print "@<Get variables, 1" case-deltadiff casep at-semi)
       (print "@<Remaining of the code" case-deltadiff casep at-semi)
       (print "   for(it=n-1;it>=0;it--){")
       (print "@<Get variables, 2" case-deltadiff casep at-semi)
       (print "   @<The main loop" case-deltadiff casep at-semi)
       (:print-close-brace)
       (print "@<Code before returning" case-deltadiff casep at-semi)
       (:print-close-brace)
)))
```

Let's go back to our program scheme. Instead of $X = f_1(A)$, we shall write $f_1(A, X)$. This means that f_1 takes as arguments two pointers. It modifies the value pointed to by the second argument. The notation $A \Rightarrow B$ means that A points to the memory location B. In the case $n = 3$, our program becomes

$A \Rightarrow Q_{-1}$, $B \Rightarrow Q_0$, $X \Rightarrow X_0$
$B = 0$, $X = 0$
$f_1(A, X)$, $f_2(X, B)$
$A \Rightarrow Q_0$, $B \Rightarrow Q_1$, $X \Rightarrow X_1$
$B = 0$, $X = 0$
$f_1(A, X)$, $f_2(X, B)$
$A \Rightarrow Q_1$, $B \Rightarrow Q_2$, $X \Rightarrow X_2$
$B = 0$, $X = 0$
$f_1(A, X)$, $f_2(X, B)$
$Q \Rightarrow Q_2$, $Y \Rightarrow Y_0$, $\psi \Rightarrow \psi_0$
$Y = 0$, $\psi = 0$
$g_1(Q, Y)$, $g_2(Y, \psi)$

If for instance $X = f_1(A)$ has the form $X = A_1^2 + A_2^2$, the code $f_1(A, X)$ is now $X \mathrel{+}= A_1^2$, $X \mathrel{+}= A_2^2$, since it is preceded by the assignment $X = 0$. If we differentiate this in reverse mode, we get $dA_1 \mathrel{+}= 2A_1 dX$, $dA_2 \mathrel{+}= 2A_2 dX$, $dX = 0$. In fact, we replace this by $dA_1 = 0$, $dA_2 = 0$, $dA_1 \mathrel{+}= 2A_1 dX$, $dA_2 \mathrel{+}= 2A_2 dX$ (there is no need to reset a variable to zero at the end of the code, but we have to clear it at the start. The only variable for which dX should not be zero initially is $d\psi$ and δy. The case of $d\psi$ is handled by `:clear-a-div`, the case of δy is a bit more complicated, and will be explained later).

Essentially, this means that the code of the derivative has the same structure than the code of the function: we have first to define $k+1$ pointers, and clear k of them, before computing something really useful. We make now the difference between 'local' and 'global' variables. A better name would perhaps be 'inner' and 'outer', because, as explained, everything is in the data structure T. A local variable is one that depends on i (it's `it` in the C code). They are used in the main loop. Global variables are used in the remaining of the code. Hence, with the notations above (see page 139), X is local, and Y is global. Variables A, and Q have a special status (they are not cleared). Variable i, together with all other indices have also a special status (for instance, there is no need to differentiate them).

In the case of a local variable, we have an assignment of the form $Z \Rightarrow X_i$. This means that, in T there is a table X, of size n. There is also a table Q of size n. In fact, we have a big table of size n, that holds X and Q, it is called `T->temp`. We have another table for the derivative. It is called `T->dtemp`. Note that Q_{-1} is not in this table. For the case of global variables, we have an assignment $Y \Rightarrow Y_0$. There is no table here, we used the index zero, because $Y \Rightarrow Y$ is a bit strange (the C code is in fact `Y=T->Y`, and this is unambiguous). We changed this: In C++, there is no variable T, so that we add a prefix before every variable.

If we differentiate, in direct mode, the start of the code above, we get

$$A \Rightarrow Q_{-1}, \quad B \Rightarrow Q_0, \quad X \Rightarrow X_0$$

$$\delta A \Rightarrow \delta Q_{-1}, \quad \delta B \Rightarrow \delta Q_0, \quad \delta X \Rightarrow \delta X_0$$

$$\delta B = 0, \quad \delta X = 0.$$

In other words, we have to declare X and δX, but to kill only δX. For this reason, the code is split into two parts: 'Get local variables' and 'Kill local variables'.

```
(defun :get-and-kill-vars (mode extp sw)
  (let (L x name)
    (setq L (prev-modes mode))
    (unless sw ; special hack for globals / hessian.
            (if (= mode 2)
                (setq L '(0 1 2 3))
              (if (= mode 3)
                  (setq L ())))))
    (if sw (setq name "local") (setq name "global"))
    (while (consp L)
      (setq x (car L) L (cdr L))
      (print "@<Get " name " variables" (vref mode_names x) "."  at-semi))
    (print "@<Kill " name " variables" (vref mode_names mode) extp at-semi)))
```

For each local or global variable v, we initialise it in the following way

```
delta_v = a_delta_v;
delta_v = delta_temp[it].v;
```

depending on whether v is local or global.

The important point here is that the name in the structure is the same as the local variable associated to it in case it is in `temp`, and it has a prefix `a_` before it otherwise. If the variable is Q in the previous notations, since Q is an inner matrix, we represent it by its numerator D (a matrix of polynomials) and its denominator q (a polynomial). The Schur algorithm computes Q_B as a function of Q_A. We write q instead of q_A and `nq` instead of q_B. This means that the data structure T contains `nq` and `nD`. This piece of code sets D and q to `nD`$[i]$ and `nq`$[i]$ for some i. This is the assignment $A \Rightarrow Q_{i-1}$ in the program scheme above.

```
(defun :access-Dq (loc extp mode)
  (let (L x e)
    (setq L (prev-modes mode))
    (while (consp L)
      (setq x (car L) L (cdr L))
      (setq e (vref prefixes x))
      (:print-subst "XD = Xtemp[Y].nD;" e loc)
      (:print-subst "Xq = Xtemp[Y].nq;" e loc))))
```

This generates the section 'Get variables, 3', that defines the last D and q (i.e. at location $n-1$) that is used to compute $\psi(Q)$, where $Q = D/q$.

```
(defun initialise-last-Dq ()
  (for (prec 0 1 3)
       (let ((extp (vref prec_types prec))
             (casep (vref precision_names prec)))
         (for (mode 0 1 3)
              (decl-sec "Get variables, 3")
              (:access-Dq "n-1" extp mode)))))
```

If we come back to our simplified program, we see that the assignments to A and B are: $A \Rrightarrow Q_{-1}$, $B \Rrightarrow Q_0$, $A \Rrightarrow Q_0$, $B \Rrightarrow Q_1$, $A \Rrightarrow Q_1$, $B \Rrightarrow Q_2$, etc. Except for the first one, we write $A = B$ (pointer assignment). In the next function `sw` is true if it is the first assignment. Note that $A = B$ is just `q=nq`. For the case of the first assignment, we assume that Q_{-1} is in the variable `n_Q`. Of course, the previous hack does not work if we differentiate the code in reverse mode, hence the test on `mode`.

```
(defun :access-mid-Dq0 (extp mode sw)
  (if (and sw (or (= mode 1) (= mode 3)))
      (:access-Dq "it-1" extp mode)
    (let (L x e name)
      (if sw (setq name "n") (setq name "g_"))
      (setq L (prev-modes mode))
      (while (consp L)
        (setq x (car L) L (cdr L))
        (setq e (vref prefixes x))
        (:print-subst "TY = TXY;" name "D" () e)
        (:print-subst "TY = TXY;" name "q" () e)))))
```

This generates the code in the any case. It prints 'if $i = 0$ then some code else some other code'.

```
(defun :access-mid-Dq (extp mode)
  (print "if(it==0) {")
  (:access-mid-Dq0 extp mode false)
  (print "} else { ")
  (:access-mid-Dq0 extp mode true)
  (print "}"))
```

Now the compiler warns about possible un-initialisation of `nD` and `nq`, because of the code $A = B, B = \ldots$, which may use B before it is set (the compiler does not understand the if test).

```
(defun make-compiler-happy ()
   (print "@ @<Special initialisation, direct@>=")
   (print "nD=NULL; nq= NULL;")
   (print "@ @<Special initialisation, delta@>=")
   (print "delta_nD=nD=NULL; delta_nq=nq= NULL;"))
```

This generates the section 'Get variables, 2', which is the section that is used in the inner loop, which is the header of the computation $B = T_\Theta(A)$. The variable A is defined by the first function call defined above, the variable B and the other variables that are needed are defined and killed by the second function call.

```
(defun initialise-midloop-vars ()
  (for (prec 0 1 3)
       (let ((extp (vref prec_types prec))
             (casep (vref precision_names prec)))
         (for (mode 0 1 3)
              (decl-sec "Get variables, 2")
              (:access-mid-Dq extp mode)
              (:get-and-kill-vars mode casep true)))))
```

This generates now the section ‘Get variables, 1’, which is the start of the C function. There are four section, one per differentiation mode, and these are all different.

```
(defun print-initialisation ()
  (for (prec 0 1 3)
       (let ((extp (vref prec_types prec))
             (prec1 (:prev-prec prec))
             (casep (vref precision_names prec)))
```

First part, computation of ψ. The function sets ψ to 1. It calls an initialisation function that sets Q_i (i.e. D_i and q_i) to zero, for each $i \geq 0$, and Q_{-1} to the identity (which is not done by `get-and-kill-vars`). It sets also the derivatives (for the reverse mode) of these quantities to zero.

```
         (show-usage)
         (print "@ @<Get variables, 1" case-direct casep at-equal)
         (print "   @<Indices@>@;")
         (print "   @<Local variables" case-direct  casep at-semi)
         (:initialise-psi "psi" prec)
         (print "   init_Dq();")
         (:get-and-kill-vars 0 casep false)
         (terpri)
```

Second part, computation of ψ', in reverse mode. It sets $d\psi$ to 1. It does not set the derivatives of Q_i (already done).

```
         (show-usage)
         (print "@ @<Get variables, 1" case-diff casep at-equal)
         (print "   @<Indices@>@;")
         (print "   @<Local variables" case-diff  casep at-semi)
         (:clear-a-div "dpsi" prec)
         (:get-and-kill-vars 1 casep false)
         (terpri)
```

Third part, computation of the Hessian. We compute the Hessian applied to a vector v_I. We define and kill global variables, set $\delta\psi$ and $\delta d\psi$ to zero, $d\psi$ to one, and after that, call the function `arl2mat_set_hess` that takes I as argument. It computes the vector v (which is somewhere in the data structure if $I = -1$, and is the Ith base vector otherwise), puts this vector into δy (recall that ψ is a function of y). This function sets also δQ and δdQ to zero.

```
         (show-usage)
         (print "@ @<Get variables, 1" case-delta casep at-equal)
         (print "   @<Indices@>@;")
         (print "   @<Local variables" case-deltadiff  casep at-semi)
         (print (vref types-vector prec1) " delta_psi, delta_dpsi, dpsi;")
         (:get-and-kill-vars 2 casep false)
         (print "set_hess(I);")
         (prin1 "delta_psi") (:call_clear_op prec1)
         (prin1 "delta_dpsi")(:call_clear_op prec1)
         (:print-set-one-only prec "dpsi")
         (terpri)
```

Last part. This declares and kills all variables δdY.

```
     (show-usage)
     (print "@ @<Get variables, 1" case-deltadiff casep at-equal)
     (:get-and-kill-vars 3 casep false))))
```

4.11.2 Managing results

This is needed to copy ψ and $\delta\psi$. Recall that ψ is, in the complex case, the only variable of real type in the program. However, the target may be complex.

```
(defun :copy-psi (src dest idx)
  (print dest "[" idx "]" (:real-ext prec) " = " src ";"))
```

When we differentiate in reverse mode $\psi = \psi(y)$, the derivative is in y', and y' is a set of complex vectors. We want to put the result in one real vector. This function is just a wrapper around `:gen-copy-cmplx2real`.

```
(defun :print-affect (dstart dend src prec)
  (let (ext)
    (setq ext (:real-ext prec))
    (:gen-copy-cmplx2real (catenate src "[j]") (catenate  dstart dend ext)
                           "k++;" prec)))
```

This prints the sections '`Code before returning`'. Here `sw` is 2 when we compute ψ, 0 when we compute the gradient, and 1 when we compute the Hessian.

```
(defun print-code-before-returning (prec sw)
  (let ((extp (vref prec_types prec))
        (casep (vref precision_names prec)))
    (show-usage)
    (print "@ @<Code before returning, "
         (if (= sw 2) "direct" (if (= sw 0) "diff" "delta diff"))
         casep at-equal)
```

This is not really needed: we store $\delta\psi$ somewhere. This should be the same as $d\psi$.

```
    (when (= sw 1)
      (prin1 "if(I>=0) ") (:copy-psi "delta_psi" "a_dgrad" "I"))
```

If `sw` is 2, we copy ψ into the data structure. We return the value of ψ, converted to double precision, if required. Otherwise for each i and j, we copy $s_{i,j}$ into d_k. The source s is dy (case of gradient), and δdy (case of Hessian). The destination is the field `grad` or `hess` in the data structure. In the case of the gradient, or Hessian applied to a vector, the starting index k is zero. In the case of full Hessian, the starting index is IN, where N is the size of one row of the Hessian, and I is the index of the component we compute.

```
    (if (= sw 2)
      (progn
        (:copy-psi "psi" "a_psi" 0)
        (if (or (= prec 0 ) (= prec 2))
            (print "return psi;")
            (print "return psi.to_double();")))
      (print "k=0;")
      (if (= sw 1)
          (print "if(I<0) w=0; else w="
                 (if (:prec-real prec) "" "2*") "I*n*p;"))
      (print "for(i=0;i<n;i++)")
      (print "for(j=0;j<p;j++){")
```

Table 4.14: Flags for memory allocation

flag	islocal	decl	init	kill	alloc
1	T	T	T	T	T
2	T	T	T	F	T
3	T	T	T	F	T
4	T	T	T	F	F
5	T	F	F	T	F
6	F	T	T	T	T
7	F	T	T	F	T
8	F	T	F	F	F
9	F	T	T	F	F

```
      (:print-affect
       (if (= sw 0) "a_grad" "a_hess")
       (if (= sw 1) "[w+k]" "[k]")
       (if (= sw 0) "dtemp[i].y"  "delta_dtemp[i].y" )
       prec)
      (print "}"))
))
```

4.11.3 Memory management

To each variable variable we associate a flag, an integer between 1 and 9, that has the meaning defined in table 4.14. This flag is formed of five bits, and explains how the variable is managed, as follows

1. In this case, the variable is a normal local variable. We have to declare it, kill it, initialise it, and allocate memory for it.
2. Same as 1, but the variable is not killed. There are three such variables: nD, nq and B. A special piece of code is used to kill nD and nq. The variable B has a special status: it contains β (independent of y) and some scalars, like $\|y\|^2$.
3. As above. There is only one variable of this type, namely y. Note that y is the input of the program, and it is initialised via a function `copy_y` not explained here. On the other hand, we shall give later on the code that initialises the derivatives of y. Note also that the memory for y is allocated in a special way.
4. The variables having this flag are u and uu^*. Since u is constant, there is no memory allocated for the derivative of it, and of course, we never put u to zero.
5. The variables having this flag are the scalars that appear in B. Since B is declared, initialised and allocated, we just have to kill these variables.
6. Like 1, but the variable is global.
7. Like 2, but the variable is global. The variables having this flag are `g_q` and `g_D`. They hold the identity matrix (Q_{-1} in the notations above).
8. This variable is to be declared only. It is D or q. When we compute $X_i = f(X_{i-1})$, we write it as $Y = f(X)$. We have to declare kill initialise and allocate memory for X_i (included the case $i = -1$), but X_{i-1} is to be declared only.
9. This is like 4, but the variable is global. The only variable of this type is G.

Table 4.15: Variables, size and flags

name	dim1	dim2	flag
`yD`	p	$I+1$	1
`E`	$I+1$	()	1
`M`	4	()	1
`N`	$p*p$	()	1
`X`	$p*p$	$I+1$	1
`Z`	$p*p$	$I+2$	1
`Y1`	p	$I+1$	1
`Y2`	$p*p$	$I+I+1$	1
`B`	8	()	2
`nq`	$I+2$	()	2
`nD`	$p*p$	$I+2$	2
`y`	p	()	3
`B4`	4	()	5
`(u)`	$*$	()	4
`(uu)`	$*$	()	4
`Gquo`	$\lambda*p$	$m+1$	6
`Grem`	$\lambda*p$	$m+1$	6
`P`	$\lambda*p$	$2*n$	6
`V`	$\lambda*p$	n	6
`g_q`	$n+1$	()	7
`g_D`	$p*p$	$n+1$	7
`G`	$*$	$*$	9
`q`	$*$	()	8
`D`	$*$	$*$	8
`tmp`	()	()	8

In the table 4.15, we give the list of all variables, with their flags. We give also two dimensions. These dimensions are needed for memory allocation. In this table p is the dimension of Q, λ is the first dimension of G, m is the degree of G, and n is the McMillan degree of the result. The quantity I is in fact `it`. It is the current index when we allocate memory for X_i. In the case where a variable has only one dimension, the second one is indicated by (). In the case where no memory is allocated, we use '*' to represent any size.

The next macro takes the next element in the list L. It sets some variables.

```
(dmd advance-L ()
  '(setq x (car L) L (cdr L)
         dim1 (vref x 1)
         dim2 (vref x 2)
         flag (vref x 3)
         x (vref x 0)
         info (vref info-array flag)))
```

The next function prints the declarations for a given mode and a given precision. We print a declaration unless `flag` is 5. If the variable is `(u)` or `(uu)`, it is constant, so that the derivatives are not declared. We have to print something like `double**X,**dX,**delta_X,**delta_dX;` for each variable.

```
(defun :generate-decl-aux (L)
  (let (x dim1 dim2 flag aux type stars nbdim y)
    (setq aux (prev-modes mode) type (vref types-vector prec))
    (while L
      (advance-L)
      (if dim2 (setq nbdim 2) (if dim1 (setq nbdim 1) (setq nbdim 0)))
      (setq stars (vref stars-vect nbdim))
      (if (consp x)
          (setq x (car x) y '(0))
          (setq y aux))
      (unless (= flag 5)
        (prin1 type)
        (while y
          (prin1 stars) (prin1 (vref prefixes (car y)))
          (prin1 x)
          (setq y (cdr y))
          (if y (prin1 ",")))
        (print ";")))))
```

The next function loops over all modes and precision.

```
(defun generate-decl (L)
 (for (prec 0 1 3)
        (for (mode 0 1 3)
             (unless (= mode 2)
                (decl-sec "Local variables")
               (:generate-decl-aux L)))))
```

The next function generates the sections 'Get local variables' and 'Get global variables', depending on the value of `sw`.

```
(defun compute-lp (varlist sw)
  (let (p pp L x dim1 dim2 flag info name loc)
    (if sw
        (setq name "Get local variables" loc "temp[it]." pp "")
        (setq name "Get global variables" loc "" pp "a_" ))
    (for (prec 0 1 3)
         (for (mode 0 1 3)
              (setq p (vref prefixes mode)
                    L varlist)
              (decl-sec1 name mode)
              (while L
                  (advance-L)
                  (if (consp x)
                      (if (= mode 0)
                          (setq x (car x)) (setq x ())))
                  (if (and x (eq sw (vref info 0)) (vref info 2))
                      (print p x " = " pp p loc x ";")))))))
```

4.11.4 Hand-written code

We consider here code that is not generated by the automatic differentiator.

We define a table B of size 8, that holds 3 polynomials of degree 1, B, B_2, B_4, and 2 numbers s and F. In the C code, s and F are called `ss` and `FF`. The constant term of B is at location 0, the coefficient of z at location 1. The constant term of B_2 is at location 2, the coefficient of z at location 3. Finally, the constant term of B_4 is at location 4, the coefficient of z at location 5.

```
(defun print-aliases ()
  (let (e (s "@d TX=(TY)"))
    (show-usage)
    (print "@ Rename some variables.")
    (for (i 0 1 3)
         (setq e (vref prefixes i))
         (:print-subst s "B2" "B+2" () e)
         (:print-subst s "B4" "B+4" () e)
         (:print-subst s "ss" "B[6]" () e)
         (:print-subst s "FF" "B[7]" () e))))
```

We have to compute $uy^* - y^*u$ in the case $n = 2$. If M contains uy^*, this is a 2×2 matrix. If F contains y^*u, this is a scalar. Because of our representation of matrices, the code is just $M_0 -= F$, and $M_3 -= F$.

```
(defun print-M-minus-F ()
  (for (prec 0 1 3)
       (for (mode 0 1 3)
            (decl-sec "Compute $M-F$")
            (if (or (= mode 0) (= mode 2))
                (progn
                  (:print-neg-equal "M[0]" "FF" (vref prefixes mode))
                  (:print-neg-equal "M[3]" "FF" (vref prefixes mode)))
              (progn
                (:print-neg-equal "FF" "M[0]"  (vref prefixes mode))
                (:print-neg-equal "FF" "M[3]"  (vref prefixes mode)))))))
```

The five next sections compute $s = \|y\|^2$ and its derivatives. Note that s is real, but declared complex, the imaginary part is always zero. We replace s by 1, if it is very near to it. (I don't know if this is very interesting. In theory, we should have $s \leq 1$).

```
(defun :s-code-mode0 (prec)
    (cond ((= prec 0)
           (print "ss += y[w]*y[w];"))
          ((= prec 1) (print "ss.add_mul(y[w],y[w]);"))
          ((= prec 2)
           (print "ss.r += y[w].r * y[w].r + y[w].i * y[w].i;")
           (print "if(fabs(ss.r-1.0)<1.e-15) ss.r = 1.0;"))
          ((= prec 3)
           (print "ss.r.square(y[w]);")
           (print "if(fabs(ss.r.to_double()-1.0)<1.e-15)")
           (print "  ss.r.get_one();"))))
```

This computes the derivative, in reverse mode, of $s = \|y\|^2$. This is just $dy \mathrel{+}= 2y\,ds$.

```
(defun :s-code-mode1 (prec)
    (cond ((= prec 0)
           (print "dy[w] += 2*y[w]*dss;"))
          ((= prec 1) (print "dy[w].two_times(y[w],dss);"))
          (true (print "dy[w].two_times_r(y[w],dss.r);"))))
```

This computes the derivative, in direct mode. The code is $\delta s \mathrel{+}= 2y\,\delta y$.

```
(defun :s-code-mode2 (prec)
  (cond ((= prec 0)
         (print "delta_ss += 2*delta_y[w]*y[w];"))
        ((= prec 1) (print "delta_ss.two_times(delta_y[w],y[w]);"))
        ((= prec 2)
         (print "delta_ss.r += 2*(delta_y[w].r*y[w].r+ delta_y[w].i*y[w].i);"))
        ((= prec 3)
         (print "delta_ss.r.two_times(delta_y[w],y[w]);"))))
```

And now the code for the Hessian $\delta dy \mathrel{+}= 2y\,\delta ds + 2\delta y\,ds$.

```
(defun :s-code-mode3 (prec)
    (print "{")
    (cond ((= prec 0)
           (print "delta_dy[w] += 2*delta_y[w]*dss;")
           (print "delta_dy[w] += 2*y[w]*delta_dss;"))
          ((= prec 1)
            (print "delta_dy[w].two_times(delta_y[w],dss);")
            (print "delta_dy[w].two_times(y[w],delta_dss);"))
          (true
            (print "delta_dy[w].two_times_r(delta_y[w],dss.r);")
            (print "delta_dy[w].two_times_r(y[w],delta_dss.r);")))
    (print "}"))
```

This calls one of the four previously defined functions.

```
(defun print-s-code (mode prec)
  (let ((extm (vref prefixes mode))
         (casem (vref mode_names mode))
         (extp (vref prec_types prec))
         (casep (vref precision_names prec))
        )
    (show-usage)
    (print "@ @<Code of s" casem casep at-equal)
    (print "   for(w=0;w<p;w++)")
    (cond ((eq mode 0) (:s-code-mode0 prec))
          ((eq mode 1) (:s-code-mode1 prec))
          ((eq mode 2) (:s-code-mode2 prec))
          ((eq mode 3) (:s-code-mode3 prec)))))
```

Big hack for computing B. More will come later. We assume that B_0 and B_1 hold the coefficients of $b = (z-\omega)(1-\overline{\omega})$. We compute in B_4 and B_5 the coefficients of $b_4 = b - \|y\|^2\tilde{b}$. What we have to do is compute $B_4 = B_0 - s\overline{B_1}$, $B_5 = B_1 - s\overline{B_0}$, where $s = \|y\|^2$. If `prec` is zero (real, double case) we compute everything, otherwise just $B_4 = B_0$ and $B_5 = B_1$.

This computes B_4 if `prec` is zero, sets B_4 to b otherwise.

```
(defun :print-B-code-direct (mode prec)
  (if (= prec 0)
      (progn
         (print "   B[4] = B[0] - ss * B[1];")
         (print "   B[5] = B[1] - ss * B[0];"))
    (print "B[4] = B[0];")
    (print "B[5] = B[1];")))
```

This is the derivative in direct mode of the previous code. We have to compute $\delta B_4 = -\delta s\,\overline{B_1}$ and $\delta B_5 = -\delta s\,\overline{B_0}$.

```
(defun :print-B-code-delta (mode prec)
  (if (= prec 0)
      (progn
         (print "   delta_B[4] = -B[1] * delta_ss;")
         (print "   delta_B[5] = -B[0] * delta_ss;"))
       (prin1 "delta_B[4]") (:call_clear_op prec)
       (prin1 "delta_B[5]") (:call_clear_op prec)))
```

In case the mode is 0 or 2, there is nothing to do if the precision is zero. Otherwise, we have to execute $B_4 \mathrel{-}= \overline{B_1}s$, $B_5 \mathrel{-}= \overline{B_0}s$ or $\delta B_4 \mathrel{-}= \overline{B_1}\,\delta s$, $\delta B_5 \mathrel{-}= \overline{B_0}\,\delta s$. If the mode is 1, we differentiate in reverse mode, hence write $ds \mathrel{-}= B_1\,dB_4$, $ds \mathrel{-}= B_0\,dB_5$. Finally, if the mode is 3, we generate $\delta ds \mathrel{-}= B_1\,\delta dB_4$, $\delta ds \mathrel{-}= B_0\,\delta dB_5$.

```
(defun :B-code-hack (mode prec)
  (let ((extm (vref prefixes mode))
        (casem (vref mode_names mode))
        (extp (vref prec_types prec))
        (casep (vref precision_names prec))
        (sub-mul-mac ':op-sub-mul-K1)
        (sub-mul-conj-mac ':op-sub-mul-conj-K1)
        op
       )
       (if (or (= mode 1) (= mode 3))
           (setq op sub-mul-mac)
           (setq op sub-mul-conj-mac))
       (when (or (= mode 1) (= mode 3) (> prec 0))
             (if (or (= mode 0) (= mode 2))
                 (progn
                   (funcall op prec '("B[1]" "ss" "B[4]") extm)
                   (funcall op prec '("B[0]" "ss" "B[5]") extm))
               (progn
                 (funcall op prec '("B[1]" "B[4]" "ss") extm)
                 (funcall op prec '("B[0]" "B[5]" "ss") extm))))))
```

This is the code that computes s and B_4. Note the order of instructions.

```
(defun print-B-code (mode prec)
       (decl-sec "Complete code of B")
       (if (or (= mode 0) (= mode 2))
           (use-sec "Code of s"))
       (cond ((eq mode 0) (:print-B-code-direct mode prec))
             ((eq mode 2) (:print-B-code-delta mode prec)))
       (:print-B-code-hack mode prec)
       (if (or (= mode 1) (= mode 3))
           (use-sec "Code of s"))))
```

This copies `G` into `Grem`. Remember: a copy of G is needed for the division in the first equation of (4.18.a).

```
(defun print-copy ()
 (for (mode 0 1 3)
      (for (prec 0 1 3)
           (let ((casem (vref mode_names mode))
                 (casep (vref precision_names prec)))
             (show-usage)
             (print "@ @<Copy $G$ into |Grem|" casem casep "@>=")
             (when (= mode 0)
                   (print "for(k=0;k<gdim1*p;k++)")
                   (print "for(i=0;i<=m;i++)")
                   (print "Grem[k][i] = G[k][i];"))))))
```

4.11.5 Memory management

This piece of code generates the code that clears all variables; depending of the value `sw`, we kill local or global variables.

```
(defun generate-clear (l sw)
  (let (x dim1 dim2 flag info L prefix idx)
    (for (prec 0 1 3)
         (for (mode 0 1 3)
             (decl-sec (if sw "Kill local variables" "Kill global variables"))
             (setq prefix (vref prefixes mode))
             (setq L l)
             (while (consp L)
               (advance-L)
               (when (and (vref info 3) (eq sw (vref info 0)))
                 (print "for(i=0;i<" dim1 ";i++)")
                 (setq idx "[i]);")
                 (when dim2
                   (setq idx "[i][j]);")
                   (print "for(j=0;j<" dim2 ";j++)"))
                 (prin  prefix x idx) (:call_clear_op prec)))))))
```

This generates all declarations, initialisations, etc.

```
(defun declarations ()
  (generate-clear big-varlist true)
  (generate-clear big-varlist false)
  (compute-lp big-varlist true)
  (compute-lp big-varlist false)
  (generate-decl big-varlist))
```

4.11.6 Auxiliary code

The code generated in this section is printed in another file. Memory allocation for global variables is trivial: we print something like `g_q = (Type) my_malloc_vector(n+1,type);` for each variable. The C variable `type` contains the current precision. Note that C++ wants a type, this is `matrix` or `polynom`, which are `typedefs` in the `template`.

```
(defun print-a-global-malloc (L)
  (let (dim1 dim2 x flag info extm)
    (show-usage)
    (print "@ @<Allocate memory for all global variables@>=")
    (while L
      (advance-L)
      (when (and (vref info 4) (not (vref info 0)))
        (for (mode 0 1 3)
             (setq extm (vref prefixes mode))
             (prin1 "  a_") (prin1 extm) (prin1 x)
             (if dim2
                 (print " = (matrix) my_malloc_matrix(" dim1 ","
                        dim2 ",type);")
                 (print " = (polynom) my_malloc_vector(" dim1 ",type);")))))))
```

In the case of local variables, we print something like `temp[it].B = (Type) alloc(8,type);` for each variable. The allocator is `my_malloc_matrix` or `my_malloc_vector`. The size may depend on `it` (this is because the k-th matrix we construct has McMillan degree k). Everything is put in a loop, and a section is created for each mode. Special hack: space for y is allocated elsewhere, space for the derivatives of y is allocated here.

```
(defun print-a-local-malloc (l)
  (let (dim1 dim2 x flag info extm L)
    (for (mode 0 1 3)
         (setq extm (vref prefixes mode))
         (setq L l)
         (show-usage)
         (print "@ @<Fill |" extm "temp|@>=")
         (print "for(it=0;it<n;it++){")
         (while (consp L)
           (advance-L)
           (when (and (vref info 4) (vref info 0))
             (if (= flag 3)
                 (if (= mode 0) (setq dim1 ())))
             (when dim1
               (prin1 extm)(prin1 "temp[it].") (prin1 x)
               (prin1 " = ")
               (if dim2 (prin1 "(matrix) my_malloc_matrix(")
                 (prin1 "(polynom) my_malloc_vector("))
               (prin1 dim1)
               (when dim2 (prin1 ",") (prin1 dim2))
               (print ",type);"))))
           (print "}"))))
```

As explained above, the quantities D and q are initialised in a special way. In the next function, if `aux` is empty, we put zero in Q_i and dQ_i $(-1 \leq i < n)$. This piece of code is executed whenever we compute ψ, and never when we compute the derivative. As a side effect, when we compute ψ, some of the code here may be useless. We also put zero in dy.

If the value of `aux` is `delta_`, we put zero in δQ_i, δdQ_i and δdy_i. This is used when we compute the second derivative of ψ.

```
(defun print-init-Dq-aux (prec extp aux)
     (print "{")
     (print "int p = a_p;")
     (print "int n = a_n;")
     (print "int i,j,k;")
     (prin "a_" aux "g_q[0]") (:call_clear_op prec)
     (prin "a_" aux "dg_q[0]") (:call_clear_op prec)
     (print "for(i=0;i<p*p;i++) {")
     (prin "a_" aux "g_D[i][0]")(:call_clear_op prec)
     (prin "a_" aux "dg_D[i][0]")(:call_clear_op prec)
     (print "}")
     (print "for(k=0;k<n;k++) ")
     (print " for(i=0;i<=k+1;i++) {")
     (print "    for(j=0;j<p*p;j++) {")
     (prin  aux "temp[k].nD[j][i]")(:call_clear_op prec)
     (prin  aux "dtemp[k].nD[j][i]")(:call_clear_op prec)
     (print "}")
     (prin  aux "temp[k].nq[i]")(:call_clear_op prec)
     (prin  aux "dtemp[k].nq[i]")(:call_clear_op prec)
     (print "}")
     (print "for(k=0;k<n;k++)")
     (print " for(i=0;i<p;i++)")
     (prin  aux "dtemp[k].y[i]" )(:call_clear_op prec))
```

This defines for each precision a function that calls the previous code. It puts in Q_{-1} the identity matrix. In other words, it puts some ones in the variables `g_q` and `g_D`. Then, it defines a single function that calls the one that corresponds to the current precision.

```
(defun :generate_init_D ()
  (for (prec 0 1 3)
       (let ((extp (vref prec_types prec)))
         (show-usage)
         (print "@ @u void data2p" extp "::init_Dq()")
         (print-init-Dq-aux prec extp "")
         (set-cr-one "a_g_q" "[0]" prec "")
         (print "for(i=0;i<p;i++)")
         (set-cr-one "a_g_D" "[i*(p+1)][0]" prec "")
         (print "}")))
  (show-usage))
```

This defines for each precision a function, that sets every δD, δdD, δq and δdq to zero.

```
(defun :init-delta-dq ()
  (for (prec 0 1 3)
       (let ((extp (vref prec_types prec)))
         (show-usage)
         (print "@ @u void data2p" extp "::init_delta_Dq()")
         (print-init-Dq-aux prec extp "delta_")
         (print "}"))))
```

The code of the Hessian always computes the Hessian applied to a vector v_I, which is the Ith base vector (we do not consider here the case $I = -1$, case where v_I is set by the caller). This means that we have to set δY to the vector that contains 1 in location I. However δY is a set of n vectors of size p that may be complex. Assume $J_1 = I$ and $J_2 = I - 1$. Write

$$J_1 = c_k + pc_i \qquad J_2 = p_k + pp_i$$

by Euclidean division by p. Now, location J_1 in Y is location c_k in y_{c_i}. Assume $I > 1$. What we computed before is the Hessian of our function applied to v_{I-1}. Thus, we have zeroes everywhere, but in $I - 1$, which is location p_k in y_{p_i}. In the complex case, we let $J_1 = I/2$, and $J_2 = (I - 1)/2$. Thus we have to put a one at location c_k in y_{c_i}, using the real part if I is even, the imaginary part if I is odd. This function computes these funny quantities.

```
(defun :compute-indices ()
       (if (:prec-imag prec)
           (print "int is_even;"))
       (if (:prec-real prec)
           (print "J1=I;J2=I-1;")
         (print "is_even = I&1 ? 0 : 1;")
         (print "J1=I/2;J2 = (I-1)/2;"))
       (print "cur_i = J1/p;")
       (print "cur_k = J1 - p*cur_i;")
       (print "prev_i = J2/p;")
       (print "prev_k = J2-p*prev_i;")
)
```

In case $I = 0$, we put zeroes everywhere, otherwise at location J_2 (in the complex case, we put a zero in the real and imaginary part).

```
(defun :set-zero-where-indicated ()
       (print "if(I==0)")
       (print "for(i=0;i<n;i++) {")
       (print "delta_y = delta_temp[i].y;")
       (print "for(k=0;k<p;k++)")
       (prin  "delta_y[k]")(:call_clear_op prec)
       (print " } else  {")
       (prin  "delta_temp[prev_i].y[prev_k]")(:call_clear_op prec)
       (print "}")
```

This is the main function. It sets δy, and calls the function that clears the derivatives of Q and y. There is one function per precision.

```
(defun :print-set-hess1 ()
  (let (e var)
  (for (prec 0 1 3)
       (setq e (vref prec_types prec))
       (show-usage)
       (print "@ @u void data2p" e "::set_hess(int I)")
       (print "{")
       (print "int p = a_p, n = a_n;")
       (print "int i,k,J1,J2;")
       (print "int cur_i,cur_k, prev_i,prev_k;")
       (print (vref types-vector prec) "* delta_y;")
       (:compute-indices)
       (:set-zero-where-indicated)
       (setq var (catenate  "delta_temp[cur_i].y[cur_k]"))
       (if (:prec-real prec)
           (:print-set-one-only prec var)
         (print "if(is_even)")
         (:print-set-one-only prec (:real var prec))
         (print "else")
         (:print-set-one-only prec (:imag var prec)))
       (print "init_delta_Dq();")
       (print "}"))))
```

The Lisp function that generates all the code that is put in the 'arl2mat2aux' file.

```
(defun init-aux2 ()
  (:print-bk-code)
  (:generate_init_D)
  (:init-delta-dq)
  (:print-set-hess1)
)
```

4.11.7 Inverse Schur code

In this section, we compute $b = (z - \omega)(1 - \overline{\omega})$ and some formulas needed by the inverse Schur algorithm.

Recall that $b = (z - \omega)(1 - \overline{\omega})$. We assume that `or` and `oi` hold the real and imaginary part of ω. We have $b = B_0 + zB_1$, $b - \tilde{b} = B_2 + zB_3$, i.e. $B_0 = |\omega|^2 - \omega$, $B_1 = 1 - \overline{\omega}$, $B_2 = |\omega|^2 - 1$, $B_3 = -B_2$. This is code for the real case. We use the relation $B_0 = -\omega B_1$.

```
(defun :print-bk-real ()
  (show-usage)
  (print "@")
  (print "@u void data2pD::arl2mat_compute_B()")
  (print "{")
  (print "int it, n=a_n;")
  (print "for(it=0;it<n;it++) {")
  (print "double or = a_omega[it];")
  (print "double* B= temp[it].B;")
  (print "  B[1] = 1.0-or;")
  (print "  B[0] = -or *B[1];")
  (print "  B[2] = or *or-1.0;")
  (print "  B[3] = -B[2];")
  (print "}}"))
```

This is the complex code. We put $|\omega|^2$ in x, so that $B_0 = x - \omega$.

```
(defun :print-bk-complex ()
  (show-usage)
  (print "@")
  (print "@u void data2pC::arl2mat_compute_B()")
  (print "{")
  (print "int it, n=a_n;")
  (print "for(it=0;it<n;it++) {")
  (print "  double or,oi,x;")
  (print "  Complex* B= temp[it].B;")
  (print "  or = a_omega[it].r;")
  (print "  oi = a_omega[it].i;")
  (print "  B[1].r = 1.0-or;")
  (print "  B[1].i = oi;")
  (print "  x = or*or + oi*oi;")
  (print "  B[0].r = x -or;")
  (print "  B[0].i = -oi;")
  (print "  B[2].r = x-1.0;")
  (print "  B[3].r = -B[2].r;")
  (print "  B[2].i = 0.0;")
  (print "  B[3].i = 0.0;")
  (print "}}"))
```

This is the code in the SLD case. Note that we declare `or`, `B`, and a quantity `t1` that holds the constant one. We use $B_0 = -\omega B_1$, $B_2 = |\omega|^2 - 1$. The code is: $B_3 = |\omega|^2$, $B_2 = B_3 - 1$, then $B_3 = -B_2$.

```
(defun :print-bk-SLD ()
  (show-usage)
  (print "@")
  (print "@u void data2pS::arl2mat_compute_B()")
  (print "{")
  (print "int it, n=a_n;")
  (print "for(it=0;it<n;it++)")
  (print "{")
  (print "  SLD t1;")
  (print "  SLD& or = a_omega[it];")
  (print "  SLD * B=temp[it].B;")
  (print "  t1.get_one();")
  (print "  B[1].sub(t1,or);")
  (print "  B[0].mul(or,B[1]);")
  (print "  B[0].neg();")
  (print "  B[3].mul(or,or);")
  (print "  B[2].sub(B[3],t1);")
  (print "  B[3].neg(B[2]);")
  (print "}}"))
```

And this is the code in the complex SLD case. Note that we declare `or`, `oi`, `B`, and two constants `t1`, `t2` that hold 1 and $|\omega|^2$.

```
(defun :print-bk-SLD-complex ()
  (show-usage)
  (print "@")
  (print "@u void data2pSC::arl2mat_compute_B()")
  (print "{")
  (print "int it, n=a_n;")
  (print "for(it=0;it<n;it++)")
  (print "{")
  (print "  SLD t1,t2;")
  (print "  SLD_complex * B=temp[it].B;")
  (print "  SLD& or = a_omega[it].r;")
  (print "  SLD& oi = a_omega[it].i;")
  (print "  t1.get_one();")
  (print "  B[1].r.sub(t1,or);")
  (print "  B[1].i = oi;")
  (print "  t2.mul(or,or);")
  (print "  t2.add_mul(oi,oi);")
  (print "  B[0].r.sub(t2,or);")
  (print "  B[0].i.neg(oi);")
  (print "  B[2].kill();")
  (print "  B[3].kill();")
  (print "  B[2].r.sub(t2,t1);")
  (print "  B[3].r.neg(B[2].r);")
  (print "}}"))
```

This is the code that computes B in every case. We declare the variables in case this is not yet done.

```
(defun :print-bk-code ()
    (:print-bk-real)
    (:print-bk-complex)
    (:print-bk-SLD)
    (:print-bk-SLD-complex))
```

From now on, we consider the inverse Schur algorithm. We use a data structure, called T, that contains now Lisp objects, instead of raw pointers. The first part of the algorithm computes y such that $y = Q(\omega)^* u$. We do not explain here how u and ω are computed. The important point is that `omega` and `u` are some real vectors of size n and np (in the complex case, of size $2n$ and $2np$) and we extract the j value as a real or complex vector, and put it in `cur_omega` and `cur_u`.

This copies ω.

```
(defun :print-construct-cur-omega ()
  (let (ext1 ext2 x y)
    (for (prec 0 1 3)
         (decl-sec0 "Construct |cur_omega|" prec)
         (if (:prec-real prec)
             (print "k=j;")
           (print "k=2*j;"))
         (setq ext2 (vref prec_types prec))
         (setq ext1 (vref prec_types (:prev-prec prec)))
         (setq x (catenate "pol_coef" ext1 "(omega)[k]"))
         (setq y (catenate "pol_coef" ext2 "(cur_omega)[0]"))
         (:gen-copy-real2cmplx x y  "k++;" prec))))
```

This copies u.

```
(defun :print-construct-cur-u ()
  (let ( m)
    (for (prec 0 1 3)
         (decl-sec0 "Construct |cur_u|" prec)
         (print "{")
         (print (vref types-vector prec) "*U=mat_coef"
           (vref prec_types prec) "(cur_u);")
         (setq m (:prev-prec prec))
         (print (vref types-vector m) "*cu=pol_coef" (vref prec_types m) "(u);")
         (if (:prec-real prec)
             (print "k=j*p;")
           (print "k=2*j*p;"))
         (print "for(i=0;i<p;i++){")
         (:gen-copy-real2cmplx "cu[k]" "U[i]" "k++;" prec)
         (print "}}"))))
```

First part of Schur inverse. We fetch ω and u, compute $\tilde{D}$ and $\tilde{q}$. If Q is the quotient of these quantities, we evaluate the result at ω, then compute $y = Q(\omega)^* u$. Note that $Q = D_B/\tilde{q}_B$, and $D = \tilde{D}_B$, $q = q_B$, so that D/q is the tilde of Q. We tilde these matrices, evaluate the result, and tilde it. This gives $Q(\omega)$. If we multiply by u, we get y.

```
(defun :print-schur-inverse1 ()
  (print "@ @u")
  (print "void Schur_inverse::part1(int j)")
  (print "{")
  (print "int i,k;")
  (print "switch(pol_type){")
  (for (prec  0 1 3)
       (:print-switch-case prec)
       (use-sec0 "Construct |cur_omega|" prec)
       (use-sec0 "Construct |cur_u|" prec)
       (print "       break;"))
  (print "}")
  (print "}"))
```

This computes $s = \|y\|^2$. Here `y` is a pointer into some array, namely `mat_coef(T->cur_y).D`.

```
(defun :print-ss-code-inv ()
  (for (prec 0 1 3)
       (decl-sec0 "Compute $\\norm y^2$ in |s|" prec)
       (prin "ss")(:call_clear_op prec)
       (print "for(w=0;w<p;w++) {")
       (:print-add-norm "y[w]" (:real "ss" prec) prec)
       (print "}")))
```

This defines `or` and `oi`, the real and imaginary part of ω.

```
(defun :fetch-or-oi (prec)
  (let ((x "pol_coef") (y "(cur_omega)[0]") (z (vref prec_types prec)))
    (cond ((= prec 0) (print "or = " x z y ";"))
          ((= prec 1) (print "SLD& or = " x z y ";"))
          ((= prec 2)
           (print "or = " x z y ".r;")
           (print "oi = " x z y ".i;"))
          ((= prec 3)
           (print "SLD& or = " x z y ".r;")
           (print "SLD& oi = " x z y ".i;")))))
```

This computes $B_0 = (z-\omega)(1-\overline{\omega})$. We compute first the coefficient of z, namely $1-\overline{\omega}$, then $x = |\omega|^2$, finally, the constant term, which is $x - \omega$. We put 1 in `t1`.

```
(defun :print-b0-code-inv ()
  (let (prec1)
    (for (prec 0 1 3)
         (setq prec1 (:prev-prec prec))
         (decl-sec0 "Put $(z-\\omega)(1-\\overline\\omega)$ in $B_0$" prec)
         (:fetch-or-oi prec)
         (:print-set-one-only prec "t1")
         (:op-sub prec1
           (list "t1" "or" (if (:prec-real prec) "B0[1]" "B0[1].r")))
         (when (:prec-imag prec)
               (print (:imag "B0[1]" prec) "=oi;"))
         (:print-add-norm1 "or" "oi" "x" prec)
         (:op-sub prec1
           (list "x" "or" (if (:prec-real prec) "B0[0]" "B0[0].r")))
         (when (:prec-imag prec)
               (:print-neg "oi" (:imag "B0[0]" prec) prec1)))))
```

This puts $b - \tilde{b}$ into B_1. The coefficient of z is $1 - x$, the constant term is the opposite of it.

```
(defun :print-b1-code-inv ()
  (let (prec1)
    (for (prec 0 1 3)
         (setq prec1 (:prev-prec prec))
         (decl-sec0 "Put $b-\\tilde b$ in $B_1$" prec)
         (when (:prec-imag prec)
               (prin "B1[0]")(:call_clear_op prec)
               (prin "B1[1]")(:call_clear_op prec))
         (:op-sub prec1
           (list  "x" "t1" (if (:prec-real prec) "B1[0]" "B1[0].r")))
         (if (:spec-imag prec)
              (:print-neg "B1[0].r" "B1[1].r" prec1)
           (:print-neg "B1[0]" "B1[1]" prec1)))))
```

This puts $\tilde{b} - \|y\|^2 b$ in B_2. If $b = A + zB$, we generate $\overline{B} - sA + z(\overline{A} - sB)$ where $s = \|y\|^2$.

```
(defun :print-b2-code-inv ()
  (let (prec1 op)
    (for (prec 0 1 3)
         (setq prec1 (:prev-prec prec))
         (decl-sec0 "Put $\\tilde b-\\norm y b$ in $B_2$" prec)
         (:op-transpose prec '("B0[1]" "B2[0]"))
         (:op-transpose prec '("B0[0]" "B2[1]"))
         (:op-sub-mul-K prec '("B0[0]" "ss" "B2[0]"))
         (:op-sub-mul-K prec '("B0[1]" "ss" "B2[1]")))))
```

Prints the test: is ω zero?. According to theorem 21, $(z-\omega)(1-\overline{\omega}z)$ divides every term in (3.11), and we have to do the division. This polynomial is of degree 2, unless $\omega = 0$, case where the degree is one.

```
(defun :print-deg-test (prec)
  (cond ((= prec 0)  (print "or==0)"))
        ((= prec 2) (print "or==0 && oi==0)"))
        ((= prec 1) (print "or.is_zero())"))
        ((= prec 3) (print "or.is_zero() && oi.is_zero())"))))
```

Put $-(z-\omega)(1-\bar\omega z)$ in B_3. Note that B_3 is a Lisp polynomial, whose degree is zero or one. We set the degree. The polynomial is $\overline{\omega}x^2-(1+x)z+\omega$, with $x=|\omega|^2$.

```
(defun :print-b3-code-inv ()
  (let (prec1 op)
    (for (prec 0 1 3)
         (setq prec1 (:prev-prec prec))
         (decl-sec0 "Put $-(z-\\omega)(1-\\bar \\omega z)$ in $B_3$" prec)
         (print (:real "B3[0]" prec) " = or;")
         (print (:real "B3[2]" prec) " = or;")
         (when (:prec-imag prec)
               (print (:imag "B3[0]" prec) " = oi;")
               (:print-neg "oi" (:imag "B3[2]" prec) prec1))
         (prin "B3[1]")(:call_clear_op prec)
         (if (:prec-imag prec)
             (progn
               (:op-add prec1 '("t1" "x" "B3[1].r"))
               (:op-neg prec1 '("B3[1].r" "B3[1].r")))
           (:op-add prec1 '("t1" "x" "B3[1]"))
           (:op-neg prec1 '("B3[1]" "B3[1]")))
         (prin1 "if(")
         (:print-deg-test prec)
         (print "pol_deg(b3)=1; else pol_deg(b3) = 2;")
)))
```

Prints the declarations of all variables involved in the previous code.

```
(defun :print-inv-header ()
  (let (prec1 op e)
    (for (prec 0 1 3)
         (setq prec1 (:prev-prec prec))
         (decl-sec0 "Initialize inverse" prec)
         (setq op (vref types-vector prec))
         (setq e (vref prec_types prec))
         (print op " * B0 = pol_coef" e "(b0);")
         (print op " * B1 = pol_coef" e "(b1);")
         (print op " * B2 = pol_coef" e "(b2);")
         (print op " * B3 = pol_coef" e "(b3);")
         (print op " * y = mat_coef" e "(cur_y);")
         (print op " ss;")
         (setq op (vref types-vector prec1))
         (print op " x,t1;")
         (if (<> prec1 1) (print op " or;"))
         (if (= prec 2) (print op " oi;"))
         (print "int w;"))))
```

Constructs a section ‘Compute b_k and related constants’. This is the start of the function `schur_inverse2`. Remaining of the code is written directly in C.

```
(defun :merge-inv ()
  (for (prec 0 1 3)
       (decl-sec0 "Main inverse code" prec)
       (print "{")
       (use-sec0 "Initialize inverse" prec)
       (use-sec0 "Compute $\\norm y^2$ in |s|" prec)
       (use-sec0 "Put $(z-\\omega)(1-\\overline\\omega)$ in $B_0$" prec)
       (use-sec0 "Put $b-\\tilde b$ in $B_1$" prec)
       (use-sec0 "Put $\\tilde b-\\norm y b$ in $B_2$" prec)
       (use-sec0 "Put $-(z-\\omega)(1-\\bar \\omega z)$ in $B_3$" prec)
       (print "}"))
  (print "@ @<Compute $b_k$ and related constants@>=")
  (print "switch (prec_type) {")
  (for (prec  0 1 3)
       (:print-switch-case prec)
       (use-sec0 "Main inverse code" prec)
       (print "       break;"))
  (print "}"))
```

The Lisp function that computes all that is put in the file ‘arl2mat3aux’.

```
(defun print-schur-inverse ()
  (:print-construct-cur-omega)
  (:print-construct-cur-u)
  (:print-schur-inverse1)
  (:print-ss-code-inv)
  (:print-b0-code-inv)
  (:print-b1-code-inv)
  (:print-b2-code-inv)
  (:print-b3-code-inv)
  (:print-inv-header)
  (:merge-inv)
)
```

4.12 Main file

The description of all variables is found in table 4.16. Note that each variable has three dimensions here.

4.12.1 Preliminaries

Start of the file: load the differentiator, the patterns, the file with the main Lisp code. We open also the output file.

```
(load "diff.ll")
(load "patterns.ll")
(load "make_hessian1.ll")
(outchan (openo "arl2mat.web"))
```

We omit here the start of the WEB file: it contains some TEX header, and a C header, formed of C includes, and WEB includes.

4.12.2 The code of the functions

```
(print "@* Main code.")
(print "These are the user callable functions.")

(print-fct-00)
(initialise-last-Dq)
(initialise-midloop-vars)
(print-initialisation)
(print-M-minus-F)
```

```
(for (prec 0 1 3)
     (print-code-before-returning prec 0)
     (print-code-before-returning prec 1)
```

Table 4.16: List of variables

name	specification
y	(y p 1 0)
u	(u p 1 0 constant)
yt	(y p 1 0 transpose)
ut	(u p 1 0 transpose constant)
D	(D p p it)
E	(E 1 1 it)
q	(q 1 1 it)
yD	(yD 1 p it)
Eneg	(E 1 1 it neg)
F	(FF 1 1 0)
M	(M p p 0)
N	(N p p 0)
qt	(q 1 1 it transpose)
qdiv	(q 1 1 it div)
qneg	(q 1 1 it neg)
uu	(uu p p 0 constant)
Z	(Z p p it)
X	(X p p it)
Y	(Y p p it)
Gcopy	(Grem gdim1 p m)
Grem	(Grem gdim1 p n-1)
Gremneg	(Grem gdim1 p n-1 neg)
Y1	(Y1 p 1 it)
Y2	(Y2 p p it+it)
P	(P gdim1 p n-1+n)
V	(V gdim1 p n-1)
psi	(psi 1 1 0 psi)
s	(ss 1 1 0)
B2	(B2 1 1 1 constant)
B2neg	(B2 1 1 1 neg constant)
B4	(B4 1 1 1)
lastD	(D p p n)
lastqdiv	(q 1 1 n div)
G	(G gdim1 p m constant)

Table 4.17: All calls to differentiate

arg1	arg2	output	section name
yt	D	yD	Code of —yD—
yD	u	E	Code of *E*
yt	u	FF	Code of *F*
u	yt	M	Code of M
y	ut	N	Code of N
qt	M	Z	Code of *Z*
D	uu	X	Code of X, 1
y	yD	X	Code of X, 2
qneg	N	X	Code of X, 3
D	u	Y1	Code of Y, 1
Y1	yD	Y2	Code of Y, 2
Eneg	D	Y2	Code of Y, 3
Y2	qdiv	Z	Code of Y, 4
B4	D	nD	New D, 1
B2	Z	nD	New D, 2
B2neg	X	nD	New D, 3
B4	q	nq	New q, 1
B2neg	E	nq	New q, 2
Gcopy	lastqdiv	Gquo	First division
Grem	lastD	P	Code of P
P	lastqdiv	V	Code of V
V	psi	psi	Code of psi, 1
Gremneg	psi	psi	Code of psi, 2

```
     (print-code-before-returning prec 2)
     )

(print_a_switch_call1 "matrix_psi_hess" 3)
(print_a_switch_call1 "matrix_psi_hess_single" 4)
(terpri)
```

```
(show-usage)
(print "@* Hand written code.")
(print "@ Indices used everywhere. @<Indices@>=")
(print "    int p=a_p;")
(print "    int n = a_n, m=a_m;")
(print "    int i,j,k,l,w,it;")
(terpri)

;;; define the aliases.
(print-aliases)

(print "")
(print "@")
(print "")
(for (prec 0 1 3)
     (for (mode 0 1 3)
          (print-B-code mode prec)))


(for (prec 0 1 3)
       (for (mode 0 1 3)
            (print-s-code mode prec)))
```

```
(print "@* The functions.")
(print "    ")
(print "@ ")
(print "")
(print "")
(print "@ @<External functions@>=")
(print "extern void * GC_malloc(size_t size_in_bytes);")
(print "extern void * GC_malloc_atomic(size_t size_in_bytes);")
```

4.12.3 The calls to the differentiator

```
(print "@* Code of Z.")
(differentiate1
 ‘("Complete code of $Z$"
   (,yt ,u FF "Code of $F$" "Define $F$ to be $y^*u$"
        "We call this |FF|, since it’s an alias.")
   (,u ,yt M "Code of M" "Define $M$ to be $uy^*$")
   "Compute $M-F$"
   (,qt ,M Z "Code of $Z$" "Compute now $M\\tilde q$ in $Z$.")))
```

```
(print "@* Code of Y.")
(differentiate1 (list
   "Code of $Y$"
   (list D u ’Y1 "Code of Y, 1")
   (list Y1 yD ’Y2 "Code of Y, 2")
   (list Eneg D ’Y2 "Code of Y, 3")
   (list Y2 qdiv ’Z "Code of Y, 4")
))
```

```
(print "@* Code of X, Y and Z.")
(differentiate1 (list
   "Prepare $X$ and $Y$"
   "Complete code of B"
   (list yt D 'yD "Code of |yD|")
   (list yD u 'E "Code of $E$")
   (list y ut 'N "Code of N")
   (list D uu 'X "Code of $X$, 1")
   (list y yD 'X "Code of $X$, 2")
   (list qneg N 'X "Code of $X$, 3")
   (list 'if "(p==2)" "Complete code of $Z$")
   (list 'if "(p>2)" "Code of $Y$")
))

(print "@* Inner loop.")
(differentiate1 (list
  "The main loop"
  "Prepare $X$ and $Y$"
  (list B4 D 'nD "New D, 1")
  (list B2  Z 'nD "New D, 2")
  (list B2neg  X 'nD '"New D, 3")
  (list B4 q 'nq "New q, 1")
  (list B2neg E 'nq "New q, 2")
))

(print "@* Outer code.")

(print-copy)

(differentiate1 (list
   "Remaining of the  code"
   "Copy $G$ into |Grem|"             ; generated by print-copy.
   (list Gcopy lastqdiv 'Gquo "First division")
   (list Grem lastD 'P "Code of P")
   (list P lastqdiv 'V "Code of V")
   (list V psi 'psi "Code of psi, 1")
   (list Gremneg psi 'psi "Code of psi, 2")
))

(print "@* The functions.")
(declarations)
(make-compiler-happy)
(print "@* Index.")
```

We print now to a second file.

```
(close (outchan))
(outchan ())
(outchan (openo "arl2mat2aux.web"))
(print "@ Start of include file arl2mat2aux.web.")
(print-banner "")
(print-a-global-malloc global-list)
(print-a-local-malloc local-list)
(init-aux2)
(print "@ End of include file arl2mat2aux.web.")
(close (outchan))
(outchan ())
```

We use yet another file for the inverse Schur code.

```
(outchan (openo "arl2mat3aux.web"))
(print "@ Start of include file arl2mat3aux.web.")
(print-banner "")
(print-schur-inverse)
(print "@ End of include file arl2mat3aux.web.")
(close (outchan))
(outchan ())
```

4.13 Scalar case

4.13.1 Introduction

This is the code used in the scalar case. We generate two sets of functions one for the case with weight, and one for the case without weight. The variable `have-weight` is true when we generate the code for the weight.

This function sets a variable to zero.

```
(defun :kill-var (var prec)
  (if (= prec 0)
      (print var "=0;")
      (print var ".kill();")))
```

This kills a whole array. `i2` is the end test. The array is named `var`, the index is `idx`, and the first value is `i1`.

```
(defun :kill-array (var idx i1 i2 prec)
  (print "for(" idx "=" i1 ";" idx i2 ";" idx "++)")
  (:kill-var (catenate var "[" idx "]") prec))
```

Generic function that calls an operator with three arguments. Uses a table.

```
(defun :gen-op (a b c prec table)
  (let (op)
       (setq op (vref table prec))
       (print c "." op "(" a "," b ");")))
```

Computes $c \mathrel{+}= \overline{a}b$.

```
(defun :add-mult-conj (a b c prec)
   (if (= prec 0)
       (print c " += "  a "*" b ";")
     (:gen-op a b c prec
         #[0 add_mul add_mul_conj  add_mul_conj])))
```

Computes $c \mathrel{-}= \overline{a}b$.

```
(defun :sub-mult-conj (a b c prec)
   (if (= prec 0)
       (print c " -= "  a "*" b ";")
     (:gen-op a b c prec
        #[0  sub_mul sub_mul_conj  sub_mul_conj])))
```

Computes $c \mathrel{-}= ab$.

```
(defun :sub-mult (a b c prec)
   (if (= prec 0)
       (print c " -= "  a "*" b ";")
     (:gen-op a b c prec
        #[0 sub_mul sub_mul sub_mul])))
```

Computes $c += ab$.

```
(defun :add-mult (a b c prec)
   (if (= prec 0)
       (print c " += "  a "*" b ";")
     (:gen-op a b c prec
        #[0 add_mul add_mult add_mul])))
```

Computes $\psi += |var|^2$.

```
(defun :norm2 (var psi prec)
  (cond ((= prec 0)
         (print psi " += " var "*" var ";"))
        ((= prec 2)
         (print psi " += " var ".r* " var ".r + " var ".i*" var ".i;"))
        ((= prec 1)
         (print psi ".add_mul(" var "," var ");"))
        ((= prec 3)
         (print psi ".add_mul(" var ".r," var ".r);")
         (print psi ".add_mul(" var ".i," var ".i);"))))
```

Derivative in reverse mode of the previous code. If we assume $d\psi = 1$, this is just $dv += 2v$. We open code it. In the SLD case, we copy $2v$ in a temporary variable. Note that the name of the variable ψ is unused.

```
(defun :norm2-diff (var prec)
  (cond ((= prec 0)
         (print "d" var  " += 2*" var  ";"))
        ((= prec 2)
         (print "d" var ".r += 2*" var ".r;")
         (print "d" var ".i += 2*" var ".i;"))
        ((= prec 1)
         (print "temp.double_it(" var ");")
         (print "d" var " += temp;"))
        ((= prec 3)
         (print "temp.double_it(" var ".r);")
         (print "d" var ".r += temp;")
         (print "temp.double_it(" var ".i);")
         (print "d" var ".i += temp;"))))
```

Derivative in direct mode. $r += 2\Re(\overline{x}x')$.

```
(defun :cmplx_2times (x xp r prec)
  (cond ((= prec 0)
         (print r "+= 2*" x "*" xp ";"))
        ((= prec 2)
         (print r "+= 2*(" x ".r*" xp
                   ".r +" x ".i*" xp ".i);"))
        ((= prec 1)
         (print r ".two_times(" x "," xp ");"))
        ((= prec 3)
         (print r ".two_times(" x "," xp ");"))))
```

Second derivative. Since the name of ψ is unused, we do not use it. The code is $\delta dx += 2\delta x$.

```
(defun :norm2-delta-diff (var prec)
  (cond ((= prec 0)
         (print "delta_d" var  " += 2*delta_" var  ";"))
        ((= prec 2)
         (print "delta_d" var ".r += 2*delta_" var ".r;")
         (print "delta_d" var ".i += 2*delta_" var ".i;"))
        ((= prec 1)
         (print "temp.double_it(delta_" var ");")
         (print "delta_d" var "+= temp;"))
        ((= prec 3)
         (print "temp.double_it(delta_" var ".r);")
         (print "delta_d" var ".r += temp;")
         (print "temp.double_it(delta_" var ".i);")
         (print "delta_d" var ".i += temp;"))))
```

4.13.2 Basic code

This computes the product of G by $\tilde{q}$. In the case of weight, we have to multiply also by z^d. Since G is constant, the code of the derivative is easy in this case.

```
(defun :multiplication1 (prec)
  (let ((prefix (vref prefixes mode)) varp varq)
    (print "for(j=0;j<=m;j++)")
    (print "for(i=1;i<=n;i++)")
    (setq varq (catenate prefix "q[n-i]"))
    (setq varp (catenate prefix "p[i+j" (if have-weight "+Wdeg" "") "]"))
    (cond ((eq mode 0) (:add-mult-conj varq "g[j]" varp prec))
          ((eq mode 1) (:add-mult-conj varp "g[j]" varq prec))
          ((eq mode 2) (:add-mult-conj varq "g[j]" varp prec))
          ((eq mode 3) (:add-mult-conj varp "g[j]" varq prec)))))
```

We should also consider the case where we multiply g_w by the tilde of qw. This assumes that we have a weight.

```
(defun :multiplication-spec (prec)
  (let ((prefix (vref prefixes mode)) varp varq)
    (print "for(j=0;j<=m;j++)")
    (print "for(i=1;i<=nw;i++)")
    (setq varq (catenate prefix "qw[nw-i]"))
    (setq varp (catenate prefix "p[i+j]"))
    (cond ((eq mode 0) (:add-mult-conj varq "g[j]" varp prec))
          ((eq mode 1) (:add-mult-conj varp "g[j]" varq prec))
          ((eq mode 2) (:add-mult-conj varq "g[j]" varp prec))
          ((eq mode 3) (:add-mult-conj varp "g[j]" varq prec)))))
```

This is now the real code of the multiplication. We do not use the second function, because it is too slow in some cases. We just keep the code somewhere, in case we wanted to implement it.

```
(defun :multiplication (prec)
    (:multiplication1 prec))
(defun :multiplication-unused (prec)
   (if (not have-weight)
       (:multiplication1 prec)
     (print "if(use_fw) {")
     (:multiplication-spec prec)
     (print "} else {")
     (:multiplication1 prec)
     (print "}")))
```

This computes the quotient. In the case of weight, we divide by qw, otherwise by q. The quotient has m terms. The degree of qw is `nw`. Without weight, it is still `nw`, this makes is easier.

```
(defun :division (prec)
   (let (q dq delta_q delta_dq)
     (if (or (= mode 0) (= mode 2))
         (print "for(j=m;j>=0;j--){")
       (print "for(j=0;j<=m;j++){"))
     (if (or (= mode 0) (= mode 2))
         (progn (:division-lc) (:division-other))
         (progn (:division-other) (:division-lc)))
     (print "}")))
```

If `lc` is the coefficieent of P_{i+n+d}, then we have to add to ψ the square of this quantity.

```
(defun :division-lc ()
  (cond ((eq mode 0)
         (print "lc = p[j+nw];")
         (:norm2 "lc" "psi" prec))
        ((eq mode 1)
         (:norm2-diff "p[j+nw]" prec))
        ((eq mode 2)
         (print "lc = p[j+nw];")
         (print "delta_lc = delta_p[j+nw];")
         (:cmplx_2times "lc" "delta_lc" "delta_psi" prec))
        ((eq mode 3)
         (:norm2-delta-diff "p[j+nw]" prec))))
```

This is the real code of the division. The code is: we decrement p_{i+j} by the product of q_i and p_{j+n}. This quantity is in `lc` is the mode is even.

```
(defun :division-other ()
  (if have-weight
      (setq q "qw[i]" dq "dqw[i]" delta_q "delta_qw[i]" delta_dq "delta_dqw[i]")
      (setq q "q[i]" dq "dq[i]" delta_q "delta_q[i]" delta_dq "delta_dq[i]"))
  (print "for(i=0;i<nw;i++){")
  (cond ((eq mode 0)
         (:sub-mult q "lc" "p[i+j]" prec))
        ((eq mode 1)
         (:sub-mult-conj "p[j+nw]" "dp[i+j]" dq prec)
         (:sub-mult-conj q "dp[i+j]" "dp[j+nw]" prec))
        ((eq mode 2)
         (:sub-mult q "delta_lc" "delta_p[i+j]" prec)
         (:sub-mult delta_q "lc" "delta_p[i+j]" prec))
        ((eq mode 3)
         (:sub-mult-conj "p[j+nw]" "delta_dp[j+i]" delta_dq prec)
         (:sub-mult-conj "delta_p[j+nw]" "dp[j+i]" delta_dq prec)
         (:sub-mult-conj q "delta_dp[i+j]" "delta_dp[j+nw]" prec)
         (:sub-mult-conj delta_q "dp[i+j]" "delta_dp[j+nw]" prec)))
  (print "}"))
```

In the case of the weight, we have to compute qw, and the derivatives. We assume that the orthogonal polynomials have been computed. Since one these polynomials is qw, there is no need to compute them. The code is rather easy, since w is constant.

```
(defun :qw-product (prec)
  (print "for(j=0;j<=Wdeg;j++)")
  (print "for(i=0;i<n;i++)")
  (cond ((eq mode 1)
         (:add-mult-conj "w[j]" "dqw[i+j]" "dq[i]" prec))
        ((eq mode 2)
         (:add-mult "w[j]" "delta_q[i]" "delta_qw[i+j]" prec))
        ((eq mode 3)
         (:add-mult-conj "w[j]" "delta_dqw[i+j]" "delta_dq[i]" prec))))
```

4.13.3 The code of the function

This function multiplies G by the constant term of qw. Since this term is one, no multiplication is required, and the derivative of this is zero. We use the same hack as for `:multiplication`, since this is the start of the multiplication.

```
(defun :print-init-p-weight ()
       (print "for(j=0;j<=m;j++)")
       (print "p[j+Wdeg] = g[j];"))
(defun :print-init-p-weight-unused ()
       (print "if(use_fw)")
       (print "for(j=0;j<=m;j++) {")
       (print " p[j] = g[j];")
       (print "} else {")
       (print "for(j=0;j<=m;j++)")
       (print "p[j+Wdeg] = g[j];")
       (print "}"))
```

```
(defun :print-init-p (prec)
  (if have-weight
     (progn
       (:kill-array "p" "j" "0" "<=m+nw" prec)
       (:print-init-p-weight))
    (progn
      (:kill-array "p" "j" "m+1" "<=m+n" prec)
      (print "for(j=0;j<=m;j++)")
      (print "p[j] = g[j];"))))
```

We compute now ψ for each k, i.e. for each component of G. In the case of weight, we have two parts, see equation (2.71.3).

```
(defun :print-main-psi (prec)
    (setq mode 0)
    (print "for(k=0;k<K;k++){")
    (print "p=P[k];")
    (print "g=G[k];")
    (:print-init-p prec)
    (:multiplication prec)
    (:division prec)
    (if have-weight (use-sec "Compute $\\psi_2$"))
    (print "}"))
```

This is the code for the function ψ. We need the following variables. In the case of weight, we also need the degree of the weight, and the degree of qw. For simplicity, we declare `nw` to be n is the case without weight. Note that m should be replaced by `bigm` in case we use g_w. We fear that this m might be a little too big.

```
(defun :declare-scal-var-psi ()
   (show-usage)
   (print "@ @<Declare some local variable@>=")
   (print "int n = a_n;")
;   (print "int m = use_fw ? bigm : a_m;")
   (print "int m = a_m;")
   (print "int K = g_dim1;")
   (print "int i,j,k;")
   (if have-weight
     (progn
       (print "int Wdeg = Wdeg;")
       (print "int nw = a_nw;"))
     (print "int nw = n;")))

(defun :print-scal-var-psi ()
   (print "@<Declare some local variable@>@;")
   (if (or (= prec 1) (= prec 3)) (print "SLD temp;")))
```

We declare now the variables g, q and p, and G, P. In the case of weight, we have to declare qw.

```
(defun :print-mat-var-psi (prec)
  (let ((type (vref types-vector prec)) (ext (vref prec_types prec)))
    (print  type "*q=Y->q;")
    (print  type "**G=use_fw ? Y->Gw : Y->G, *g;")
    (print  type "**P=Y->p, *p;")
    (when have-weight
      (print type "**ps = Y->ps;")
      (print type "*qw = pol_coef" ext "(Y->qw);"))))
```

This defines three global variables. `prev-prec` is the real type associated to the current precision, `type-psi` is the type of ψ and `type-lc` is the type of the leading coefficient (used for the division). It is declared as a pointer in the SLD case.

```
(defun :decl-psi-lc (prec)
    (setq prev-prec (:prev-prec prec))
    (setq type-psi (vref types-vector prev-prec))
    (setq type-lc (vref types-vector prec)))
```

We define now the variables `lc` and `psi`. We initialise ψ to zero.

```
(defun :print-other-var-psi (prec)
  (let (type-psi type-lc prev-prec aux)
    (:decl-psi-lc prec)
    (if (= prev-prec 1)
        (setq aux "; psi.kill();")
      (setq aux " = 0;"))
    (print type-lc " lc;")
    (print type-psi " psi" aux)))
```

We copy ψ in the structure X and Y. The code is a bit easier in the double case.

```
(defun :print-copy-psi (prec)
  (let ((prev (:prev-prec prec)))
    (if (= prev prec)
        (print "Y->psi[0] = psi;")
      (print "Y->psi[0].r = psi;"))
    (if (= prev 1)
        (progn
          (print "temp.get_one();");
          (print "a_psi = psi.to_double();")
          (print "psi.sub(temp,psi);")
          (print "one_minus_psi = psi.to_double();"))
      (print "a_psi = psi;")
      (print "one_minus_psi = 1-psi;"))))
```

We now put everything together. We use the same name, in the case of weight, and in the case without weight, because the function is declared static, and these are put in two different files.

```
(defun :print-scal-psi (prec)
  (show-usage)
  (print "@ @u")
  (print "void arl2_data::scalar_psi" (:w-ext)
          " (arl2_scal" (vref prec_types prec) "*Y)")
  (print "{")
  (:print-scal-var-psi)
  (:print-mat-var-psi prec)
  (:print-other-var-psi prec)
  (:print-main-psi prec)
  (:print-copy-psi prec)
  (print "}"))
```

Some useful function. Note that Y is the data structure that contains everything that depends on the current precision, while X contains everything. The `:w-ext` function adds the extension `weight` to the name of the function, in case it is exported.

```
(defun :fetch-Y ()
  (print "arl2_scal_temp* Y = cur_prec? SQ : SD;"))
(defun :print-cast (prec)
  (let ((ext (vref prec_types prec)))
    (print "arl2_scal" ext "*Y1= static_cast<arl2_scal" ext "*>(Y);")))
(defun :w-ext ()
  (if have-weight "_weight" ""))
```

This is the function that computes ψ. It calls one of the previous ones. The name of the function depends on whether we use weight or not. If we do not use weight, we declare the function with weight.

```
(defun :scalar-psi-def ()
  (show-usage)
  (print "@ @u")
  (print "void arl2_data::scalar_psi" (:w-ext) "()")
  (print "{")
  (:fetch-Y)
  (if have-weight (print "get_bezout(Y,1);"))
  (print "switch(Y->type){")
  (for (i 0 1 3)
       (:print-switch-case i) (print "{")
       (:print-cast i)
       (print "scalar_psi" (:w-ext) "(Y1);")
       (print "break;}"))
  (print "}")
  (print "}"))
```

4.13.4 Code of the derivative

We start with the main function. In principle, it is like the previous one. Note that the first thing to do is to evaluate ψ. This computes also the scalar products in `ps`. After that, we compute the Bezout coefficients. These are needed for ψ_2.

```
(defun :scalar-psi-prime-def()
  (show-usage)
  (print "@ @u")
  (print "void arl2_data::scalar_gradient" (:w-ext)
         "(double * grad, SLD* qqdot)")
  (print "{")
  (print "int N = a_N,i;");
  (:fetch-Y)
  (print "scalar_psi" (:w-ext) "();")
  (if have-weight (print "get_bezout(Y,2);"))
  (print "for(i=0;i<N;i++) grad[i]=0;")
  (print "switch(Y->type){")
  (for (prec 0 1 3)
       (:print-switch-case prec)
       (print "{");
       (:print-cast prec)
       (print "scalar_gradient" (:w-ext) "(grad,Y1);")
       (:psi-prime-end prec)
       (print "break;}"))
  (print "}")
  (print "a_norm = l2_norm(N,grad);")
  (print "}"))
```

At the end, the derivative is in dq. We have to copy it into `grad`. In the multi-precision case, we have also to copy it into `qqdot`. This last copy is done is the main function, so that we do not need to pass `qqdot` as an argument

```
(defun :psi-prime-end (prec)
  (if (or (= prec 1) (= prec 3))
      (progn
        (print "if(qqdot) {")
        (cond ((eq prec 1)
               (print "SLD *dq =  Y1->dq;")
               (print "for(i=0;i<N;i++)")
               (print "qqdot[i] = dq[i];"))
              ((eq prec 3)
               (print "int n = N/2;")
               (print "SLD_complex *dq = Y1->dq;")
               (print "for(i=0;i<n;i++){")
               (print "  qqdot[2*i] = dq[i].r;")
               (print "  qqdot[2*i+1] = dq[i].i;")
               (print "}")))
        (print "}"))))
```

The first thing to do is to get all the variables. In the case of weight, q is not needed. However, the polynomial q is needed for ψ_2.

```
(defun :print-mat-var-psi-prime (prec)
  (let ((type (vref types-vector prec)) (ext (vref prec_types prec)))
    (if have-weight
       (progn
         (print type "*w, *qw, *dq, *g, *p, *dp, *dqw;")
         (print "Polynom* table = Y->table;"))
       (print type "*q, *dq, *g, *p, *dp;"))
    (if (not have-weight)
        (print "q=Y->q;"))
    (when have-weight
          (print type "**ps = Y->ps;")
          (print type "**dps = Y->dps;")
          (print "int N = a_N;");
          (print "Polynom pol_q = Y->pol_q;")
          (print "qw = pol_coef" ext "(Y->qw);")
          (print "w = pol_coef" ext "(Y->a_w);")
          (print "dqw = Y->dqw;"))
    (print "dq = Y->dq;")))
```

We start by initialising the derivative of q and qw to zero.

```
(defun :kill-dq-dqw (prec)
  (:kill-array "dq" "i" 0 "<n" prec)
  (when have-weight (:kill-array "dqw" "i" 0 "<nw" prec)))
```

This is the start of the loop over k. We get g, p and dp, and clear the dp array.

```
(defun :fetch-par-kill-p (prec)
    (print "g=Y->G[k];")
    (print "p=Y->p[k];")
    (print "dp=Y->dp[k];")
    (:kill-array "dp" "i" 0 "<=m+nw" prec))
```

This copies `var` into `grad`. Here `var` can be a complex variable, but `grad` is always real.

```
(defun :copy-to-grad (var prec)
  (let ((need-brace false))
    (if (>= prec 2) (setq need-brace true))
    (print "for(i=0;i<n;i++)")
    (if need-brace (print "{"))
    (cond ((eq prec 0)
           (print "grad[i]=" var "[i];"))
          ((eq prec 1)                        ; SLD
           (print "grad[i] = " var "[i].to_double();"))
          ((eq prec 2)
           (print "grad[2*i]=" var "[i].r;")
           (print "grad[2*i+1]=" var "[i].i;"))
          ((eq prec 3)
           (print "grad[2*i] = " var "[i].r.to_double();")
           (print "grad[2*i+1] = " var "[i].i.to_double();")))
    (if need-brace (print "}"))))
```

Does the same operation the other way. Used for the Hessian. The value of `grad` is the direction in which we compute the Hessian.

```
(defun :copy-from-grad (var prec)
  (let ((need-brace false))
    (if (>= prec 2) (setq need-brace true))
    (print "for(i=0;i<n;i++)")
    (if need-brace (print "{"))
    (cond ((eq prec 0)
           (print  var "[i]=grad[i];"))
          ((eq prec 1)                        ; SLD
           (print  var "[i]=grad[i];"))
          ((eq prec 2)
           (print  var "[i].r =grad[2*i];")
           (print  var "[i].i =grad[2*i+1];"))
          ((eq prec 3)                        ; SLD complex
           (print var "[i].r = grad[2*i];")
           (print var "[i].i = grad[2*i+1];")))
    (if need-brace (print "}"))))
```

This is now the function.

```
(defun :print-real-psi-prime (prec)
  (show-usage)
  (print "@ @u")
  (print "void arl2_data::scalar_gradient" (:w-ext)
         " (double* grad, arl2_scal" (vref prec_types prec) "*Y)")
  (print "{")
  (:print-scal-var-psi)
  (:print-mat-var-psi-prime prec)
  (:kill-dq-dqw prec)
  (print "for(k=0;k<K;k++){")
  (:fetch-par-kill-p prec)
  (:division prec)
  (:multiplication prec)
  (print "}")
  (if have-weight (:qw-product prec))
  (if have-weight (use-sec "Compute $\\psi_2$"))
  (:copy-to-grad "dq" prec)
  (print "}"))
```

This piece of code fills `delta_q` by putting 1 in the i-th component and zero everywhere else. This is used for computing the full Hessian.

```
(defun :fill-delta-q (prec)
   (decl-sec0 "Fill |delta_q| for the Hessian" prec)
   (:kill-array "delta_q" "j" 0 "<n" prec)
   (cond ((eq prec 0)
          (print "delta_q[i] = 1;"))
         ((eq prec 1)
          (print "delta_q[i].get_one();"))
         ((eq prec 2)
          (print "j = i/2;")
          (print "if(i&1) delta_q[j].i = 1;")
          (print "else delta_q[j].r = 1;"))
         ((eq prec 3)
          (print "j = i/2;")
          (print "if(i&1) delta_q[j].i.get_one();")
          (print "else delta_q[j].r.get_one();"))))
```

4.13.5 Derivative in direct mode

We fetch here the variables that are are needed for both parts.

```
(defun :vars-for-hess (prec)
  (let ((ext (vref prec_types prec)))
    (print "q = Y->q;")
    (print "delta_q = Y->delta_q;")
    (print "delta_dq = Y->delta_dq;")
    (when have-weight
       (print "qw = pol_coef" ext "(Y->qw);")
       (print "w = pol_coef" ext "(Y->a_w);")
       (print "delta_qw = Y->delta_qw;")
       (print "delta_dqw = Y->delta_dqw;"))))
```

We fetch the variables p, δp and g, and kill δp.

```
(defun :get-p-prime-direct (prec)
  (print "p = Y->p[k];")
  (print "delta_p = Y->delta_p[k];")
  (print "g = Y->G[k];")
  (:kill-array "delta_p" "j" 0 "<=m+nw" prec))
```

This is now the complete code of the first part of the Hessian.

```
(defun :scalar-psi-prime-direct (prec)
  (let ((type (vref types-vector prec)) (ext (vref prec_types prec)))
    (setq mode 2)
    (decl-sec  "Compute $\\psi$")
    (:vars-for-hess prec)
    (when have-weight
      (:kill-array "delta_qw" "i" 0 "<nw" prec)
      (:qw-product prec))
    (print "for(k=0;k<K;k++){")
    (:get-p-prime-direct prec)
    (:multiplication prec)
    (:division prec)
    (print "}")))
```

4.13.6 Hessian

This clears the arrays δdqw and δdq.

```
(defun :kill-delta-qw (prec)
    (:kill-array "delta_dq" "i" 0 "<n" prec)
    (if have-weight  (:kill-array "delta_dqw" "i" 0 "<nw" prec)))
```

Fetch the variables, kill δdp.

```
(defun :get-p-hessian (prec)
  (print "p = Y->p[k];")
  (print "dp = Y->dp[k];")
  (print "delta_p = Y->delta_p[k];")
  (print "delta_dp = Y->delta_dp[k];")
  (print "g = Y->G[k];")
  (:kill-array "delta_dp" "j" 0 "<=m+nw" prec))
```

This computes now the second part of the Hessian.

```
(defun :scalar-psi-hess (prec)
  (setq mode 3)
  (let ((type (vref types-vector prec)) (ext (vref prec_types prec)))
    (decl-sec  "Compute $\\psi$")
    (:kill-delta-qw prec)
    (print "for(k=0;k<K;k++){")
    (:get-p-hessian prec)
    (:division prec)
    (:multiplication prec)
    (if have-weight (use-sec "Compute $\\psi_2$"))
    (print "}")
    (if have-weight (:qw-product prec))))
```

This function computes one row of the Hessian.

```
(defun :def-scalar-hess (prec)
  (let ((type (vref types-vector prec)) (ext (vref prec_types prec))
        prev-prec type-psi type-lc aux)
    (show-usage)
    (print "@ @u")
    (print "void arl2_data::scalar_hess"
           "(arl2_scal" (vref prec_types prec) "*Y, double* grad"
           (if have-weight ",int l" "") ")")
    (print "{")
    (:scal-hess-prepare prec)
    (setq mode 2)
    (use-sec  "Compute $\\psi$")
    (setq mode 3)
    (use-sec  "Compute $\\psi$")
    (:copy-to-grad "delta_dq" prec)
    (print "}")))
```

This initialises the variables for the Hessian.

```
(defun :scal-hess-prepare (prec)
    (:print-scal-var-psi)
    (print type "*q, *delta_q, *delta_dq, *g;")
    (print type "*p, *dp,*delta_p, *delta_dp;")
    (when have-weight
          (print type "**ps = Y->ps;")
          (print type "**dps = Y->dps;")
          (print type "**ddps = Y->ddps;")
          (print "int N = a_N;");
      (print type "*w, *qw, *delta_qw, *delta_dqw;"))
    (:decl-psi-lc prec)
    (print type-lc " lc;")
    (print type-lc " delta_lc;")
    (if (= prev-prec 1)
        (setq aux "; delta_psi.kill();")
      (setq aux " = 0;"))
    (print type-psi " delta_psi" aux))
```

This computes the full Hessian if the flag is non-zero, the Hessian applied to a vector in the other case. The vector is in `grad`. The result also. In the case of a weight, we have to compute the full Hessian (perhaps we could do better). In the case where we have to compute the Hessian applied to a vector, we compute the whole Hessian in temp storage and then, multiply. We could do better than allocate memory each time, but this code is still unused.

```
(defun :def-scalar-hessian (prec)
 (let ((ex (vref prec_types prec)))
    (show-usage)
    (print "@ @u")
    (print "void arl2_data::scalar_hessian" (:w-ext)
       "(arl2_scal" ex "*Y, double* grad, int flag)")
    (print "{")
    (print "int n=a_n, N=a_N,i,j;")
    (print (vref types-vector prec) "*delta_q = Y->delta_q;")
    (unless have-weight
      (print "if(flag==0) {")
      (:copy-from-grad "delta_q" prec)
      (print "scalar_hess(Y,grad);")
      (print "return;")
      (print "}"))
    (when have-weight
        (print "double * temp_grad;")
        (print "double * aux=NULL,*aux1=NULL;")
        (print "if(flag==0) {")
        (print " temp_grad= (double*) GC_malloc_atomic(N*N*sizeof(double));")
        (print "aux1 = temp_grad;")
        (print " aux = (double*) GC_malloc_atomic(N*sizeof(double));")
        (print "} else temp_grad = grad;"))
    (print "for(i=0;i<N;i++){")
    (use-sec0 "Fill |delta_q| for the Hessian" prec)
    (if have-weight
        (print "scalar_hess(Y,temp_grad,i);")
      (print "scalar_hess(Y,grad);"))
    (if have-weight
       (print "temp_grad += N;")
       (print "grad += N;"))
    (print "}")
    (when have-weight
      (print "if(flag) return;")
      (print "for(i=0;i<N;i++)")
      (print " { aux[i] = grad[i]; grad[i] = 0; }")
      (print "temp_grad = aux1;")
      (print "for(i=0;i<N;i++)")
      (print "for(j=0;j<N;j++)")
      (print "grad[i]+= temp_grad[N*i+j]* aux[j];"))
    (print "}")))
```

This is now the main function. It is assumed that a call to the gradient is made before we compute the Hessian.

```
(defun :the-scalar-hessian ()
  (show-usage)
  (print "@ @u")
  (if have-weight
      (print "void arl2_data::scalar_hessian_weight(double* grad,int flag)")
    (print "void arl2_data::scalar_hessian(double* grad,int flag)"))
 (print "{")
 (:fetch-Y)
 (if have-weight (print "get_bezout(Y,3);"))
 (if have-weight (print "hess_of_scal(Y);"))
 (print "switch(Y->type){")
  (for (i 0 1 3)
       (:print-switch-case i)
       (print "{")
       (:print-cast i)
       (print "scalar_hessian" (:w-ext) "(Y1,grad,flag);")
       (print "break;}"))
 (print "}}"))
```

4.13.7 Second part of ψ

We have $\psi_2 = \sum |a_{kj}|^2$ where

$$a_{kj} = \langle F/we_k \,|\, \Phi_j/qw \rangle$$

The sum is over all k, j and e_k is the k-th base vector. Every piece of code that computes ψ_2 is in a loop over k. The loop over j is from n to $n+d-1$, where d is the degree of the weight.

The scalar product $\langle F/w \,|\, \Phi/x \rangle$ is computed by a call to `my_long_div`, that takes 3 arguments: the polynomial Φ, the data structure Y and the index k. We may (or may not) use the Bezout relation. It is assumed that the denominator qw, and the Bezout coefficient are stored somewhere in Y.

```
(defun :get-psi2-direct (prec)
   (let (aux (ext (vref prec_types prec)))
     (setq aux (catenate "ld_res->" ext))
     (cond ((eq prec 0) (setq aux "ld_res->C.r"))
           ((eq prec 1) (setq aux "ld_res->SC.r")))
     (setq mode 0)
    (decl-sec "Compute $\\psi_2$")
    (print "{")
    (print "for(j=0;j<Wdeg;j++){ ")
    (print "my_long_div(Y->orth_poly[0][0][j],Y,k);")
    (:norm2 aux "psi" prec)
    (print "ps[k][j] = " aux ";")
    (print "}}")))
```

This computes now the derivative. We can safely put the result into dq. This piece of code is (mostly) independent of the precision. The computation is done in direct mode! This has as side effect that we have to loop over i. We have to put the result into the real or imaginary part of dq. This makes the code a little complicated. Moreover, we have to compute ∂q, the direction in which to evaluate the derivative. This is done by the function `mk_zn`, which puts the result in `table` at location 5. Note that we compute the derivative of the scalar product of f and Φ/qw. This is the scalar product of f and the derivative of Φ/qw. This is $(\partial\Phi q - \Phi\partial q)/q^2w$. We have to compute the numerator of this. It is assumed that the denominator q^2w is somewhere in the `table`. The result is stored in `dps`. Recall that if a is the scalar product, then we have to add a^2 to ψ, hence $2aa'$ to ψ', and $2(aa'' + a'^2)$ to ψ''.

```
(defun :get-psi2-diff (prec)
   (let (aux (ext (vref prec_types prec)))
     (setq aux (catenate "ld_res->" ext))
     (cond ((eq prec 0) (setq aux "ld_res->C.r"))
           ((eq prec 1) (setq aux "ld_res->SC.r")))
    (setq mode 1)
    (decl-sec "Compute $\\psi_2$")
    (print "{")
    (if (or (= prec 2) (= prec 3)) (print "int I,ev;"))
    (print "for(i=0;i<N;i++){ ")
    (if (or (= prec 2) (= prec 3))
       (print "I=i/2; ev = i&1;"))
    (print "pol_deg(table[5]) = -1;")
    (if (or (= prec 2) (= prec 3))
       (print "table[5]->change_deg(I);")
       (print "table[5]->change_deg(i);"))
    (cond ((eq prec 0) (print "pol_coefD(table[5])[i] = 1.0;"))
          ((eq prec 1) (print "pol_coefS(table[5])[i].get_one();"))
          ((eq prec 2)
             (print "if(ev) pol_coefC(table[5])[I].i= 1;")
             (print "    pol_coefC(table[5])[I].r= 1;"))
          ((eq prec 3)
           (print "if(ev) pol_coefSC(table[5])[I].i.get_one();")
             (print "    pol_coefSC(table[5])[I].r.get_one();")) )
    (print "mk_zn(i,5,table);")
    (print "for(j=0;j<Wdeg;j++){ ")
    (print "table[7]->mul(Y->orth_poly[i+1][0][j],pol_q);")
    (print "table[7]->sub_mul(Y->orth_poly[0][0][j],table[5]);")
    (print "for(k=0;k<K;k++){")
    (print "my_long_div(table[7],Y,k);")
    (print "dps[k][j*N+i] = " aux ";")
    (if (or (= prec 2) (= prec 3))
       (progn
          (print "if(ev){")
          (:cmplx_2times "ps[k][j]" aux "dq[I].i" prec)
          (print "} else { ")
          (:cmplx_2times "ps[k][j]" aux "dq[I].r" prec)
          (print "}"))
      (:cmplx_2times "ps[k][j]" aux "dq[i]" prec))
    (print "}}}}")))
```

Let's now compute the Hessian. The easiest thing to do is to compute the full Hessian of the scalar product, and then to multiply by whatever is needed. This is now the function that computes it. We have to compute the scalar product of f/w and x/q^3w where

$$x = \Phi(-q_{12}q + 2q_1q_2) - \Phi_1q_2q - \Phi_2q_1q + \Phi_{12}q^2$$

where x_i is the derivative of x in one direction. This piece of code computes the polynomial x. We compute the derivative in the directions i and l. Note that $q_{12} = 0$. It is assumed that $q^3 w$ is somewhere in the table.

```
(defun :compute-one-hess-elt ()
  (print "for(i=0;i<N;i++){ ")
  (print "  mk_zn(i,5,table);")
  (print "  for(l=0;l<=i;l++){ ")
  (print "      mk_zn(l,6,table);")
  (print "      table[9]->mul(table[5],table[6]);")
  (print "      table[9]->mul(table[9],2);")
  (print "        for(j=0;j<Wdeg;j++){ ")
  (print "table[7]->mul(table[9],Y->orth_poly[0][0][j]);")
  (print "table[8]->mul(table[5],Y->orth_poly[l+1][0][j]);")
  (print "table[8]->add_mul(table[6],Y->orth_poly[i+1][0][j]);")
  (print "table[7]->sub_mul(table[8],pol_q);")
  (print "table[7]->add_mul(table[10],Y->orth_poly[i+1][l+1][j]);"))
```

We compute now the scalar products, and put them in a big table. This function is independent of the type. The type is needed when we do the copy, but we know the type, since it is in the result of the long division. Since the Hessian is symmetric, we copy A_{ij} into A_{ji}.

```
(defun :compute-more-hess-elts ()
    (print "table[10]->mul(pol_q,pol_q);")
    (:compute-one-hess-elt)
    (print "for(k=0;k<K;k++){")
    (print "my_long_div(table[7],Y,k);")
    (print "my_copy_ld(ddps,k,l+N*(j*N+i));")
    (print "if(i!=l) my_copy_ld(ddps,k,i+N*(j*N+l));")
    (print "}}}}"))
```

We add now the declarations.

```
(defun :hess-of-scal ()
    (print "@ @u void arl2_data::hess_of_scal"
         " (arl2_scal_temp*Y)")
    (print "{")
    (print "int N=a_N, i,j,k,l;")
    (print "int K = g_dim1;")
    (print "void* ddps = Y->get_ddps();")
    (print "Polynom* table = Y->table;")
    (print "Polynom pol_q = Y->pol_q;")
    (print "orth_poly(1);")
    (:compute-more-hess-elts)
    (print "}"))
```

This computes now the Hessian. In fact, we compute the Hessian in a given direction. It is assumed that the direction is `delta_q`. This direction is the l-th partial derivative of q. The result should be added to `delta_dq`. Recall that a_{ij} is the scalar product. Denote this by a. We have $\psi = |a|^2$. This gives $\psi_{12} = 2(aa_{12} + a_1 a_2)$.

```
(defun :get-psi2-hess (prec)
  (let ((dest))
    (setq mode 3)
    (decl-sec "Compute $\\psi_2$")
    (print "{")
    (if (or (= prec 2) (= prec 3)) (print "int I,ev;"))
    (print "for(i=0;i<N;i++) {")
    (if (or (= prec 2) (= prec 3))
        (print "I=i/2; ev = i&1;"))
    (print "for(j=0;j<Wdeg;j++){")
    (if (or (= prec 2) (= prec 3))
       (progn
          (print "if(ev){")
          (if (= prec 2)
              (:cpl-aux "delta_dq[I].i")
            (:cpl-aux "delta_dq[I].i"))
          (print "} else { ")
          (if (= prec 2)
              (:cpl-aux "delta_dq[I].r")
            (:cpl-aux "delta_dq[I].r"))
          (print "}"))
      (:cpl-aux "delta_dq[i]"))
    (print "}}}")))


(defun :cpl-aux (r)
    (:cmplx_2times "ps[k][j]" "ddps[k][l+N*(j*N+i)]" r prec)
    (:cmplx_2times "dps[k][j*N+i]" "dps[k][j*N+l]"  r prec))
```

This is now the main function of this file.

```
(defun main-code ()
  (for (i 0 1 3) (:print-scal-psi i))
  (:scalar-psi-def)
  (:declare-scal-var-psi)
  (print "@* First derivative.")
  (setq mode 1)
  (for (i 0 1 3) (:print-real-psi-prime i))
  (:scalar-psi-prime-def)
  (print "@* Second derivative.")
  (setq mode 2)
  (for (i 0 1 3) (:scalar-psi-prime-direct i))
  (for (i 0 1 3) (:scalar-psi-hess i))
  (for (i 0 1 3) (:fill-delta-q i))
  (for (i 0 1 3) (:def-scalar-hess i))
  (for (i 0 1 3) (:def-scalar-hessian i))
  (when have-weight
     (:hess-of-scal)
     (for (i 0 1 3) (:get-psi2-direct i))
     (for (i 0 1 3) (:get-psi2-diff i))
     (for (i 0 1 3) (:get-psi2-hess i))
  )
  (:the-scalar-hessian))

```

This piece of code generates everything in the case without weight.

```
(setq have-weight ())
(outchan (openo "arl3aux.web"))
(rmargin 100)
(print "@* The function psi.")
(print "@ Start of include file arl3aux.web")
(print-banner "") (print "Copyright 1998-1999 INRIA/MIAOU")
(print "The source was make\\_scalar.ll.")
(main-code)
(print "@ End of include file arl3aux.web.")
```

```
(close (outchan))
(outchan ())

```

This one generates everything in the case of weight.

```
(setq have-weight true)
(outchan (openo "arl3w1.web"))
(rmargin 100)
(print "@* The function psi.")
(print "@ Start of include file arl3auxw.web")
(print-banner "") (print "Copyright 1998-1999 INRIA/MIAOU")
(print "The source was make\\_scalar.ll.")
(main-code)
(print "@ End of include file arl3auxw.web.")
(close (outchan))
(outchan ())
```

Chapter 5

Complexity

5.1 Scalar case of dimension one

Recall that, if $q = z - \alpha$, then $\psi(q) = \|g\|^2 - (1 - |\alpha|^2)|g(\alpha)|^2$. We can always assume that $\|g\| = 1$. In this section, we shall write $\psi(\alpha)$ instead of $\psi(q)$.

5.1.1 Real case of dimension one

In the real case, we can forget the absolute values, so that we have

$$\psi(x) = 1 - (1 - x^2)g(x)^2. \tag{5.1}$$

We want to find the minimum of ψ for $-1 \leq x \leq 1$. We know that $\psi(\pm 1) = 1$, and that $0 \leq \psi(x) \leq 1$. Thus, ψ has at least one local minimum. In fact, if ψ has n local minima, and $m + 2$ local maxima (including ± 1), then $n = m + 1$.

Note that a zero of g is a maximum of ψ, so that, if g is of degree M, ψ could have more than M local minima. Assume that we approximate a fraction p/q. Then g is roughly $\tilde{p}/\tilde{q}$. If equality holds, then $\psi' = 0$ if $\tilde{p} = 0$ or $x\tilde{p}\tilde{q} = (1 - x^2)(\tilde{p}\tilde{q}' - \tilde{p}'\tilde{q})$. This shows that ψ cannot have more than $3n/2$ minima, which is much better.

Note that our algorithm will fail in the case where $\psi' = \psi'' = 0$. We have $\psi' = 0$ if $g(x) = 0$ or $xg = (1 - x^2)g'$. The additional condition is

$$-2g^2g'^2 + g^3g'' - 9g'^4 + 6g'^2gg'' - g^2g''^2 = 0.$$

We give here the algorithm that is used to find all local minima of ψ.

Step one. The first three derivatives of ψ are given by:

$$\psi' = 2g(xg - (1 - x^2)g') \tag{5.2.a}$$

$$\psi'' = 2g^2 + 8xgg' - 2(1 - x^2)(g'^2 + gg'') \tag{5.2.b}$$

$$\psi''' = 12(gg' + xg'^2 + xgg'') - 2(1 - x^2)(3g'g'' + gg'''). \tag{5.2.c}$$

Let I_k be the interval $[-1 + k/20, -1 + (k + 1)/20]$, $0 \leq k < 40$. The interval $I = [-1, 1]$ is hence split into 40 smaller intervals. On each of them, we compute the maximum of the absolute value of ψ'' and ψ''' by evaluating these quantities on one thousand points. The main assumption is that the number of evaluation points is large enough.

Step two. If f is any function, we know that, for each interval $[a,b]$, there exists a point ξ in the interval such that

$$\frac{f(a)-f(b)}{a-b} = f'(\xi).$$

Now, if M is the maximum of $|f'|$ on $[a,b]$, and if $\epsilon = |f(a)|/M$, then f does not change sign on the interval $[a,b]$ in case $b = a+\epsilon$. We give here a procedure that, given a, returns ϵ, such that f is of constant sign on $[a,a+\epsilon]$. The trouble is, of course, that M depends on ϵ, and ϵ is defined as a function of M. We assume here that Z is a table, such that Z_k is the maximum of $|f|$ on I_k.

1. Let $i=0$, $z_{i-1}=0$, and k be such that $a \in I_k$.
2. Let z_i be the max of z_{i-1} and Z_{k+i} (loop invariant: it is the max of f' on $I_k, \dots, I_{k+i}$).
3. Let $\epsilon = |f(a)|/z_i$, and j such that $a+\epsilon$ is in I_{k+j}.
4. If $j \le i$, we have found ϵ. If $a+|\epsilon| > 1$, we take $\epsilon = 1-a$ (so that $a+\epsilon = 1$). Otherwise, we increment i and go back to point 2.

Step three. Given a, we compute here b such that ψ has zero or one local minimum in the interval $[a,b]$. We compute first ϵ and ϵ_1 such that ψ' does not change sign on $[a,a+\epsilon]$ and ψ'' does not change sign on $[a,a+\epsilon_1]$. Let $b = a+\max(\epsilon,\epsilon_1)$.

- If $\epsilon > \epsilon_1$, there is no root of ψ' on $[a,b]$, because ψ' has constant sign on $[a,b]$.
- Assume now $\epsilon < \epsilon_1$. The assumption is now that ψ'' has constant sign on $[a,b]$. If ψ' and ψ'' have the same signs at a, then there is no root of ψ' on $[a,b]$ (because either ψ' is positive and increasing, or negative and decreasing).
- Now, ψ' may change sign, but only once, in the interval $[a,b]$. In the case $\psi'(a) < 0$ and $\psi'(b) > 0$, then we have a local minimum in the interval.
- Otherwise, there is no local minimum in the interval.

Step four. We have now a procedure that given a, returns b such that the interval $[a,b]$ contains a single local minimum of ψ, or none. In the first case, we compute it, in the second case we do nothing. After that, we replace a by b, and restart with this new value of a, until we get $a=1$.

The trouble is that the quantity $b-a$ may be too small. If this is the case (i.e. if $b-a<\epsilon_0$, for some fixed ϵ_0), we replace b by $a+\epsilon_0$, and decide that we have a local minimum, if and only if $\psi'(a) < 0$ and $\psi'(b) > 0$.

Step five. In the case where we know that there is a root between a and b, we try the Newton method to find it. There is however no warranty that this will succeed and that the result is between a and b. For this reason, we try three starting points, namely a, b and $(a+b)/2$.

5.1.2 Complex case of dimension one

If we write the equation $\psi' = 0$, we get either $g(\alpha) = 0$ or

$$\overline{\alpha} g(\alpha) - (1-|\alpha|)^2 g'(\alpha) = 0. \tag{5.3}$$

In the first case, α is a maximum of ψ. Hence, one way to minimise ψ is solve equation (5.3), evaluate the second derivative of ψ, and check whether the Hessian is positive definite or not.

There is a way to find the roots of a function $h(x,y)$, called *exclusion method*, see for instance [5]. The method works in the case where h has more than two arguments (in the case of 3, replace “squares” by “cubes” and “four sub-squares” by “eight sub-cubes”). Let’s write $z = x+iy$, so that h becomes a

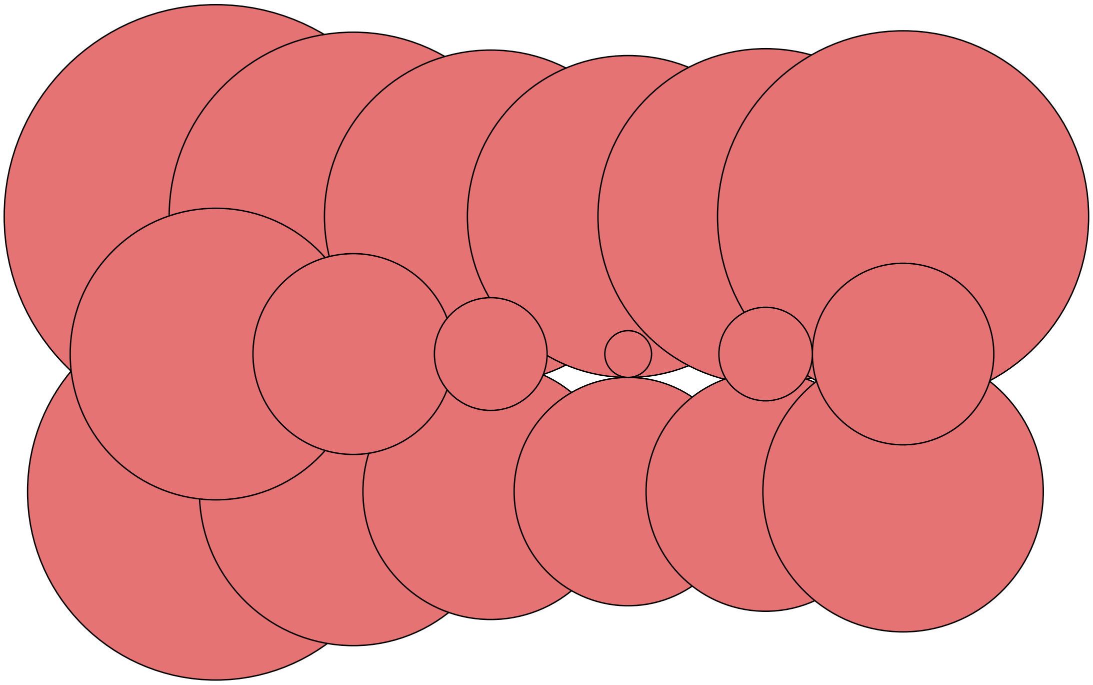

Figure 5.1: Exclusion circles, showing the minimum of ψ, which is in one of the two white zones. It is to be noticed that the size of the circle is not a function of the distance to the minimum: we have chosen the centre on three horizontal lines. On the first and the last, the radius is rather constant, on the middle line, the radius varies a lot.

real function of the complex variable z. Let Z be the set of all z such that $h(z) = 0$, this is the set of zeroes of h. The idea is to take a point z_0, evaluate h at z_0, and if this is not zero, find ϵ such that $z \notin Z$ if $|z - z_0| \leq \epsilon$. Hence, we can exclude zones in which the function does not vanish. The function $z_0 \to \epsilon$ is called the exclusion function. On figure 5.1 we have shown a certain number of disks $|z - z_0| \leq \epsilon$ near a zero of h.

There are different ways of using the exclusion function. One way is the following. We consider a square that covers the domain of interest D. After a while, we get a certain number of squares S_i, such that, if $z \in Z$, then x is in at least one square. For each square S_i, we evaluate $\epsilon(x)$ at the centre of the square. In the good case, we can eliminate this square. In the bad case, we split the square into four sub-squares. Some of these sub-squares may be outside D, hence can also be eliminated. After a while, if the function $\epsilon(z)$ satisfies good properties, we can "see" the set of zeroes of h. The case of interest is when h has only isolated zeroes (for instance if h is a polynomial). The hope is that, after a while, we have as many small squares as roots of h, and a Newton method, with the centre of the square as initial condition, will find the root. In practice (see figure 5.1), we get much more squares than roots of h, and this method is not used to find the minimum of ψ.

Let $\eta(z_0)$ be the distance of z_0 to the set of roots of h. Then $0 \leq \epsilon(z)/\eta(z) \leq 1$. One property of ϵ we need is that the limit $\epsilon(z)/\eta(z)$ is not zero if z approaches Z. Assume that h is a polynomial function of z. Then Z contains a finite number of points z_i $(1 \leq i \leq n)$, and the condition z approaches Z can be written as: there is some i such that $\lim z = z_i$, and in this case $\eta(z) = |z - z_i|$.

Fix z_0. Write

$$h(z+z_0)=\sum_{k=0}^{n} h_k z^k. \tag{5.4.a}$$

Let $a_i=|h_i|$ and

$$f(x)=-a_0+\sum_{i=1}^{n} a_i x^i. \tag{5.4.b}$$

Because $a_i\geq 0$, there is a unique $\alpha\geq 0$ such that $f(\alpha)=0$. Moreover, if $x>\alpha$ then $f(x)>0$, and if $0<x<\alpha$, $f(x)<0$. It is very easy to find x such that $x>\alpha$: we start with $x=1$, and multiply x by 2 while $f(x)<0$. Moreover, if $x_0>0$, then the sequence $x_{k+1}=x_k-f(x_k)/f'(x_k)$ converges to α, and satisfies $\alpha<x_k$. Thus, it is easy to find x such that $0.99x<\alpha<x$ (in case $f(0.99x_k)<0$, then $x=x_k$ is a good value). Hence, we obtain without pain a good estimation of α (with a precision of 1%).

If we compare (5.4.a) and (5.4.b), we see that α (or anything smaller than α) is a candidate for $\epsilon(z_0)$. Using the Taylor expansion, one can show that, if z_0 is a root of multiplicity i of h, then

$$\lim\frac{\alpha}{|z-z_0|}=2^{1/i}-1.$$

In particular, this limit is one in case of a single root, $\sqrt{2}-1$ in case of a double root. Thus, if h has only single roots, our function ϵ is optimal.

In the special case where the polynomial is z^2-1, and $z_0=a+ib$, we have

$$f(x)=-|1-(a+ib)|^2+2x|a+ib|+x^2=0$$

hence

$$\epsilon=-\sqrt{a^2+b^2}+\sqrt{a^2+b^2+\sqrt{(1-a^2+b^2)^2+4a^2b^2}}.$$

Note that, if $b=0$ and $-1\leq a\leq 1$, then ϵ is the largest possible: the circle with centre z_0 and radius ϵ passes through one of the roots of the polynomial. We have shown in figure 5.2 some circles that show the root $z=1$.

We want to apply this technique in the case

$$h(a)=\overline{a}g(a)-(1-|a|^2)g'(a).$$

Instead of (5.4.a), we have now $h(z_0+z)=\sum_i(a_iz^i+b_i\overline{z}z^{i-1})$, from which we get a bound

$$|h(z_0+z)-h(z_0)|\leq\sum(|a_i|+|b_i|)|z|^i.$$

Hence we get a function f like before. Note however that h is not a polynomial. In particular, the set of zeroes of h can contain an infinite number of points (see example from chapter 2: $\psi(a)=1-|a|^2+|a|^4$).

5.2 Scalar case

As explained elsewhere, we find a minimum of ψ using integration or a Quasi-Newton algorithm. The number of times ψ' and ψ'' are computed depends a lot on the initial condition. In general, we use an initial condition that has a pole on $\mathbb{T}$, thus, is generally far from the minimum. In some other cases, we may have a good guess of where the minimum lies.

From now on, we compute the complexity of the computation of ψ, ψ' and ψ''. What we count is the number of multiplications. We assume that a division is equivalent to a multiplication. We assume that the cost of a complex multiplication or division is four times the cost of a real multiplication.

In this section, we look at the scalar case. The quantity n will be the degree of q. In the real case, we have

$$q=z^n+\sum_{i=0}^{n-1} q_i z^i$$

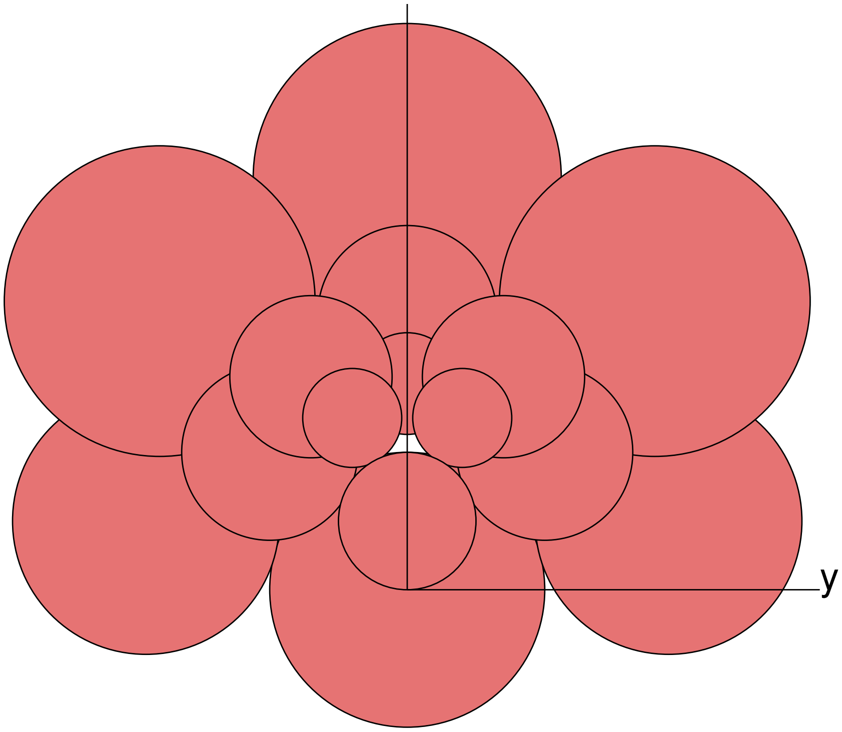

Figure 5.2: Exclusion circles for $z^2 - 1$, near the root $z = 1$.

so that ψ is a function of n arguments. Of course, ψ depends on some other quantities (the function f, and maybe the weight), so that the complexity will depend on some other quantities. The important point is that we have to compute n partial derivatives.

In the complex case, we write

$$q = z^n + \sum_{k=0}^{n-1} (q_k + ir_k) z^k$$

so that ψ will be a function of $2n$ real variables. Note that ψ is a C^∞ function of q_k and r_k, but is not an analytic function of $q_k + ir_k$.

If we compute the derivatives of ψ in direct mode, we have twice as many partial derivatives to compute in the complex case (n in the real case, and $2n$ in the complex case). Since the Hessian is a symmetric matrix, we have to compute $n(n+1)/2$ partial derivatives for ψ'' in the real case, and $n(2n+1)$ partial derivatives in the complex case. Hence, the complex code of ψ'' is a priori 16 times slower than the real code. The quantity d will be the degree of the weight. The quantity m will be the number of terms in G. Recall that G is an $M \times p$ matrix, and in the scalar case $p = 1$. Each element of G is a polynomial. We assume that all elements have the same degree (the programs adds zeroes if needed), say M_1. Then $m = M(M_1 + 1)$. In general, we have $M = 1$, and in most cases, the complexity depends only on m. If this is not the case, we shall indicate what happens if $M \neq 1$. For the numerical applications, we always assume $m = 400$. We shall consider four values of n, namely 5, 8, 10 and 20. In the case of weight, the weight will be 1, 2, 5, and 10.

5.2.1 Orthogonal polynomials

If $P = \Phi_n$ and $Q = \Phi_{n-1}$ the formula for orthogonal polynomials is

$$\tilde{Q}(0)\tilde{Q} = \tilde{P}(0)\tilde{P} - \overline{P(0)}P.$$

Recall that $\tilde{Q}(0)$ is the leading coefficient of Q, and this is a real number. Assume $P = \sum p_i z^i$ and $Q = \sum q_i z^i$. Then

$$q_{n-1} q_j = p_n p_{j+1} - p_0 \overline{p}_{n-j-1}, \tag{5.5}$$

and for $j = n-1$ we get

$$q_{n-1}^2 = p_n^2 - |p_0|^2. \tag{5.6}$$

If we differentiate these formulas, we get

$$q'_{n-1} q_j + q_{n-1} q'_j = p'_n p_{j+1} + p_n p'_{j+1} - p'_0 \overline{p}_{n-1-j} - p_0 \overline{p}'_{n-j-1}, \tag{5.7}$$

and

$$q_{n-1} q'_{n-1} = p_n p'_n - \Re(p_0 p'_0). \tag{5.8}$$

We can differentiate again. Note that the second derivative of a product ab is

$$\delta a \, db + a \, \delta db + \delta da \, b + da \, \delta b$$

if dX and δX are the derivatives of X in two directions. Hence, we get 12 terms for the derivatives of (5.7). Equation (5.8) gives us q'_{n-1} and equation (5.7) gives us q'_j. The complexity for computing the generic term q_j, q'_j and q''_j is hence 3, 6 and 12 respectively. In the complex case, we take advantage of the fact that the leading coefficients of P and Q are real, so that, instead of multiplying these numbers by 4, we get 8, 16 and 32. The cost of the leading term q_{n-1}, q'_{n-1} and q''_{n-1} is 2, 3 and 6 in the real case, 3, 4 and 8 in the complex case. Note that, for q_{n-1} we have to extract a square root.

Thus, we get a cost of $12k + 6$ for one partial second derivative of Q, in the case where Q has degree k. We have to compute all Φ_k (of degree k) for $n \le k \le n + d$. Note that $\Phi_{n+d} = qw$, so that the first derivative with respect to q_i is just $z^i w$: no multiplication is required. The second derivative is zero. Results are in table 5.1.

Table 5.1: Complexity of orthogonal polynomials (time)

	Ψ	Ψ'	Ψ''
real	$d(4n+3d/2+1/2)$	$nd(6n+3d)$	$3n(n+1)(2n+d)d$
complex	$d(12n+4d+1)$	$8nd(4n+2d-1)$	$8n(2n+1)(4n+2d-1)d$
$n=5, d=1$	22 / 65	165 / 840	990 / 9 240
$n=8, d=2$	71 / 210	864 / 4 480	7 776 / 76 160
$n=10, d=5$	240 / 705	3 750 / 19 600	41 250 / 411 600
$n=20, d=10$	955 / 2810	30 000 / 158 400	630 000 / 6 494 400

Table 5.2: Complexity of orthogonal polynomials (space)

	real	complex
	$(n^2+n+1)(n+1+d/2)(d+1)$	$(4n^2+2n+1)(2n+2+d)(d+1)$
$n=5, d=1$	403	5 106
$n=8, d=2$	2 190	16 380
$n=10, d=5$	8 981	63 992
$n=20, d=10$	120 406	938 652

Memory requirements: for simplicity we store the full Hessian. This gives hence table 5.2. The algorithm for computing Ψ' is not optimal. To see why, consider for instance the case $d=2$ and $n=3$. We have

$$\Phi_5 = w_0q_0 + (w_0q_1 + w_1q_0)z + (w_0q_2 + w_1q_1 + q_0)z^2 + (w_0 + w_1q_2 + q_1)z^3 + (w_1 + q_2)z^4 + z^5.$$

Let $\alpha = \sqrt{1 - w_0^2 q_0^2}$. Note that α depends only on q_0, and $\alpha d\alpha/dq_0 = -q_0 w_0^2$. Let $\Phi_5 = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + a_4 z^4 + z^5$. Then

$$\alpha\Phi_4 = a_1 - a_0a_4 + (a_2 - a_0a_3)z + (a_3 - a_0a_2)z^2 + (a_4 - a_0a_1)z^3 + \alpha^2 z^4.$$

Write this as $b_0 + b_1 z + b_2 z^2 + b_3 z^3 + b_4 z^4$. Now b_4 is independent of q_1, q_2. Moreover, the derivative of b_i with respect to q_j $(j \neq 0)$ has the form $a'_u - a_0 a'_v$, for some indices u and v. Given the form of a_i, we can compute a'_u without multiplication. Now a'_v is 1, w_0 or w_1. Hence it suffices to use 2 multiplications in order to compute $a_0 a'_v$. Thus the complexity is 10 (we have 8 divisions), instead of 54. Since the total cost is 144, this reduces it to 100. If we compute the second derivative, it is clear that $\partial^2\Phi_u/\partial q_i \partial q_j$ is zero, if none of i and j is zero. Thus, it suffices to compute only the first row of the Hessian.

In the general case, we can optimise the code of Φ_{n+d-1}, and perhaps the code of Φ_{n+d-2}. Given the complexity of the other terms, these optimisations can be neglected.

5.2.2 Additional code

We have to compute now ψ_0 and its derivative, where

$$\psi_0 = \sum_{j=1}^{M} \sum_{i=n}^{n+d-1} \left| \langle \frac{\Phi_i e_j}{q} \mid F \rangle_w \right|^2 \tag{5.9}$$

Denote by s_k the scalar product in this expression. Then $\psi_0 = \sum |s_k|^2$, $d\psi_0 = 2\sum \Re(\overline{s_k}\, ds_k)$ and $\delta d\psi = 2\sum \Re(\overline{\delta s_k} ds_k + \overline{s_k} \delta ds_k)$. In the next table (5.3), we assume $M=1$ (recall that M is the number of components of F). We show the cost of ψ_0 and its derivatives, assuming that s_k and their derivatives have been computed.

Table 5.3: Complexity of squares in (5.9), case $M = 1$

	ψ	ψ'	ψ''
real	Md	Mnd	$Mdn(n+1)$
complex	$2Md$	$4Mnd$	$4Mdn(2n+1)$
$n = 5, d = 1$	1/ 2	5/ 20	30/ 220
$n = 8, d = 2$	2/ 4	16/ 64	144/ 1 088
$n = 10, d = 5$	5/10	50/200	550/ 4 200
$n = 20, d = 10$	10/20	200/800	4200/32 800

We compute now the complexity of a single scalar product like

$$\psi_0 = \langle \frac{F}{w} \mid \frac{\Phi}{qw} \rangle. \tag{5.10}$$

We have

$$d\psi_0 = \langle \frac{F}{w} \mid \frac{d\Phi}{qw} \rangle - \langle \frac{F}{w} \mid \frac{\Phi dq}{q^2 w} \rangle. \tag{5.11.a}$$

$$d\psi_0 = \langle \frac{F}{w} \mid \frac{q\, d\Phi - \Phi\, dq}{q^2 w} \rangle. \tag{5.11.b}$$

$$\delta d\psi_0 = \langle \frac{F}{w} \mid \frac{\delta d\Phi}{qw} \rangle - \langle \frac{F}{w} \mid \frac{\delta q\, d\Phi + dq\, \delta\Phi}{q^2 w} \rangle + 2 \langle \frac{F}{w} \mid \frac{\Phi\, dq\, \delta q}{q^3 w} \rangle. \tag{5.12.a}$$

$$\delta d\psi_0 = \langle \frac{F}{w} \mid \frac{q^2\, \delta d\Phi - q\, \delta d\, d\Phi - q\, \delta\Phi\, dq + 2\Phi\, dq\, \delta q}{q^3 w} \rangle. \tag{5.12.b}$$

The question is now: should we implement (5.11.a) and (5.12.a) or (5.11.b) and (5.12.b). The answer depends on what is the cheapest: the scalar product $\langle F \mid d\Phi/q \rangle_w$ or the product $qd\Phi$. Let's start with easy things: the multiplication by 2 in (5.12). In equation (5.12.a), the best thing to do is multiply F by 2. This costs (in the example) 400 multiplications, and 400 memory cells. In the case of (5.12.b), the best thing to do is to multiply Φ by 2. This costs

$$d(2n + d + 1)/2 \tag{5.13}$$

multiplications and memory cells. This is small, so that we do not count it in our tables.

Assume now that we implement (5.12.b). The numerator has the form $aq^2 + bq + c$. We can compute this as

$$(aq + b)q + c = a(q^2) + bq + c. \tag{5.14}$$

In general the first method (Horner scheme) is more efficient than the second one. But $q\, d\Phi$ appears in (5.11), hence can be precomputed. In the next table (cf 5.4) we give the memory cost of storing these quantities (which is $nd(4n + d + 1)/2$).

In the next table (cf 5.5), we give the complexity of computing the denominator of (5.11.b) and (5.12.b). Recall that we have $qw = \Phi_{n+d}$, and this is already computed. The complexity is $n(n + d)$ for the product $q^2 w$ and $n(2n+d)$ for the product $q^3 w$. If we assume that we compute (5.14) as $a(q^2)+bq+c$, we count also the cost of q^2 which is n^2. Now, we can compute $q^2 w$ by multiplying q^2 by w. This costs $2nd$. Whether this is better than multiplying qw by q depends on which is the greatest, n or d. In our example, we always have $n > d$, but this is not always true. In fact, since q^2 is needed for the Hessian, and $q^2 w$ for the first derivative, the best thing to do when we compute the first derivative is to multiply qw by q. On the other hand, when we compute the Hessian, we need $q^3 w$. We can multiply qw by q^2, with a cost of $2n(n + d)$, but this is greater than $n(2n + d)$, the cost of multiplying $q^2 w$ by q.

Table 5.4: Space needed for the products $q\,d\Phi$

	real	complex
$n = 5, d = 1$	55	220
$n = 8, d = 2$	280	1 120
$n = 10, d = 5$	1150	4 600
$n = 20, d = 10$	9100	36 400

Table 5.5: Complexity of denominators in (5.11) and (5.12)

	ψ'	ψ''	ψ'' plus q^2
real	$n(n+d)$	$n(2n+d)$	$n(3n+d)$
complex	$4n(n+d)$	$4n(2n+d)$	$4n(3n+d)$
$n = 5, d = 1$	30/ 120	55/ 220	80/ 320
$n = 8, d = 2$	80/ 320	144/ 876	208/ 432
$n = 10, d = 5$	150/ 600	250/1000	350/1400
$n = 20, d = 10$	600/2400	1000/4000	1400/5600

In the next table (cf. 5.6) we give the complexity for computing the numerators of (5.11.b) and (5.12.b). In the first case, we have to compute $q\,d\Phi$. If Φ has degree i, this costs $n(i+1)$. Since i varies between n and $n+d-1$, this gives a total cost of $nd(2n+d+1)/2$. We have to multiply this by the number of directions $d\Phi$ (n in the real case, and $2n$ in the complex case). In the second column of the table, we give the costs of the right-hand side of (5.14), assuming that $q\,d\Phi$ has been computed and stored. Each term costs $2n(i+1)$, hence a total of $nd(2n+d+1)$ for each direction. In the real case, there are $n(n+1)/2$ directions (because the Hessian is a symmetric matrix). In the complex case there are $n(2n+1)$ directions. In the last last column, we give the cost when we use the left-hand side of (5.14). We have a cost of $n(n+2i+2)$ for each direction.

Table 5.6: Complexity of numerators in (5.11) and (5.12)

	ψ'	ψ'' (fast)	ψ'' (slow)
real	$n^2d(2n+d+1)/2$	$n^2d(2n+d+1)(n+1)/2$	$n^2d(3n+d+1)(n+1)/2$
complex	$4n^2d(2n+d+1)$	$4n^2d(2n+d+1)(2n+1)$	$4n^2d(3n+d+1)(2n+1)$
$n = 5, d = 1$	150/ 1 200	900/ 13 200	1 275/ 18 500
$n = 8, d = 2$	1 216/ 9 728	10 944/ 165 376	15 552/ 235 008
$n = 10, d = 5$	6 500/ 52 000	71 500/ 1 092 000	99 000/ 1 512 000
$n = 20, d = 10$	102 000/816 000	2 142 000/33 456 000	2 982 000/46 576 000

Assume now that the scalar product

$$\langle \frac{F}{w} \mid \frac{p}{q} \rangle \tag{5.15}$$

is computed via its Taylor expansion. Write

$$\frac{F}{w} = \sum_{k=0}^{\infty} f_k / z^{k+1}, \tag{5.16.a}$$

$$\frac{p}{q} = \sum_{k=0}^{\infty} p_k / z^{k+1}. \tag{5.16.b}$$

Then

$$\langle \frac{F}{w} \mid \frac{p}{q} \rangle = \sum_{k=0}^{\infty} f_k p_k. \tag{5.16.c}$$

Let F_w be the truncation of F/w with m' terms and $F' = wF_w$. If m' is big enough, the best approximation to F is very near to the best approximation to F'. This means that we can replace F by F' (in fact, in some examples given to HYPERION, the data are measured with an unknown precision, and given with 6 digits; the quantity F is an approximation to these data, with a typical precision of 10^{-3} in the good cases, 10^{-2} in the bad cases. We do not lose precision in the case where the distance between F and F' is less than 10^{-6}).

Note however that we are computing the second term in (2.71.3). If we replace F by F', we should also replace G by G'. This means that the complexity of the first term is increased. In fact, we do not replace G by G'. This has no practical consequence, but in theory, it could change some results (the behaviour on the boundary of the manifold, for instance).

The number of terms to take in the truncation depends on F and w. If the zeroes of w are small (in fact if $1 - |z|$ is small enough for each root z of w), then we can take a small value for $m' - m$, for instance $m/10$. In the examples below, we take $m' - m = m/2$, thus $m' = 600$.

Quantities p_k in (5.16.b) are computed as follows. Assume

$$\frac{p}{q} = \sum_{k=0}^{n} p_k / z^{k+1} + \frac{s}{qz^{n+1}} \tag{5.17.a}$$

then

$$\frac{p}{q} = \sum_{k=0}^{n+1} p_k / z^{k+1} + \frac{s'}{qz^{n+2}} \tag{5.17.b}$$

provided that

$$sz = p_{n+1} q + s'. \tag{5.17.c}$$

In other words, the cost of each coefficient p_k is the degree of q (we assume q monic). In (5.16.c), we have m' coefficients to compute, and for each coefficient one multiplication is required. This gives a complexity of $m'(n+1)$. In table 5.7 we give the complexity of the scalar products.

Table 5.7: Complexity of scalar products via truncation

	ψ	ψ'	ψ''
real complex	$m'(n+d+1)d$ $4m'(n+d+1)d$	$m'(2n+d+1)dn$ $8m'(2n+d+1)dn$	$m'(3n+d+1)dn(n+1)/2$ $4m'(3n+d+1)dn(2n+1)$
$n=5, d=1$	4 200/ 16 800	36 000/ 288 000	153 000/ 2 244 000
$n=8, d=2$	13 200/ 52 800	182 400/ 1 539 200	1 456 400/ 21 705 600
$n=10, d=5$	48 000/192 000	780 000/ 6 240 000	5 940 000/ 90 720 000
$n=20, d=10$	186 000/744 000	6 120 000/48 960 000	89 400 000/1 377 280 000

It is clear, according to this table, that (5.12.b) should be preferred to (5.12.a) (it is also clear that another method should be used to compute the scalar products).

One way to reduce the complexity is by avoiding the truncation. This essentially replaces m' by m. We shall give later another way to compute the scalar products. The complexity is still $O(n^3 d)$ for the Hessian, but we can remove the factor m (thus replace ten minutes by one second).

Assume that w is a polynomial of degree d. By definition of the scalar product, we have

$$\langle \frac{F}{w} \,|\, x\rangle = \langle F \,|\, \frac{xz^d}{\tilde{w}}\rangle.$$

Let $x = p/s$, and consider the equation

$$As + B\tilde{w} = z^d. \tag{5.18.a}$$

Then

$$\langle \frac{F}{w} \,|\, \frac{p}{s}\rangle = \langle F \,|\, \frac{pA}{\tilde{w}}\rangle + \langle F \,|\, \frac{Bp}{s}\rangle.$$

Since w and F are stable, F is orthogonal to $Ap/\tilde{w}$, so that

$$\langle \frac{F}{w} \,|\, \frac{p}{s}\rangle = \langle F \,|\, \frac{Bp}{s}\rangle. \tag{5.18.b}$$

In the next table 5.8, we give the cost of computing A, B and s. Recall that s is qw for ψ, q^2w for ψ' and q^3w for ψ''. This is a stable polynomial, so that the Bezout equation (5.18.a) has a solution.

Table 5.8: Complexity of the Bezout relation (5.18.a)

	ψ	ψ'	ψ''
real complex	$(2d+1)(n+2d)$ $4(2d+1)(n+2d)$	$(2d+1)(2n+2d)+n(n+d)$ $4(2d+1)(2n+2d)+4n(n+d)$	$(2d+1)(3n+2d)+(n+2d)n$ $4(2d+1)(3n+2d)+4(n+2d)n$
$n=5, d=1$	21/ 82	66/ 264	86/ 344
$n=8, d=2$	60/ 240	180/ 720	236/ 944
$n=10, d=5$	220/ 880	480/1920	660/2640
$n=20, d=10$	840/3360	1860/7440	2480/9920

In the next table 5.9, we give the cost of the scalar products $\langle F \,|\, Bp/s\rangle$. Assume that $s = q^i w$, so that its degree is $d + in$ and that p has degree j. Now B has degree $d + in - 1$ and the cost of the product Bp is $j(d + in)$. Its degree is $j + d + in - 1$. Thus Bp/s has the form $\alpha + \beta/s$, where $\deg(\beta) < \deg(s)$, and α has degree $j - 1$. Computing α and β costs $j(d + in)$, and $\langle F \,|\, Bp/s\rangle = \langle F \,|\, \beta/s\rangle$. The cost of the scalar product is $m(d + in)$. This gives a total cost of $(d + in)(m + 2j)$. Such a scalar product must be computed for each orthogonal polynomial and for each direction of differentiation.

Table 5.9: Complexity of scalar products via the Bezout relation, $\alpha = m + 2n + d + 1$

	ψ	ψ'	ψ''
real complex	$(n+d)d\alpha$ $4(n+d)d\alpha$	$n(2n+d)d\alpha$ $8n(2n+d)d\alpha$	$(3n+d)d\alpha n(n+1)/2$ $4(3n+d)d\alpha n(2n+1)$
$n=5, d=1$	2 47/ 9 888	22 660/ 181 280	98 850/ 1 449 800
$n=8, d=2$	8 380/ 33 520	120 672/ 965 376	784 368/ 11 852 672
$n=10, d=5$	31 950/127 800	532 500/ 4 260 000	4 100 250/ 63 622 000
$n=20, d=10$	135 300/541 200	4 150 000/33 220 000	66 297 000/1 035 496 000

Note that the last number in this table is still greater than 10^9.

We consider now another technique for computing the scalar products. Let d_i be defined by

$$\langle \frac{F}{w} \,|\, \frac{z^i}{s}\rangle = d_i. \tag{5.19.a}$$

Then, if $p = \sum p_i z^i$, $\langle F/w \,|\, p/s\rangle = \sum p_i d_i$, so that, if $D = \sum d_i z^i$,

$$\langle \frac{F}{w} \,|\, \frac{p}{s}\rangle = \langle p \,|\, D\rangle. \tag{5.19.b}$$

In the next table 5.10, we give the cost for (5.11.b) and (5.12.b), if the scalar products are computed in this way.

Table 5.10: Complexity of scalar products via (5.19.b)

	ψ	ψ'	ψ''
real	$d(2n+d+1)/2$	$nd(4n+d+1)/2$	$d(6n+d+1)n(n+1)/4$
complex	$2d(2n+d+1)$	$4nd(4n+d+1)$	$2d(6n+d+1)n(2n+1)$
$n=5, d=1$	6/ 24	55/ 440	240/ 3 520
$n=8, d=2$	19/ 74	280/ 2 240	1 836/ 27 744
$n=10, d=5$	65/ 260	1150/ 9 200	9 075/ 138 600
$n=20, d=10$	255/1020	9100/72 800	137 550/2 148 400

Now, since these numbers are small, it is worth implementing (5.11.a) and (5.11.b). Assume that Φ has degree j. Then the cost of $\langle F/w \,|\, d\Phi/qw\rangle$ is $j+1$. The cost of the scalar product of F/w and $\Phi dq/q^2 w$ is also $j+1$, because $\Phi\, dq$ has only $j+1$ non zero coefficients. Thus we get a cost of $2(j+1)$, hence a cost of $d(2n+d+1)$ for each direction. In the same fashion, we can write (5.11.b) as a sum of four terms, each with complexity $j+1$. Results are given in table 5.11.

Table 5.11: Complexity of scalar products via (5.19.b), and (5.11.a), (5.12.a)

	ψ	ψ'	ψ''
real	$d(2n+d+1)/2$	$nd(2n+d+1)$	$d(2n+d+1)n(n+1)$
complex	$2d(2n+d+1)$	$8nd(2n+d+1)$	$8d(2n+d+1)n(2n+1)$
$n=5, d=1$	6/ 24	60/ 480	360/ 5 280
$n=8, d=2$	19/ 74	304/ 2 432	2 736/ 41 344
$n=10, d=5$	65/ 260	1 300/10 400	14 300/ 218 400
$n=20, d=10$	255/1020	10 200/81 600	214 200/3 345 600

We have now to compute the quantities that appear in (5.19.a). Assume that s has degree n and that $z^{n-1}/s = \sum_k s_k/z^{k+1}$. Moreover, assume that we use the truncated power series expansion of F/w. Then

$$\langle \frac{F}{w} \,|\, \frac{z^i}{s}\rangle = \langle \frac{F}{w} \,|\, \sum s_k z^{i-n+1}/z^{k+1}\rangle$$

and

$$d_i = \sum_k f_{k+(n-i-1)} s_k.$$

Hence, we get a cost of $2m'n$, see table 5.12.

Assume now that we use formula (5.18.b). We get

$$d_i = \langle \frac{F}{w} \,|\, \frac{z^i}{s}\rangle = \langle F \,|\, \frac{Bz^i}{s}\rangle = \langle \frac{F}{z^i} \,|\, \frac{B}{s}\rangle.$$

This means that we can use the same technique as before. We have to compute B and the expansion B/s, the cost is given in table 5.8, and then the scalar products are as in table 5.12, with m' replaced by m (for numeric values, multiply everything in the previous table by 2/3).

Table 5.12: Complexity of the relation (5.19.a)

	ψ	ψ'	ψ''
real	$2m'(n+d)$	$2m'(2n+d)$	$2m'(3n+d)$
complex	$8m'(n+d)$	$8m'(2n+d)$	$8m'(3n+d)$
$n=5, d=1$	7 200/ 28 800	13 200/ 52 800	19 200/ 76 800
$n=8, d=2$	12 000/ 48 000	21 600/ 86 400	31 200/124 800
$n=10, d=5$	18 000/ 72 000	30 000/120 000	42 000/168 000
$n=20, d=10$	36 000/144 000	60 000/240 000	84 000/336 000

The conclusion is the following: if there are few scalar products to compute (i.e., when we compute ψ, and n, d are small), we can use a direct formula. Otherwise, the best thing to do is compute D such that $\langle F/w \,|\, x/q\rangle = \langle D \,|\, x\rangle$, and replace all scalar products with F/w by scalar products with D. In the case where n is large enough, in the case of the derivatives of ψ, the complexity of the scalar products becomes smaller that the complexity of the orthogonal polynomials.

5.2.3 Non-weighted case

Here, we have to compute ψ and its derivatives, given the formulas

$$G\tilde{q} = Vq + R, \qquad \psi = \|V\|^2. \tag{5.20}$$

A basic implementation says that the cost of ψ is $m(2n+1)$. If we differentiate in reverse mode, we get a complexity of $3mn$, because G is constant, thus a cost of $m(5n+1)$ for ψ and ψ'. If we differentiate again, we get a cost of $8mn^2$.

In the table 5.13, we give the complexity of computing ψ and its derivatives in the case of a weight. Here we have $Gz^d\tilde{q} = Vqw + R$.

Table 5.13: Complexity of ψ_1, first part of ψ with weight

	ψ	ψ'	ψ''
real	$m(2n+d+1)$	$m(3n+2d)+n(d+1)$	$nm(8n+6d+1)+2n^2(d+1)$
complex	$m(8n+4d+2)$	$4m(3n+2d)+4n(d+1)$	$mn(64n+48d+2)+16n^2(d+1)$
$n=5, d=1$	4 800/18 400	6 810/ 27 240	94 100/ 740 800
$n=8, d=2$	7 600/29 600	11 224/ 44 896	246 784/ 1 955 072
$n=10, d=5$	10 400/40 800	16 060/ 64 240	445 200/ 3 537 600
$n=20, d=10$	20 400/80 800	32 220/128 880	1 776 800/14 166 400

In the case without weight, we can simplify a bit these formulas. Results are given in table 5.14.

The question is now: what is the intrinsic complexity of ψ? We assume that G is a polynomial, of degree $m-1$, hence has m terms. If we consider the equations

$$zP_{k+1} + g_k\tilde{q} = V_kq + P_k, \tag{5.21}$$

where P_k are polynomials of degree $< n$, initialised with $P_m = 0$, then these equations allow us to compute all the coefficients V_k of V, with the same time complexity as above, but we have only to allocate n memory cells for P, instead of $n+m$ for the product $G\tilde{q}$.

Remember that, in the case where q is of degree one and $q = z - \alpha$, we have

$$\psi = \|G\|^2 - (1-|\alpha|^2)|G(\alpha)|^2, \tag{5.22}$$

Table 5.14: Complexity of ψ, scalar case, no weight

	ψ	ψ'	ψ''
real complex	$m(2n+1)$ $m(8n+2)$	$3mn$ $12mn$	$nm(8n+1)$ $mn(64n+2)$
$n=5$	4 400/16 800	6 000/24 000	82 000/ 644 000
$n=8$	6 800/26 400	9 600/38 400	208 000/ 1 644 800
$n=10$	8 400/32 800	12 000/48 000	324 000/ 2 568 000
$n=20$	16 400/64 800	24 000/96 000	1 288 000/10 256 000

and this gives a complexity of $m+3$ (recall that the norm of G is one). This is smaller than $m(2n+1)$ evaluated at $n=1$.

More generally, assume that $q=\prod(z-\alpha_i)$, and q has only simple roots. Then (5.20) says

$$G(\alpha_i)\tilde{q}(\alpha_i)=R(\alpha_i). \tag{5.23.a}$$

Let $P=\tilde{R}$. There exist some quantities β_i such that $P/q=\sum\beta_i/(z-\alpha_i)$ and

$$\langle F\mid\frac{P}{q}\rangle=\sum\langle F\mid\frac{\beta_i}{z-\alpha_i}\rangle=\sum_i\beta_i G(\alpha_i). \tag{5.23.c}$$

Now, the quantities β_i are defined by the equation:

$$\sum_j\beta_j\prod_{k\neq j}(1-\overline{\alpha_k}\alpha_i)=\overline{R(\alpha_i)}. \tag{5.23.c}$$

Recall that $\psi=\|F\|^2-\langle F\mid P/q\rangle$. Thus, the complexity for ψ is the following: we need nm multiplications for evaluating G at α_i, plus a cost that depends only on n: we have to compute the roots of q, evaluate $\tilde{q}(\alpha_i)$, compute $\prod(1-\overline{\alpha_k}\alpha_i)$, solve (5.23.$c$), etc. Hence, we have an algorithm whose complexity is $mn+f(n)$. It works only in the case where q has single roots.

In order to see what happens if q has multiple roots, let's consider the case where q has two roots α and β. We have

$$\langle F\mid\frac{P}{q}\rangle=\frac{X+Y-Z}{\Delta}, \tag{5.24.1}$$

$$\Delta=|1-\alpha\overline{\beta}|^2-(1-|\alpha|^2)(1-|\beta|^2), \tag{5.24.2}$$

$$X=|G(\alpha)|^2(1-|\alpha|^2)|1-\alpha\overline{\beta}|^2, \tag{5.24.3}$$

$$Y=|G(\beta)|^2(1-|\beta|^2)|1-\alpha\overline{\beta}|^2, \tag{5.24.4}$$

$$Z=2\Re[G(\alpha)\overline{G(\beta)}(1-\alpha\overline{\beta})](1-|\alpha|^2)(1-|\beta|^2). \tag{5.24.5}$$

These formulas are valid only in the case $\alpha\neq\beta$. In fact, $\Delta=|\alpha-\beta|^2$. But we have a limit, and this is

$$\langle F\mid\frac{P}{q}\rangle=|G'(\alpha)|^2(1-|\alpha|^2)^3+|G(\alpha)|^2(1-|\alpha|^2)-2\Re[\overline{G(\alpha)}G'(\alpha)\alpha](1-|\alpha|^2)^2. \tag{5.25}$$

5.2.4 Alternate formulas

Consider equations (5.24) in the real case. These are symmetric functions of α and β, hence must be rational functions of the coefficients of q (this is false in the complex case: ψ is not analytic, but there must be a formula). The idea is the following: since ψ depends on G only through the values of G at the roots of q, there has to be a way to express ψ as a function of the remainder of G by q. Thus the formula:

$$G=V_1q+R_1\qquad R_1\tilde{q}=V_2q+R_2, \tag{5.26.a}$$

$$\psi = \|G\|^2 - \|R_1\|^2 + \|V_2\|^2. \tag{5.26.b}$$

This trick does not work in the weighted case: essentially, we have to replace G by $G_w = Gz^d/\tilde{w}$. But this is not a polynomial. We get a polynomial if we multiply the first equation by $\tilde{w}$. This means that we have to divide Gz^d by $q\tilde{w}$. Note that the division is possible, but numerically unstable. Results are given in table 5.15.

Table 5.15: Complexity of ψ, scalar case, alternate

	ψ	ψ'	ψ''
real	$mn + 2n + n^2$	$mn + 3n^2$	$n(2mn + 10n^2 + 2n)$
complex	$4(mn + 2n + n^2)$	$4(mn + 3n^2)$	$8n(2mn + 10n^2 + 2n)$
$n = 5$	2 035/ 8 140	2 075/ 8 300	21 300/ 170 400
$n = 8$	3 280/13 120	3 392/13 568	56 448/ 451 584
$n = 10$	4 120/16 480	4 300/17 200	90 200/ 721 600
$n = 20$	8 440/33 760	9 200/36 800	400 800/3 206 400

5.2.5 Direct mode differentiation

Let's introduce the quantities defined by the following equations

$$Gz^{n-i} = U_i q + C_i, \qquad Vz^i = W_i q + D_i \tag{5.27.a}$$

$$U_i z^j = X_{ij} q + Z_{ij}, \qquad W_i z^j = Y_{ij} q + T_{ij}. \tag{5.27.b}$$

If we differentiate $G\tilde{q} = Vq + R$, $\psi = \|V\|^2$, we get

$$G\partial\tilde{q} = \partial V q + V\partial q + \partial R, \tag{5.28.a}$$

$$\partial\psi = 2\Re\langle V \,|\, \partial V\rangle. \tag{5.28.b}$$

We assume now that $q = \sum(q_k + ir_k)z^k$, where $q_n + ir_n = 1$. In (5.28), we have to consider the case where ∂X is the derivative with respect to q_k or r_k. In this case, ∂q is z^k or iz^k, while $\partial\tilde{q}$ is z^{n-k} or $-iz^{n-k}$. Hence we get

$$Gz^{n-k} - Vz^k = q\frac{\partial V}{\partial q_k} + \frac{\partial R}{\partial q_k}$$

$$-iGz^{n-k} - iVz^k = q\frac{\partial V}{\partial r_k} + \frac{\partial R}{\partial r_k}$$

According to (5.27.a), we get

$$\frac{\partial V}{\partial q_k} = U_k - W_k, \qquad \frac{\partial V}{\partial r_k} = -iU_k - iW_k, \tag{5.29.a}$$

and (5.28.b) gives

$$\frac{\partial\psi}{\partial q_k} = 2\Re\langle V \,|\, U_k - W_k\rangle, \qquad \frac{\partial\psi}{\partial r_k} = 2\Im\langle V \,|\, U_k + W_k\rangle. \tag{5.29.b}$$

If we differentiate again, we obtain

$$d\partial\psi = 2\Re\langle dV \,|\, \partial V\rangle + 2\Re\langle V \,|\, d\partial V\rangle, \tag{5.30.a}$$

$$Gd\partial\tilde{q} = d\partial V q + dV\partial q + \partial V dq + Vd\partial q + d\partial R.$$

Since $d\partial\tilde{q}$ and $d\partial q$ are zero, we get

$$-dV\partial q - \partial V dq = d\partial V q + d\partial R. \tag{5.30.b}$$

If we use (5.29.a), we get

$$-U_k z^j + W_k - U_j z^k + W_j z^k = \frac{\partial^2 V}{\partial q_k \partial q_j} q + \frac{\partial^2 R}{\partial q_k \partial q_j}$$

and some other formulas. From (5.27), we get now

$$\frac{\partial^2 \psi}{\partial q_i \partial q_j} = 2\Re\langle U_i - W_i \,|\, U_j - W_j\rangle + 2\Re\langle V \,|\, Y_{ij} + Y_{ji} - X_{ij} - X_{ji}\rangle, \tag{5.31.a}$$

$$\frac{\partial^2 \psi}{\partial q_i \partial r_j} = -2\Im\langle U_j + W_j \,|\, U_i - W_i\rangle + 2\Im\langle V \,|\, -Y_{ij} - Y_{ji} + X_{ij} - X_{ji}\rangle, \tag{5.31.b}$$

$$\frac{\partial^2 \psi}{\partial r_i \partial r_j} = 2\Re\langle U_i + W_i \,|\, U_j + W_j\rangle - 2\Re\langle V \,|\, Y_{ij} + Y_{ji} + X_{ij} + X_{ji}\rangle. \tag{5.31.c}$$

Let $Q_q(X)$ and $R_q(X)$ be the quotient and remainder in the division of X by q. Let also $Q_n(X)$ and $R_n(X)$ be the quotient and remainder in the division of X by z^n. The second equation (5.27.a) is

$$D_i z = \alpha_i q + D_{i+1}, \qquad W_i z + \alpha_i = W_{i+1}. \tag{5.32}$$

Thus $W_i = Q_1(W_{i+1})$. More generally,

$$W_i = Q_{n-i-1}(W_{n-1}), \qquad U_i = Q_i(U_0). \tag{5.33}$$

Let v_i be the coefficient of z^i in V, w_i the coefficient of z^i in W_{n-1} and u_i the coefficient of z^i in U_0. Then (5.29.b) is

$$\frac{\partial \psi}{\partial q_k} = 2\Re\sum_i v_i(u_{i-k} - w_{n-k-1+i}). \tag{5.34}$$

This relation says that we have only to compute W_{n-1} and U_0 and allocate memory for these two polynomials. Since G has degree $m-1$, U_0 has degree $m-1$ and W_{n-1} has degree $m-2$. The cost of computing these polynomials is $(2m-1)n$. The number of terms in (5.34) is

$$\max(m-k, m-n+k).$$

This is between $m-n/2$ and m. Since n is small compared to n, we can estimate it to m (in fact, in order to simplify the code, the polynomials are padded with zeroes, so that always m terms are computed). Thus, the total cost of ψ' is $(3m-1)n$.

From (5.27), we get the equivalent of (5.32), namely

$$Y_{i,j+1} = Y_{ij} z + \beta_{ij}. \tag{5.35}$$

But (5.32) says

$$W_i z^{j+1} = Y_{i+1,j} q + T_{i+1,j} - \alpha_i z^j. \tag{5.36}$$

This equation says $Y_{i+1,j} = Y_{i,j+1}$. In fact, Y_{ij} is nothing else than the quotient of Vz^{i+j} by q^2. The equivalent of (5.33) is

$$X_{ij} = Q_{n-j-1+i}(X_{0,n-1}), \qquad Y_{ij} = Q_{n-i-1+n-j-1}(Y_{n-1,n-1}). \tag{5.37}$$

Note that $X_{0,n-1}$ has degree $m-2$, and $Y_{n-1,n-1}$ has degree $m-3$. This gives a cost of $(2m-3)n$ for computing these polynomials. Let's compute the cost of equations (5.31). Each scalar product has a cost of m. In the real case, this gives a cost of $mn(n+1)$. There is another way to compute these equations: we compute $\langle V \mid X_{ij} \rangle$ and $\langle V \mid Y_{ij} \rangle$ for each i and j. Since X_{ij} depends only on $i-j$ and Y_{ij} depends only on $i+j$, we have $2(2n-1)$ such scalar products to compute. This gives a cost of $mn(n+1)/2+2m(2n-1)$. This is cheaper for $n \geq 7$. In the complex case, the first scalar products in (5.31) costs $2n(2n+1)m$. This gives a cost of $4n(2n+1)m$ if we implement directly (5.31). If we compute the scalar products $\langle V \mid X_{ij} \rangle$ and $\langle V \mid Y_{ij} \rangle$, we get a cost of $2n(2n+1)m + 8m(2n-1)$. This is cheaper for $n \geq 2$. Results are given in table 5.16. It shows that for small n, there is little difference between these two methods. In table 5.17, we give the complete complexity.

Table 5.16: Comparison of the two methods of computing ψ''. It has the form $m\alpha(n) + \beta(n)$. The table gives $\alpha(n)$.

n	real 1	real 2	complex 1	complex 2
1	4	5	20	22
2	10	13	56	60
3	18	22	108	106
4	28	32	176	160
5	40	43	260	222
6	54	55	360	292
7	70	68	476	370

Table 5.17: Complexity of the derivatives of ψ, scalar case, direct mode

	ψ	ψ'	ψ''
real	$m(2n+1)$	$(3m-1)n$	$m(n(n+1)/2+6n-2)-3n$
complex	$m(8n+2)$	$4(3m-1)n$	$m(4n^2+26n-8)-12n$
$n=5$	4 400/16 800	5 995/23 980	17 185/ 88 740
$n=8$	6 800/26 400	9 592/38 368	32 776/182 304
$n=10$	8 400/32 800	11 990/47 960	45 170/260 680
$n=20$	16 400/64 800	23 980/95 920	131 140/844 560

Consider now what happens when we implement the alternate formulas (5.26). We get

$$-V_1 \partial q = \partial V_1 q + \partial R_1, \tag{5.38.a}$$

$$\partial R_1 \tilde{q} + R_1 \partial \tilde{q} - V_2 \partial q = \partial V_2 q + \partial R_2, \tag{5.38.b}$$

$$\partial \psi = 2\langle V_2 \mid \partial V_2 \rangle - 2\langle R_1 \mid \partial R_1 \rangle. \tag{5.38.c}$$

If we differentiate with respect to q_i, the first equation becomes

$$-V_1 z^i = \partial V_1 q + \partial R_1. \tag{5.39}$$

Consider

$$V_1 z^i = W_i q + D_i. \tag{5.40}$$

Now (5.32) is still valid:

$$D_i z = \alpha_i q + D_{i+1}, \qquad W_i z + \alpha_i = W_{i+1}. \tag{5.41}$$

We cannot obtain D_{i+1} directly from D_i. However, computing D_0 via (5.40) and D_i via (5.41) has the same cost as dividing Gz^{n-1} by q: it costs $(m-1)n$, and an additional memory cost of $n(n-1)$. The cost of $\langle R_1 \,|\, \partial R_1 \rangle$ is $n(n-1)$. In order to implement (5.38.b), we have to divide $R_1 \partial \tilde{q} - V_2 \partial dq$ by q. Using the previous techniques, this costs $(2n-1)n$. The cost of $\langle V_2 \,|\, \partial V_2 \rangle$ is n^2.

Now, there is an additional term: the division of $\partial R_1 \tilde{q}$ by q. Write

$$D_i \tilde{q} = X_i q + S_i. \tag{5.42}$$

We have to compute the quantity X_i. Write $z S_i = \beta_i q + S_{i+1}$. Then

$$X_{i+1} + \alpha_i \tilde{q} = X_i z + \beta_i. \tag{5.43}$$

What we have to compute now is just the quantities $\alpha_i \tilde{q}$. This costs $n(n-1)$. Thus, we have a total cost for ψ' of

$$n(m + 5n - 4). \tag{5.44}$$

Let's consider now the Hessian. Differentiating our equations again gives

$$-dV_1 \partial q - \partial V_1 dq = d\partial V_1 q + d\partial R_1, \tag{5.45.a}$$

$$\partial R_1 d\tilde{q} + dR_1 \partial \tilde{q} + d\partial R_1 \tilde{q} - dV_2 \partial q - \partial V_1 dq = d\partial V_2 q + d\partial R_2, \tag{5.45.b}$$

$$d\partial \psi = \langle V_2 \,|\, d\partial V_2 \rangle + \langle dV_2 \,|\, \partial V_2 \rangle - \langle R_1 \,|\, d\partial R_1 \rangle - \langle dR_1 \,|\, \partial R_1 \rangle. \tag{5.45.c}$$

If

$$W_i z^j = A_{ij} q + B_{ij}$$

then $d\partial R_1 = B_{ij} + B_{ji}$. Remember that $A_{i+1,j} = A_{i,j+1}$. There is no such relation for B. Equation (5.41) says $B_{i+1,j} = B_{i,j+1} + \alpha_i z^j$. This equation says that B_{ij} can be obtained easily if we know B_{0j} and $B_{i,n-1}$. We have also, for some numbers γ_{ij}, $B_{i,j+1} = B_{ij} z - \gamma_{ij} q$. Combining these equations gives $B_{i+1,j} = B_{ij} xz - \gamma_{ij} q + \alpha_i z^j$. Hence, we can compute B_{00} with a cost of $(m-2n)n$, then $B_{0,j}$ for each j with a cost of $(n-1)n$. Once we have $B_{0,n-1}$, we compute $B_{i,n-1}$ with a cost of $(n-1)n$. Hence, we get a total cost of $(m-2)n$. The cost of $\langle R_1 \,|\, d\partial R_1 \rangle + \langle dR_1 \,|\, \partial R_1 \rangle$ is $n^2(n+1)$.

We have now to compute $d\partial V_2$ from (5.45.b). Since there is no obvious relation between the quantities $d\partial R_1$, the best thing to do is to divide for each direction. The non trivial operation on the left hand side of (5.45.b) is the product of $d\partial R_1$ by $\tilde{q}$. This costs n^2. The division costs also n^2. Thus, we have a total of $n^3(n+1)$. This means that the total cost of the second derivative of ψ is

$$(m-2)n + 2n^2(n+1) + n^3(n+1). \tag{5.46}$$

5.2.6 Other formulas

Assume that Y is defined by

$$\langle F \,|\, \frac{D}{q} \rangle = \langle \tilde{D} \,|\, Y \rangle, \quad \deg D < n. \tag{5.47}$$

From $\psi = \|F - P/q\|^2$ we get

$$\partial \psi = 2 \langle F - \frac{P}{q} \,|\, \frac{-\partial P}{q} + \frac{P \partial q}{q^2} \rangle.$$

We have

$$\langle F - \frac{P}{q} \,|\, \frac{rs}{q^2} \rangle = \langle \check{V} \frac{\tilde{q}}{q} \,|\, \frac{rs}{q^2} \rangle = \langle \frac{\tilde{r}}{q} \,|\, \frac{Vs}{q} \rangle.$$

If $r = P$ and $s = \partial q$, $V \partial q = sq + D$ we get

$$\langle F - \frac{P}{q} \,|\, \frac{p \partial q}{q^2} \rangle = \langle \frac{\tilde{p}}{q} \,|\, s + \frac{D}{q} \rangle,$$

hence

$$\partial \psi = \langle \frac{\tilde{p}}{q} \,|\, \frac{D}{q} \rangle = \langle \frac{\tilde{D}}{q} \,|\, \frac{P}{q} \rangle.$$

Remember that $\langle F \,|\, r/q\rangle = \langle P/q \,|\, r/q\rangle$. If we consider again (5.27.a):

$$Vz^i = W_i q + D_i$$

we get

$$\partial\psi = 2\langle D_i \,|\, Y\rangle. \tag{5.48}$$

Assume $F = \sum_{k=0}^{m-1} f_k/z^{k+1}$. Write

$$\frac{z^{n-1}}{q} = \sum_{k=0}^{\infty} \frac{a_k}{z^{k+1}}. \tag{5.49.a}$$

Then

$$\langle F \,|\, \frac{z^i}{q}\rangle = \sum_k f_k a_{k+1-n-i}. \tag{5.49.b}$$

This scalar product is the coefficient of z^i in Y. We have to compute a_k for $0 \le k < m$, and the products in (5.49.b). This gives a cost of $2mn$. As explained above, computing D_i for each i costs mn, and the scalar products $\langle D_i \,|\, Y\rangle$ costs n. Thus, we have a total cost of $3mn + n^2$ for ψ'.

If we differentiate (5.48) we get

$$d\partial\psi/2 = \langle dD_i \,|\, Y\rangle + \langle D_i \,|\, dY\rangle. \tag{5.50}$$

If we differentiate (5.47) we get

$$\langle F \,|\, \frac{Ddq}{q^2}\rangle = \langle \tilde{D} \,|\, dY\rangle, \qquad \forall D.$$

If Z is such that

$$\langle F \,|\, \frac{D}{q^2}\rangle = \langle \tilde{D} \,|\, Z\rangle, \qquad \deg D < 2n$$

then

$$\langle z^i \,|\, \frac{\partial Y}{\partial q_j}\rangle = \langle z^{n-i-j} \,|\, Z\rangle,$$

so that dY is trivially obtained if we know Z. The quantity Z is computed exactly like Y. It costs $4mn + n^2$. We already computed the cost of dD_i: it is $d\partial R_1$ in (5.45.a). Hence the total cost of ψ'' is

$$5mn + n^2 - 2m + n^2(n+1). \tag{5.51}$$

5.3 Matrix case

We make the following assumptions here: the size of G is $p' \times p$, and for simplicity, we assume $p' = p$. The McMillan degree of the result is n. The degree of the elements in G is $< m$. All entries in G are padded so that there are exactly m coefficients in each polynomial. For numerical applications, we shall assume $m = 400$.

There are np entries in y. This gives np columns in ψ'' in the real case, and $2np$ columns in the complex case. What we have to do is compute the cost of formulas (4.4) and (4.18).

5.3.1 Cost of ψ

We consider here formulas (4.18):

$$G = V_1 q + R_1 \qquad R_1\tilde{D} = V_2 q + R_2,$$

$$\psi = \|F\|^2 - \|R_1\|^2 + \|V_2\|^2.$$

The first equation is $G = V_1 q + R_1$. We have to divide a $p \times p$ matrix of degree $m-1$ by a polynomial of degree n. For some reasons, we do not use the fact that G is constant, so that the complexity is the same as if G were variable.

	ψ	ψ'	ψ''
real	$p^2(m-n)(n+1)$	$2p^2(m-n)(n+1)$	$p^3n(m-n)(6n+7)$
complex	$4p^2(m-n)(n+1)$	$8p^2(m-n)(n+1)$	$8p^3n(m-n)(6n+7)$

From now on, everything depends only on n and p. We have now to multiply R_1, a $p \times p$ matrix of degree $< n$ by $\tilde{D}$, which is a $p \times p$ matrix of degree n. The cost is the following.

	ψ	ψ'	ψ''
real	$p^3n(n+1)$	$2p^3n(n+1)$	$6p^4n^2(n+1)$
complex	$4p^3n(n+1)$	$8p^3n(n+1)$	$48p^4n^2(n+1)$

We have now to divide $P = R_1\tilde{D}$ by q. The cost is the same as the previous division, with m replaced by $2n$.

	ψ	ψ'	ψ''
real	$p^2n(n+1)$	$2p^2n(n+1)$	$p^3n^2(6n+7)$
complex	$4p^2n(n+1)$	$8p^2n(n+1)$	$8p^3n^2(6n+7)$

Finally, we have to compute $\psi = \|G\|^2 - \|R_1\|^2 + \|V_2\|^2$. Essentially, we compute the square of the modules of $2np^2$ terms.

	ψ	ψ'	ψ''
real	$2np^2$	$2np^2$	$6n^2p^3$
complex	$4np^2$	$4np^2$	$24n^2p^3$

If we put all these quantities together, the complexity of the function is asymptotically

$$C(\psi) = p^2(n+1)(m+pn).$$

In table 5.18 we give the cost of computing ψ and its derivatives. This cost does not include the cost of computing the Schur parameters.

5.3.2 Cost of the Schur parameters

We compute here the complexity of the following formulas, as implemented in chapter 4.

$$q_B = (b - \tilde{b}\|y\|^2)q_A + (\tilde{b} - b)y^*\tilde{D}_A u.$$

$$\tilde{D}_B = (b - \tilde{b}\|y\|^2)\tilde{D}_A + (\tilde{b} - b)[\tilde{D}_A uu^* + yy^*\tilde{D}_A - y^*uq_A] + \frac{b-\tilde{b}}{q_A}[\tilde{D}_A uy^*\tilde{D}_A - y^*\tilde{D}_A u\tilde{D}_A].$$

We start by computing $s = \|y\|^2$. The complexity is the following.

	ψ	ψ'	ψ''
real	np	np	$3n^2p^2$
complex	$2np$	$2np$	$12n^2p^2$

We compute then $b - \tilde{b}\|y\|^2$. Here $\tilde{b}$ is constant and has two coefficients.

	ψ	ψ'	ψ''
real	$2n$	$2n$	$4n^2p$
complex	$4n$	$4n$	$16n^2p$

In the case $p = 2$, we have to compute $Z = \tilde{q}(y^*u - uy^*)$. The product y^*u costs p, and the product uy^* costs p^2. The cost of $y^*u - uy^*$ is hence the following:

Table 5.18: Cost of ψ and its derivatives, real and complex case. We have $p = 2$ and $p = 4$.

n	Real ψ	Real ψ'	Real ψ''	Complex ψ	Complex ψ'	Complex ψ''
1	3 224	6 440	41 840	12 880	25 744	334 528
2	4 864	9 712	122 944	19 424	38 816	982 784
3	6 520	13 016	243 888	26 032	52 016	1 949 376
4	8 192	16 352	405 248	32 704	65 344	3 238 912
5	9 880	19 720	607 600	39 440	78 800	4 856 000
6	11 584	23 120	851 520	46 240	92 384	6 805 248
7	13 304	26 552	1 137 584	53 104	106 096	9 091 264
8	15 040	30 016	1 466 368	60 032	119 936	11 718 656
9	16 792	33 512	1 838 448	67 024	133 904	14 692 032
10	18 560	37 040	2 254 400	74 080	148 000	18 016 000
1	12 960	25 888	336 256	51 776	103 488	2 688 512
2	19 648	39 232	992 768	78 464	156 800	7 936 000
3	26 464	52 832	1 978 752	105 664	211 136	15 816 192
4	33 408	66 688	3 303 424	133 376	266 496	26 402 816
5	40 480	80 800	4 976 000	161 600	322 880	39 769 600
6	47 680	95 168	7 005 696	190 336	380 288	55 990 272
7	55 008	109 792	9 401 728	219 584	438 720	75 138 560
8	62 464	124 672	12 173 312	249 344	498 176	97 288 192
9	70 048	139 808	15 329 664	279 616	558 656	122 512 896
10	77 760	155 200	18 880 000	310 400	620 160	150 886 400

Table 5.19: Complexity of $Z = \tilde{q}(y^*u - uy^*)$ in the case $p = 2$

n	Real ψ	Real ψ'	Real ψ''	Complex ψ	Complex ψ'	Complex ψ''
1	10	14	72	40	56	480
2	18	30	336	72	120	2 496
3	30	54	936	120	216	7 200
4	46	86	2 016	184	344	15 744
5	66	126	3 720	264	504	29 280
6	90	174	6 192	360	696	48 960
7	118	230	9 576	472	920	75 936
8	150	294	14 016	600	1 176	111 360
9	186	366	19 656	744	1 464	156 384
10	226	446	26 640	904	1 784	212 160

	ψ	ψ'	ψ''
real	6	6	$24n$
complex	24	24	$96n$

At iteration I, the polynomial q has degree I. The cost of multiplication of y^*u-uy^* by $\tilde{q}$ is $(I+1)p^2$. This gives a total of $p^2n(n+1)/2$.

	ψ	ψ'	ψ''
real	$2n(n+1)$	$4n(n+1)$	$24n^2(n+1)$
complex	$8n(n+1)$	$16n(n+1)$	$192n^2(n+1)$

Table 5.20: Cost of Y, in the case $p = 4$.

n	Real ψ	Real ψ'	Real ψ''	Complex ψ	Complex ψ'	Complex ψ''
1	64	112	1 728	256	448	13 824
2	288	528	14 976	1 152	2 112	119 808
3	768	1 440	58 752	3 072	5 760	470 016
4	1 600	3 040	161 280	6 400	12 160	1 290 240
5	2 880	5 520	360 000	11 520	22 080	2 880 000
6	4 704	9 072	701 568	18 816	36 288	5 612 544
7	7 168	13 888	1 241 856	28 672	55 552	9 934 848
8	10 368	20 160	2 045 952	41 472	80 640	16 367 616
9	14 400	28 080	3 188 160	57 600	112 320	25 505 280
10	19 360	37 840	4 752 000	77 440	151 360	38 016 000

In table 5.19, we have put the total complexity of Z, in the case $p = 2$. Let's now compute the complexity of Y. This is the same quantity as above, but in the case $p \neq 2$. Note that if $p = 1$, this expression is zero. In general it is:

$$\frac{\tilde{D}_A u y^* \tilde{D}_A - y^* \tilde{D}_A u \tilde{D}_A}{q_A}$$

We assume that y_D contains $y^* \tilde{D}_A$ and that E contains $y^* \tilde{D}_A u$. Thus, we have to divide $Duy_D - ED$ by q.

The complexity of the product Du is $p^2(I+1)$, because we multiply a matrix of size $p \times p$ and degree I by a vector of size $p \times 1$ of degree zero. The result is a vector of size $p \times 1$, of degree I. Since we multiply it by a vector of size $1 \times p$, of degree I, the cost is $p^2(I+1)^2$. Now, we multiply the scalar E, of degree I by the $p \times p$ matrix D of degree I. This costs also $p^2(I+1)^2$. Finally, we have to divide this all by q. The division costs also $p^2(I+1)^2$.

In the next table, α is the sum of $(I+1)^2$, this is $n(n+1)(2n+1)/6$. We take into account the fact that u is constant in the multiplication of D by u.

	ψ	ψ'	ψ''
real	$3p^2\alpha + p^2n(n+1)/2$	$6p^2\alpha + p^2n(n+1)/2$	$[18p^2\alpha + 7p^2n(n+1)/2]np$
complex	$12p^2\alpha + 2p^2n(n+1)$	$24p^2\alpha + 2p^2n(n+1)$	$8[18p^2\alpha + 7p^2n(n+1)/2]np$

In table 5.20 we give the cost of Y in the case $p = 4$. This is the equivalent of table 5.19, which is valid only if $p = 2$. The cost of Y is essentially

$$C(Y) = p^2n^3.$$

We have now to compute $X = Duu^* + yy^*D - y^*uq$. In the next table, we start with three products: there is the product $y_D = y^*D$; this has a cost of $p^2(I+1)$. We multiply this by u in order to get $E = y^*Du$; it has a cost of $p(I+1)$. Finally, we compute $N = yu^*$; this has a cost of p^2. Note that u is a constant. We obtain the following result.

	ψ	ψ'	ψ''
real	$(p^2+p)n(n+1)/2 + np^2$	$(2p^2+p)n(n+1)/2 + np^2$	$[(p+3p^2)n(n+1) + 2np^2]np$
complex	$2(p^2+p)n(n+1) + 4np^2$	$2(2p^2+p)n(n+1) + 4np^2$	$8[(p+3p^2)n(n+1) + 2np^2]np$

The complexity of X is the following. We have to compute Duu^*. This costs $p^3(I+1)$ (we assume that uu^* is already computed). Then we compute yy^*D, product of y by y_D; this costs $p^2(I+1)$. Finally, we compute Nq, this costs also $p^2(I+1)$.

Table 5.21: Complexity of X, for $p = 2$ and $p = 4$

n	Real ψ	Real ψ'	Real ψ''	Complex ψ	Complex ψ'	Complex ψ''
1	30	42	220	112	160	1 680
2	82	118	1 248	312	456	9 664
3	156	228	3 636	600	888	28 368
4	252	372	7 936	976	1 456	62 208
5	370	550	14 700	1 440	2 160	115 600
6	510	762	24 480	1 992	3 000	192 960
7	672	1 008	37 828	2 632	3 976	298 704
8	856	1 288	55 296	3 360	5 088	437 248
9	1 062	1 602	77 436	4 176	6 336	613 008
10	1 290	1 950	104 800	5 080	7 720	830 400
1	138	186	1 888	540	732	14 848
2	392	536	10 944	1 544	2 120	86 528
3	762	1 050	32 256	3 012	4 164	255 744
4	1 248	1 728	70 912	4 944	6 864	563 200
5	1 850	2 570	132 000	7 340	10 220	1 049 600
6	2 568	3 576	220 608	10 200	14 232	1 755 648
7	3 402	4 746	341 824	13 524	18 900	2 722 048
8	4 352	6 080	500 736	17 312	24 224	3 989 504
9	5 418	7 578	702 432	21 564	30 204	5 598 720
10	6 600	9 240	952 000	26 280	36 840	7 590 400

	ψ	ψ'	ψ''
real	$(p^3 + 2p^2)n(n + 1)/2$	$(p^3 + 4p^2)n(n + 1)/2$	$(p^3 + 6p^2)n^2(n + 1)p$
complex	$2(p^3 + 2p^2)n(n + 1)$	$2(p^3 + 4p^2)n(n + 1)$	$8(p^3 + 6p^2)n^2(n + 1)p$

Let's now compute D. We have to multiply D by b_4. This costs $2p^2(I + 1)$. We have to compute b_2Z. This has the same cost, but b_2 is constant. We are in the same case for b_2X. For q, we have to compute $b_4q - b_2E$. Each term costs $2(I + 1)$, but b_2 is still constant.

	ψ	ψ'	ψ''
real	$(3p^2 + 2)n(n + 1)$	$(4p^2 + 3)n(n + 1)$	$(10p^2 + 8)n^2(n + 1)p$
complex	$4(3p^2 + 2)n(n + 1)$	$4(4p^2 + 3)n(n + 1)$	$8(10p^2 + 8)n^2(n + 1)p$

5.3.3 Memory requirements

Since all variables are declared in table 4.15 on page 146, computation of memory space is easy. The cost of inner variables (used by the Schur algorithm) is

$$(5p^2 + 2p + 2)n(n + 1)/2 + (2p^2 + p + 13)n + p^2 + 1$$

We have to add to this quantity the memory cost for u and uu^*, namely $n(1 + p^2)$, but no memory is needed for the derivative of it. The variables needed for computing ψ need $p^2(m + 4n)$. Note that, if G is not a square matrix, but of size $q \times p$, this has to be replaced by $qp(m + 4n)$. Results are given in table 5.22.

Note that the space for ψ' is more or less the space for ψ. If this quantity is N, then we allocate $3N$ objects (for ψ, ψ' and ψ''). Each object is a number in double and quadruple precision. Hence $84N$ bytes are allocated. In the complex case twice as many memory is needed. In the case $p = 4$, $n = 8$, this is $1.8M$. The total memory allocated by HYPERION is $4.3M$.

Table 5.22: Space complexity for $p = 2$ and $p = 4$

n	ψ	ψ'	ψ	ψ'
1	1675	5	6637	6620
2	1771	10	6947	6913
3	1893	15	7347	7296
4	2041	20	7837	7769
5	2215	25	8417	8332
6	2415	30	9087	8985
7	2641	35	9847	9728
8	2893	40	10697	10561
9	3171	45	11637	11484
10	3475	50	12667	12497
11	3805	55	13787	13600
12	4161	60	14997	14793
13	4543	65	16297	16076
14	4951	70	17687	17449
15	5385	75	19167	18912
16	5845	80	20737	20465
17	6331	85	22397	22108
18	6843	90	24147	23841
19	7381	95	25987	25664
20	7945	100	27917	27577

The number of multiplications needed is the sum of all quantities given above. At first approximation it is:

$$C = p^2 n(m + 3pn/2 + n^2).$$

Note that, in general, m is greater than $3pn/2 + n^2$, in practice, the ratio is between 2 and 4. Thus, it could be interesting to truncate first G with $m/2$ coefficients, find the best approximation of this, and use this result as starting condition for the approximation of G with all its coefficients. Such an algorithm is not yet implemented in HYPERION.

In the case $p = 2$, the complexity can be approximated by $35n^2 + 4mn$ for ψ, $56n^2 + 8mn$ for ψ' and $308n^3 + 48n^2m$ for ψ''. For large n, the quotient ψ'/ψ is 1.6 (recall that we have a theorem that says that this is bounded by 2, see results on figure 5.3.3).

In the case $p > 2$, and n large, the complexity comes essentially from Y. It is p^2n^3. In this case the ratio ψ'/ψ has 2 as limit. This means essentially that the number of terms in our sums that are products of a constant and a variable is small against the number of terms where each factor is variable.

In the table 5.23, we give the exact total cost in the cases $p = 2$, $p = 3$ and $p = 4$. We give also on figure 5.3, 5.4 and 5.5 the values of the cost. We take $p = 2$, $p = 3$, $p = 4$, $p = 5$ and $p = 6$. On the y-axis we have put the logarithm of the time.

5.4 Derivatives in direct mode

In this section, we shall compute the cost of ψ' and ψ'' if the derivatives are computed in direct mode. We assume that we have enough memory, so that everything will be stored, rather then recomputed. For simplicity, we consider only the real case: in the complex case, we have to multiply the complexity of ψ' and ψ'' by a factor which is approximatively 8 or 16. In all tables, we show some values for $p = 2$ and $p = 10$. More values of p will be used for the figures.

Table 5.23: Global complexity, $p = 2$, $p = 3$ and $p = 4$

n	Real ψ	Real ψ'	Real ψ''	Complex ψ	Complex ψ'	Complex ψ''
1	3 292	6 534	42 324	13 144	26 112	338 224
2	5 048	9 974	125 680	20 144	39 848	1 004 160
3	6 874	13 526	251 916	27 424	54 032	2 012 592
4	8 770	17 190	422 880	34 984	68 664	3 378 304
5	10 736	20 966	640 420	42 824	83 744	5 116 080
6	12 772	24 854	906 384	50 944	99 272	7 240 704
7	14 878	28 854	1 222 620	59 344	115 248	9 766 960
8	17 054	32 966	1 590 976	68 024	131 672	12 709 632
9	19 300	37 190	2 013 300	76 984	148 544	16 083 504
10	21 616	41 526	2 491 440	86 224	165 864	19 903 360
1	7 437	14 765	143 610	29 702	59 014	1 148 076
2	11 533	22 771	431 094	46 040	90 992	3 445 536
3	15 942	31 326	877 149	63 630	125 166	7 009 956
4	20 718	40 538	1 500 360	82 688	161 968	11 990 016
5	25 915	50 515	2 323 200	103 430	201 830	18 565 500
6	31 587	61 365	3 372 030	126 072	245 184	26 947 296
7	37 788	73 196	4 677 099	150 830	292 462	37 377 396
8	44 572	86 116	6 272 544	177 920	344 096	50 128 896
9	51 993	100 233	8 196 390	207 558	400 518	65 505 996
10	60 105	115 655	10 490 550	239 960	462 160	83 844 000
1	13 262	26 320	341 216	52 972	105 204	2 727 936
2	20 628	40 698	1 026 752	82 360	162 640	8 206 848
3	28 594	56 126	2 093 952	114 148	224 276	16 735 488
4	37 256	72 796	3 589 376	148 720	290 880	28 686 336
5	46 710	90 900	5 568 800	186 460	363 220	44 505 600
6	57 052	110 630	8 097 216	227 752	442 064	64 713 216
7	68 378	132 178	11 248 832	272 980	528 180	89 902 848
8	80 784	155 736	15 107 072	322 528	622 336	120 741 888
9	94 366	181 496	19 764 576	376 780	725 300	157 971 456
10	109 220	209 650	25 323 200	436 120	837 840	202 406 400

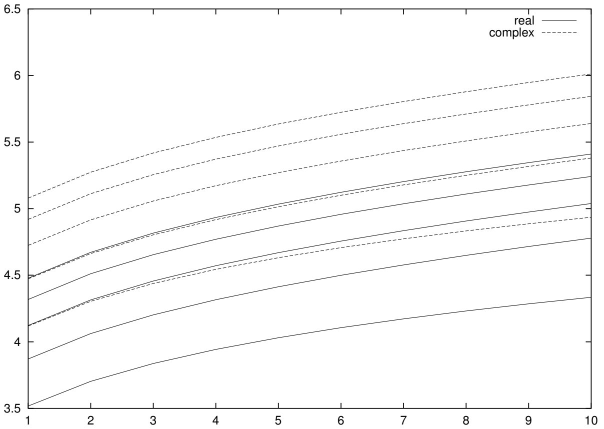

Figure 5.3: Complexity of the function

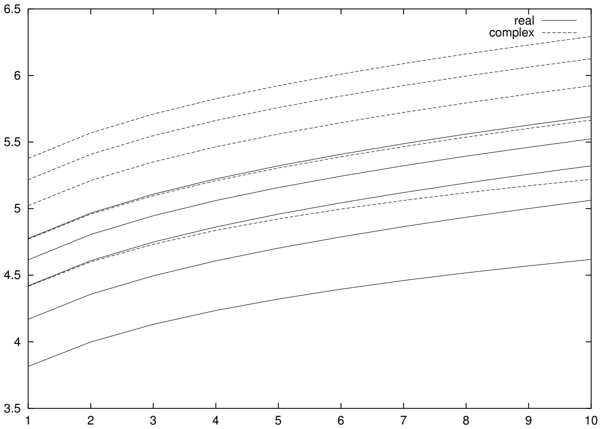

Figure 5.4: Complexity of the first derivative

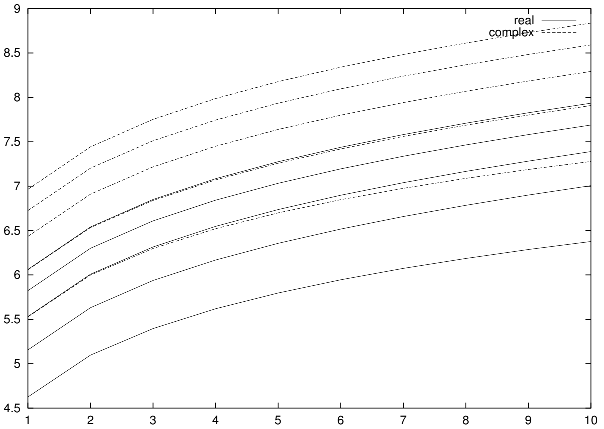

Figure 5.5: Complexity of the second derivative. On each curve p is constant ($2 \leq p \leq 6$). On the x-axis we have n, on the y-axis we have $\log_{10}(T)$, the base-ten logarithm of time complexity. Computations are done for the algorithm described in chapter 4.

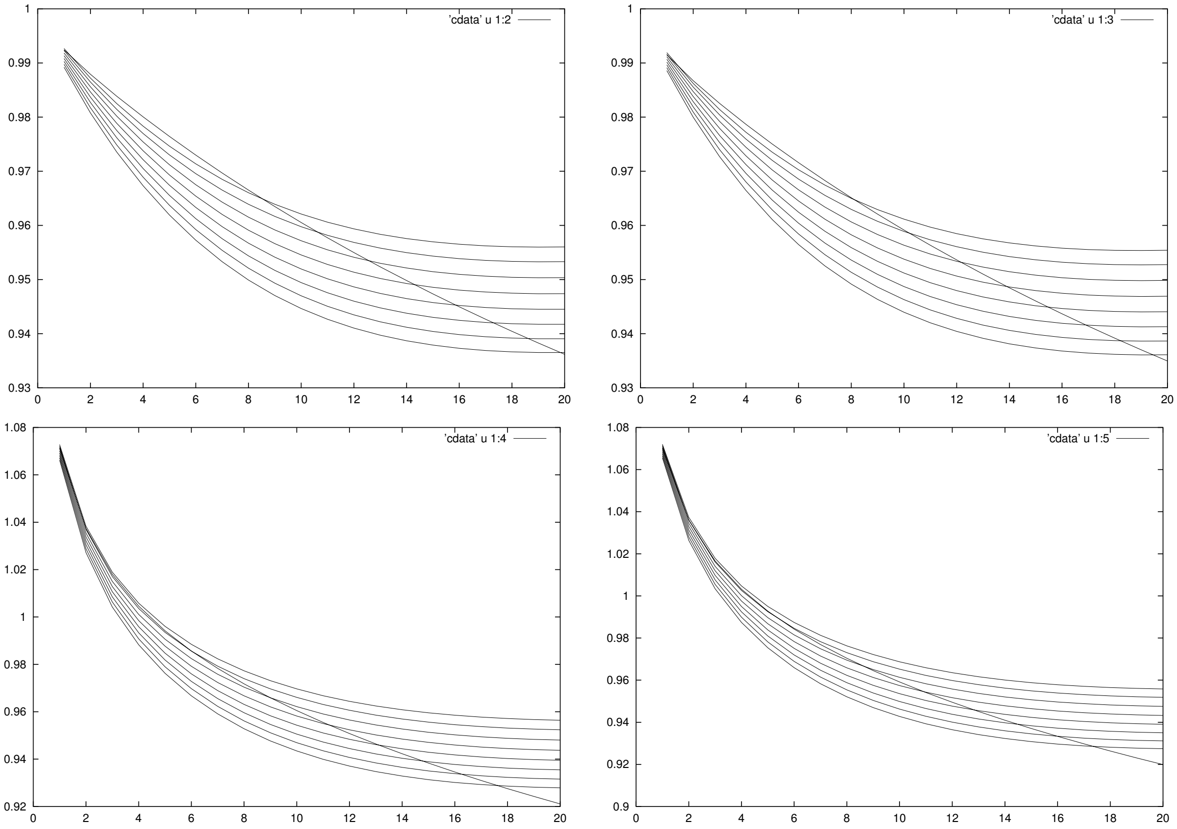

Figure 5.6: Normalised cost: upper left, cost of $\psi'_r/2\psi_r$; upper right, cost of $\psi'_c/8\psi_r$; lower left, $\psi''_r/6np\psi_r$; lower right $\psi''_c/48np\psi_r$. Here ψ_r is the cost of ψ in the real case, ψ'_r and ψ''_r the cost of the first and second derivatives of ψ; ψ'_c and ψ''_c are the cost of the first and second derivatives of ψ in the complex case. We have $2 \leq n \leq 20$, $2 \leq p \leq 10$. In principle the value should be one. Note the special case $p = 2$.

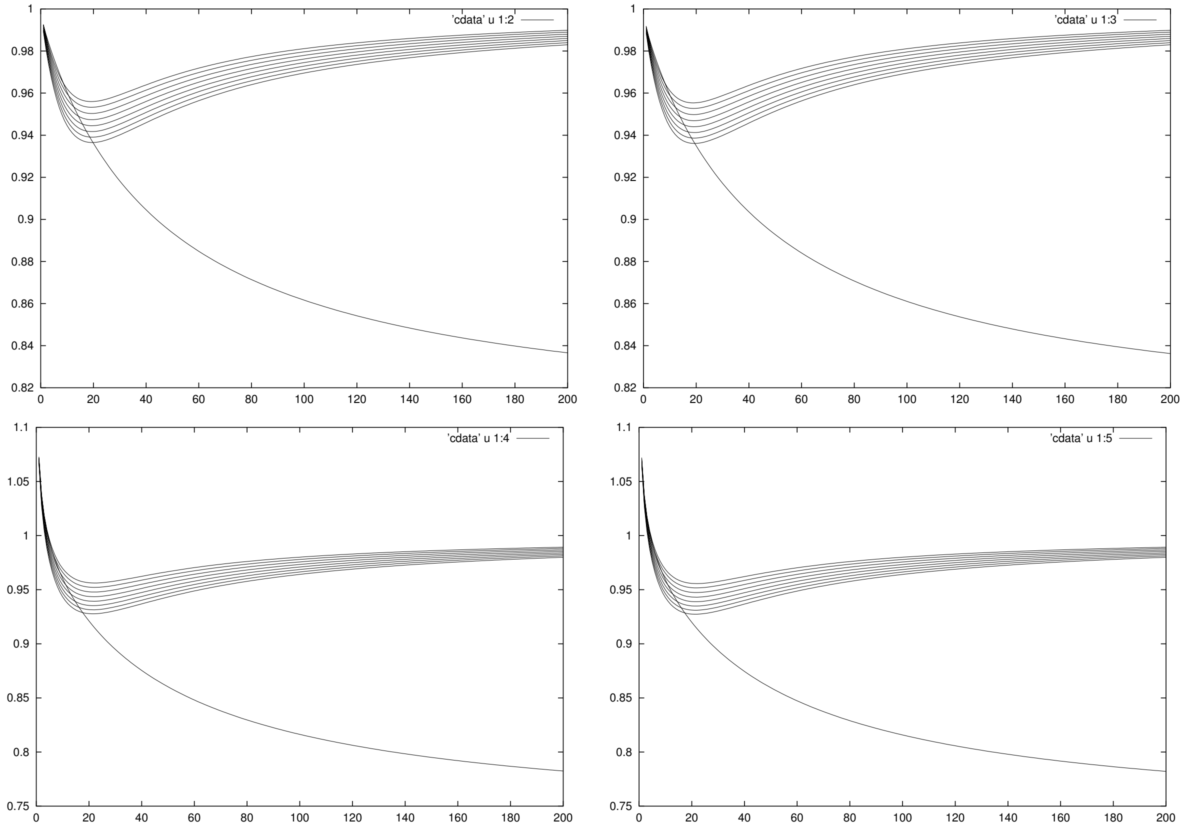

Figure 5.7: Normalised cost: here n ranges from 1 to 200. The limits for $n = \infty$ are 1 in the general case. In the case $p = 2$, the limit is 0.8 for the derivative, and 0.733 for the second derivative.

Recall that the cost of the product AB, where A and B are matrices of size (n_1, n_2) and (n_2, n_3) and degree p and q is

$$n_1 n_2 n_3 (p+1)(q+1).$$

If C has size (n_3, n_4) and degree s, then the cost of the product ABC is

$$T[A(BC)] = n_2 n_4 [n_1(p+1)(q+s+1) + n_3(q+1)(s+1)],$$

$$T[(AB)C] = n_1 n_3 [n_2(p+1)(q+1) + n_4(p+q+1)(s+1)].$$

The first computation is the best if

$$\frac{(p+1)(q+s+1)}{n_3} + \frac{(q+1)(s+1)}{n_1} < \frac{(p+1)(q+1)}{n_4} + \frac{(p+q+1)(s+1)}{n_2}.$$

For instance, if the matrices are of degree zero, the condition becomes:

$$\frac{1}{n_1} + \frac{1}{n_3} < \frac{1}{n_2} + \frac{1}{n_4}.$$

In the case where $p = q = s$ is large, the condition becomes

$$\frac{1}{n_1} + \frac{2}{n_3} < \frac{2}{n_2} + \frac{1}{n_4}.$$

These formulas show that it is not obvious a priori to say which is the best order of computation. Note that, if $n_1 = n_4$ (case where the result is a square matrix), in these two cases, the conditions simplify to $n_2 < n_3$.

In what follows, we consider $B = T_\Theta(A)$, where A is of degree i, and we differentiate this with respect to y_j and y_k. We assume that y_j and y_k appear in Θ_J and Θ_K. For the Hessian, we may always assume $k \le j$, hence $K \le J$ (recall that the Hessian is a symmetric matrix). This has the following consequences:

- If $J > i+1$, then B is independent of y_j.
- If $J = i+1$, then A is independent of y_j and Θ depends on y_j.
- If $J < i+1$, then Θ is independent of y_j, but A depends on y_j.

We shall make a heavy use of the following formulas.

$$\sum_{i<n} i = \frac{n(n-1)}{2}.$$

$$\sum_{i<n} i^2 = \frac{n(n-1)(2n-1)}{6}.$$

$$\sum_{i<n} i^3 = \frac{n^2(n-1)^2}{4}.$$

$$\sum_{i<n} i(i+1) = \frac{n(n-1)(n+1)}{3}.$$

$$\sum_{i<n} (i+1)ip(ip+1) = \frac{pn(n-1)(n+1)}{3}(1 + \frac{p(3n-2)}{4}).$$

5.4.1 Pseudo-code

The polynomial q is defined

$$q_B = (b - \tilde{b}\|y\|^2)q_A + (\tilde{b} - b)y^*\tilde{D}_A u. \tag{$*$}$$

In order to make explicit the equations for which we compute the complexity, we shall give an implementation in pseudo-code.

In equation $(*)$, we assume that the quantities q_A and $\tilde{D}_A$ are in $q(i)$ and $D(i)$, so that what we have to compute is $q(i+1)$. We assume that $b - \tilde{b}\|y\|^2$ and $\tilde{b} - b$ have been computed and stored in $B_4(i)$ and $B_2(i)$ (we use here the same notations as in the previous chapter). The pseudo-code associated to $(*)$ is

$$\forall i : \quad q(i+1) = B_2(i) * q(i) + B_4(i) * (y_t(i) * (D(i) * u(i))).$$

In fact, the pseudo-code will be more complicated than that, because we have to indicate the storage used for every operation. Assume that we put $\tilde{D}_A u$ in D_u and $y^*\tilde{D}_A u$ in D_{uy}. Then the pseudo-code will be

$$\forall i : \quad D_u(i) = D(i) * u(i),$$

$$\forall i : \quad D_{uy}(i) = y_t(u) * D_u(i),$$

$$\forall i : \quad q(i+1) = B_4(i) * D_{uy}(i),$$

$$\forall i : \quad q(i+1) \mathrel{+}= b_2(i) * q(i).$$

It will always be understood that the quantification $\forall i$ means: for all i from 0 up to $n-1$. The derivative of the first part of $(*)$ looks like

$$\forall i, j, (J \le i) : \quad dq(i,j) = B_4(i) * dq(i,j),$$

$$\forall i, j, (J = i+1) : \quad dq(i,j) = -T_1(i) * 2y_j.$$

Here $dq(i,j)$ is the derivative of $q_A = q(i)$, with respect to y_j, which is one component of the vector y. In the first case, we assume $J \le i$, and in the second case $J = i+1$. The derivative is zero in case $J > i+1$. The quantity T_1 in the second equation is $\tilde{b}q_A$. The code of the second derivative has the form

$$\forall i, j, k, (K \le J \le i) : \quad ddq(i+1, j, k) = B_4(i) * ddq(i, j, k).$$

Here $ddq(i,j,k)$ is the second derivative of q_A in the directions y_j and y_k. Since this is symmetric, we compute only one half of the Hessian. Thus we assume $k \le j$. This implies $K \le J$.

5.4.2 Sample formula

Assume that we have an expression of the form

$$B = Ay, \tag{5.52.a}$$

where A depends on y_j for $J \le i$ and y depends on y_j, $J = i+1$. We have

$$\frac{\partial B}{\partial y_j} = y\frac{\partial A}{\partial y_j}, \qquad J \le i, \tag{5.52.b}$$

$$\frac{\partial B}{\partial y_j} = \frac{\partial y}{\partial y_j}A, \qquad J = i+1. \tag{5.52.c}$$

Let $T(A'_i)$ be the complexity of equation (5.52.b) and $T(A_i)$ the complexity of (5.52.c). We assume that these complexities depend only on i. Let's denote by $T(B')$ and $T(B'')$ the complexity of the complete first and second derivatives of B.

There are ip vectors y_j such that $J \le i$ and p vectors such that $J = i+1$. Hence

$$T(B') = p\sum iT(A'_i) + p\sum T(A_i). \tag{5.52.d}$$

Let's now compute the complexity of the Hessian. Since $k \le j$, we have $K \le J$. So

$$\frac{\partial^2 B}{\partial y_j \partial y_k} = y\frac{\partial^2 A}{\partial y_j \partial y_k}, \qquad J \le i.$$

The number of vectors y_j, y_k such that $k \le j$ and $J \le i$ is $ip(ip+1)/2$. We denote by $T(A''_i)$ the complexity of one of these terms. If we differentiate now (5.52.c), the result will be zero if $K = J$. In some cases, it can be non-zero, but only if y appears non-linearly in the expression. There remains

$$\frac{\partial^2 B}{\partial y_j \partial y_k} = \frac{\partial y}{\partial y_j}\frac{\partial A}{\partial y_k}, \qquad J = i+1, K \le i.$$

Let's denote by $T(A'_i)$ the cost of such an expression. For each i, there are ip vectors y_k with $K \le i$ and p vectors y_j with $J = i$. Hence

$$T(B'') = \sum ip(ip+1)T(A''_i)/2 + p^2\sum iT(A'_i). \tag{5.52.e}$$

5.4.3 Notations

We have to implement

$$q_B = (b - \tilde{b}\|y\|^2)q_A + (\tilde{b} - b)y^*\tilde{D}_A u. \tag{5.53.a}$$

$$\tilde{D}_B = (b - \tilde{b}\|y\|^2)\tilde{D}_A + (\tilde{b} - b)[\tilde{D}_A uu^* + yy^*\tilde{D}_A - yu^*q_A - \frac{\tilde{D}_A uy^*\tilde{D}_A - y^*\tilde{D}_A u\tilde{D}_A}{q_A}]. \tag{5.53.b}$$

The computation is done by introducing some variables (main variables)

$$X_1 = (b - \tilde{b}\|y\|^2)q_A, \tag{5.54.a}$$

$$X_2 = (\tilde{b} - b)y^*\tilde{D}_A u, \tag{5.54.b}$$

$$X_3 = (b - \tilde{b}\|y\|^2)\tilde{D}_A, \tag{5.54.c}$$

$$X_4 = \tilde{D}_A uu^*, \tag{5.54.d}$$

$$X_5 = yy^*\tilde{D}_A, \tag{5.54.e}$$

$$X_6 = yu^*q_A, \tag{5.54.f}$$

$$X_7 = \tilde{D}_A uy^*\tilde{D}_A, \tag{5.54.g}$$

$$X_8 = y^*\tilde{D}_A u\tilde{D}_A, \tag{5.54.h}$$

$$X_9 = X_7 - X_8, \tag{5.54.i}$$

$$X = X_9/q_A, \tag{5.54.j}$$

$$Z = (\tilde{b} - b)(X_4 + X_5 - X_6 - X). \tag{5.54.k}$$

With these notations, $q_B = X_1 + X_2$ and $\tilde{D}_B = X_3 + Z$.

We introduce other variables (temporaries)

$$B_4 = b - \tilde{b}\|y\|^2, \tag{5.55.a}$$

$$B_2 = \tilde{b} - b, \tag{5.55.b}$$

$$T_1 = \tilde{b}q_A, \tag{5.55.c}$$

$$T_2 = \tilde{b}\frac{\partial \|y\|^2}{\partial y_j}, \tag{5.55.d}$$

$$T_3 = (\tilde{b} - b)y, \tag{5.55.e}$$

$$T_4 = \frac{\partial}{\partial y_j}\tilde{D}_A u, \tag{5.55.f}$$

$$T_5 = \frac{\partial^2}{\partial y_j \partial y_k}\tilde{D}_A u, \tag{5.55.g}$$

$$T_6 = \tilde{b}\tilde{D}_A, \tag{5.55.h}$$

$$T_7 = y^*\tilde{D}_A, \tag{5.55.i}$$

$$T_8 = y^*\frac{\partial}{\partial y_j}\tilde{D}_A, \tag{5.55.j}$$

$$T_9 = \tilde{D}_A u, \tag{5.55.k}$$

$$T_{10} = y^*\tilde{D}_A u, \tag{5.55.l}$$

$$T_{11} = y^*\frac{\partial}{\partial y_j}\tilde{D}_A u, \tag{5.55.m}$$

$$T_{12} = yu^*. \tag{5.55.n}$$

In some cases, we write b_t for $\tilde{b}$, and y_t for y^*. A scalar temporary T will also be used.

5.4.4 Space complexity

We have to store q, D and the derivatives of these quantities. This has a cost of $\sum \alpha(i+1)$, where i ranges between 1 and n, and α is 1 for q, ip for q' and $ip(ip+1)/2$ for q''. This gives a total of

$$\frac{(n+1)(n+2)}{2}, \quad p\frac{n(n+1)(n+2)}{3}, \qquad \frac{pn(n+1)(n+2)}{6}(1+p\frac{3n+1}{4}).$$

The memory cost for D is the same, multiplied by p^2.

We shall see later, that for $X = X_9/q_A$, we have to allocate memory for X_9 and its derivatives, and for X and its first derivative. No space is needed for the second derivative of X. The degree of X_9 is $2i$, and the degree of X is i. This gives hence

$$S(X) = p^2 n(n+1)/2, \quad S(X') = p^3(n^3 - n)/3.$$

$$S(X_9) = p^2 n^2, \quad S(X_9') = p^3 n(4n+1)(n-1)/6, \quad S(X_9'') = p^3(n-1)n(3pn^2 - pn + 4n + 1 - p)/12.$$

We need also space for Z_0. This is the same as for D, but with n replaced by $n-1$.

We need space for b, $\tilde{b}$, $b - \tilde{b}$ and $b - \tilde{b}\|y\|^2$. This gives a total of $8p$.

We have to allocate space for the temporary variables, see table 5.24.

This gives a total of

$$S = p^2(n+1)(3n+1) + p(n^2 + 3n + 8) + (n+1)^2. \tag{5.56.a}$$

$$S' = p^3 n\frac{10n^2 + 3n - 1}{6} + p^2\frac{2n(n-1)(n+1)}{3} + p\frac{2n(n+1)(2n+1)}{3} + + \frac{n(n+1)}{2}. \tag{5.56.b}$$

$$S'' = \frac{pn}{24}[p^3(6+6n+12n^3) + p^2(4+3n+14n^2+3n^3) + p(-2+9n+14n^2+3n^3) + 56 + 12n + 4n^2]. \tag{5.56.c}$$

Table 5.24: Space complexity for temporary variables

name	used by	complexity
T_1	ψ'	$n(n+1)/2$
T_2	ψ''	$2np$
T_3	ψ	$2np$
T_4	ψ'	$p^2n(n-1)(n+1)/2$
T_5	ψ''	$p^2n(n-1)(n+1)(1+p(3n-2)/4)/6$
T_6	ψ	$p^2n(n+1)/2$
T_7	ψ	$pn(n+1)/2$
T_8	ψ'	$p^2n(n-1)(n+1)/3$
T_9	ψ	$pn(n+1)/2$
T_{10}	ψ	$n(n+1)/2$
T_{11}	ψ'	$pn(n-1)(n+1)/3$
T_{12}	ψ	np^2

Table 5.25: Space complexity

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	129	179	260	2 289	15 603	103 380
5	516	2 535	7 090	10 116	230 215	3 371 450
8	1 173	10 084	43 224	23 541	925 796	21 564 760
10	1 761	19 495	103 180	35 601	1 796 455	52 311 900
20	6 501	152 690	1 581 860	133 221	14 177 010	828 267 300

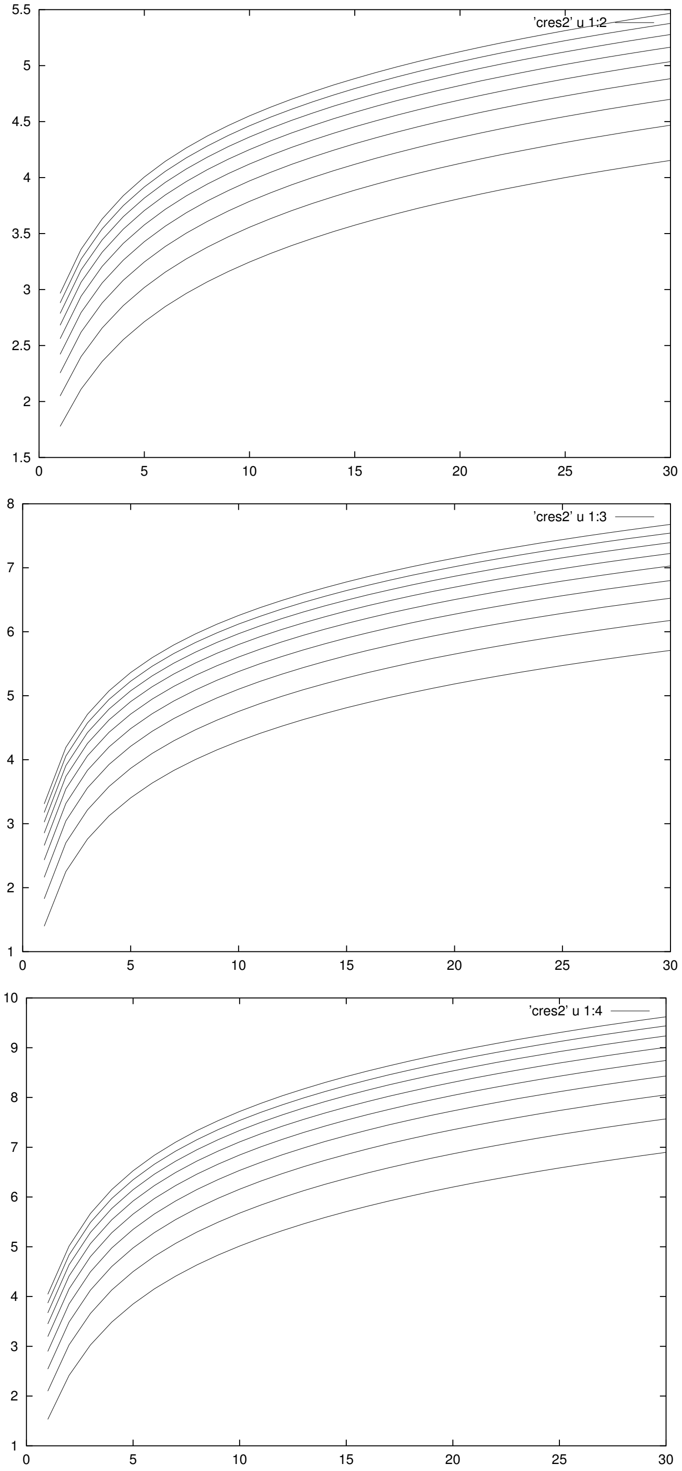

Figure 5.8: Space complexity for direct mode. On each curve p is fixed (between 2 and 10). We give the cost of ψ, ψ' and ψ''

5.4.5 Case of X_1

We have

$$X_1 = (b - \tilde{b}\|y\|^2)q_A, \tag{5.57.a}$$

$$\frac{\partial X_1}{\partial y_j} = (b - \tilde{b}\|y\|^2)\frac{\partial q_A}{\partial y_j}, \quad J \le i, \tag{5.57.b}$$

$$\frac{\partial X_1}{\partial y_j} = -\tilde{b}q_A\frac{\partial \|y\|^2}{\partial y_j}, \quad J = i+1, \tag{5.57.c}$$

$$\frac{\partial^2 X_1}{\partial y_j \partial y_k} = (b - \tilde{b}\|y\|^2)\frac{\partial^2 q_A}{\partial y_j \partial y_k}, \quad K \le J \le i, \tag{5.57.d}$$

$$\frac{\partial^2 X_1}{\partial y_j \partial y_k} = -\tilde{b}q_A\frac{\partial^2 \|y\|^2}{\partial y_j \partial y_k}, \quad K = J = i+1, \tag{5.57.e}$$

$$\frac{\partial^2 X_1}{\partial y_j \partial y_k} = -\tilde{b}\frac{\partial q_A}{\partial y_k}\frac{\partial \|y\|^2}{\partial y_j}, \quad K < J = i+1. \tag{5.57.f}$$

Since $q_B = X_1 + X_2$, we can store X_1 and its derivatives in $q(i+1)$. This gives the following pseudo-code

$$\forall i: \quad q(i+1) = B_4(i) * q(i), \tag{5.58.a}$$

$$\forall i, j \ (J \le i): \quad dq(i+1, j) = B_4(i) * dq(i, j), \tag{5.58.b}$$

$$\forall i: \quad T_1(i) = b_t(i) * q(i), \tag{5.58.c}$$

$$\forall i, j \ (J = i+1): \quad dq(i+1, j) = -T_1(i) * 2y_j, \tag{5.58.d}$$

$$\forall i, j \ (J = i+1): \quad T_2(i, j) = b_t(i) * 2y_j, \tag{5.58.e}$$

$$\forall i, j, k \, (K = J = i+1, k = j): \quad ddq(i+1, j, k) = -2T_1(i), \tag{5.58.f}$$

$$\forall i, j, k \ (K < J = i+1): \quad ddq(i+1, j, k) = -T_2(i, k) * dq(i, j), \tag{5.58.g}$$

$$\forall i, j, k \ (K \le J \le i): \quad ddq(i+1, j, k) = B_4(i) * ddq(i, j, k). \tag{5.58.h}$$

Time complexity: The cost of the product in (a) is $2(i+1)$, hence

$$T(X_1) = n(n+1) + 2np. \tag{5.59.a}$$

The complexity for each multiplication in (b), (c) and (d) is $2(i+1)$, $2(i+1)$ and $i+2$, so that we get $\sum 2(i+1)ip + \sum 2(i+1) + \sum (i+2)p$ and

$$T(X_1') = n(n+1) + pn\frac{5 + 3n + 4n^2}{6}. \tag{5.59.b}$$

The complexity of the multiplication in (e), (f), (g) and (h) is 2, 0, $2(i+1)$ and $2(i+1)$, so that we get $\sum 2p + 2i(i+1)p^2 + (i+1)ip(ip+1)$, hence

$$T(X_1'') = 2(n-1)p + \frac{n(n-1)(n+1)}{3}(p + p^2\frac{3n+6}{4}). \tag{5.59.c}$$

Note that $T(X_1'')$ is zero for $n = 1$. This is because equation (e) is not executed for $i = 1$.

Table 5.26: Cost of X_1 in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	6	24	32	6	96	640
5	30	230	936	30	1 030	21 480
8	72	832	5 404	72	3 872	127 820
10	110	1 560	12 576	110	7 360	300 480
20	420	11 520	180 956	420	55 920	4 415 980

5.4.6 Case of X_2

$$X_2 = (\tilde{b} - b) y^* \tilde{D}_A u, \tag{5.60.a}$$

$$\frac{\partial X_2}{\partial y_j} = (\tilde{b} - b) y^* \frac{\partial \tilde{D}_A}{\partial y_j} u, \quad J \leq i, \tag{5.60.b}$$

$$\frac{\partial X_2}{\partial y_j} = (\tilde{b} - b) \frac{\partial y^*}{\partial y_j} \tilde{D}_A u, \quad J = i + 1, \tag{5.60.c}$$

$$\frac{\partial^2 X_2}{\partial y_j \partial y_k} = (\tilde{b} - b) y^* \frac{\partial^2 \tilde{D}_A}{\partial y_j \partial y_k} u, \quad K \leq J \leq i, \tag{5.60.d}$$

$$\frac{\partial^2 X_2}{\partial y_j \partial y_k} = (\tilde{b} - b) \frac{\partial y^*}{\partial y_j} \frac{\partial \tilde{D}_A}{\partial y_k} u, \quad K < J = i + 1. \tag{5.60.e}$$

Pseudo code

$$\forall i: \quad T_9(i) = D(i) * u(i), \tag{5.61.a}$$

$$\forall i: \quad T_3(i) = B_2(i) * y_t(i), \tag{5.61.b}$$

$$\forall i: \quad q(i+1) = T_3(i) * T_9(i), \tag{5.61.c}$$

$$\forall i, j \ (J \leq i): \quad T_4(i, j) = dD(i, j) * u(i), \tag{5.61.d}$$

$$\forall i, j \ (J \leq i): \quad dq(i+1, j) = T_3(i) * T_4(i, j), \tag{5.61.e}$$

$$\forall i, j \ (J = i+1): \quad dq(i+1, j) = B_2(i) * T_9(i)(j), \tag{5.61.f}$$

$$\forall i, j, k \ (K \leq J \leq i): \quad T_5(i, j, k) = ddD(i, j, k) * u(i), \tag{5.61.g}$$

$$\forall i, j, k \ (K \leq J \leq i): \quad ddq(i+1, j, k) = T_3(i) * T_5(i, j, k), \tag{5.61.h}$$

$$\forall i, j, k \ (K < J = i+1): \quad ddq(i+1, j, k) = B_2(i) * T_4(i, k)(j). \tag{5.61.i}$$

The cost of (a), (b) and (c) is $(i+1)p^2$, $2p$ and $2p^2(i+1)$, hence

$$T(X_2) = \frac{3p^2 n(n+1)}{2} + 2np. \tag{5.62.a}$$

The cost of the multiplication in (d), (e) and (f) is $p^2(i+1)$, $2p^2(i+1)$ and $2(i+1)$, hence

$$T(X_2') = pn(n+1) + p^3 n(n-1)(n+1). \tag{5.62.b}$$

The cost of the multiplications in (g), (h) and (i) is $p^2(i+1)$, $2p^2(i+1)$ and $2(i+1)$, hence

$$T(X_2'') = p^2 n(n-1)(n+1)\left(\frac{2}{3} + \frac{p}{2} + p^2 \frac{3n-2}{8}\right). \tag{5.62.c}$$

Table 5.27: Cost of X_2 in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	44	60	88	940	6 060	33 400
5	200	1 020	3 920	4 600	120 300	2 018 000
8	464	4 176	25 536	10 960	504 720	14 145 600
10	700	8 140	62 040	16 700	991 100	35 211 000
20	2 600	64 680	978 880	63 400	7 984 200	583 072 000

Table 5.28: Cost of X_3 in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	24	104	112	600	10 600	62 000
5	120	928	3 680	3 000	104 000	2 140 000
8	288	3 336	21 504	7 200	388 200	12 768 000
10	440	6 248	50 160	11 000	737 000	30 030 000
20	1 680	46 088	723 520	42 000	5 593 000	441 560 000

5.4.7 Case of X_3

We have

$$X_3 = (b - \tilde{b}\|y\|^2)\tilde{D}_A.$$

This is like X_1, but q_A is replaced by $\tilde{D}_A$, and we use other temporaries. Note that $\tilde{b}\partial\|y\|^2/\partial y_j$ is already computed. The result is stored in $D(i+1)$. Thus we get

$$T(X_3) = p^2 n(n+1). \tag{5.63.a}$$

$$T(X_3') = p^2[n(n+1) + p\frac{(n+1)(n+2)}{2} + \frac{2pn(n-1)(n+1)}{3}]. \tag{5.63.b}$$

$$T(X_3'') = p^3\frac{n(n-1)(n+1)}{3}(1 + p\frac{3n+6}{4}). \tag{5.63.c}$$

5.4.8 Case of X_4

$$X_4 = \tilde{D}_A uu^*, \tag{5.64.a}$$

$$\frac{\partial X_4}{\partial y_j} = \frac{\partial \tilde{D}_A}{\partial y_j} uu^*, \tag{5.64.b}$$

$$\frac{\partial^2 X_4}{\partial y_j \partial y_k} = \frac{\partial^2 \tilde{D}_A}{\partial y_j \partial y_k} uu^*. \tag{5.64.c}$$

We put X_4, X_5, X_6 and X in Z_0. The pseudo code is

$$\forall i: \quad Z_0(i) = T_9(i) * u(i), \tag{5.65.a}$$

$$\forall i, j\ (J \le i): \quad dZ_0(i+1, j) = T_4(i, j) * u(i), \tag{5.65.b}$$

$$\forall i, j, k\ (K \le J \le i): \quad ddZ_0(i+1, j, k) = T_5(i, j, k) * u(i). \tag{5.65.c}$$

Table 5.29: Cost of X_4 in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	12	16	24	300	2 000	11 000
5	60	320	1 200	1 500	40 000	670 000
8	144	1 344	8 064	3 600	168 000	4 704 000
10	220	2 640	19 800	5 500	330 000	11 715 000
20	840	21 280	319 200	21 000	2 660 000	194 180 000

Each multiplication in (a), (b) and (c) costs $p^2(i+1)$ hence

$$T(X_4) = p^2 n(n+1)/2. \tag{5.66.a}$$

$$T(X_4') = p^3 \frac{n(n-1)(n+1)}{3}. \tag{5.66.b}$$

$$T(X_4'') = p^3 \frac{n(n-1)(n+1)}{6}(1 + p\frac{3n-2}{4}). \tag{5.66.c}$$

5.4.9 Code of X_5

$$X_5 = yy^* \tilde{D}_A, \tag{5.67.a}$$

$$\frac{\partial X_5}{\partial y_j} = yy^* \frac{\partial \tilde{D}_A}{\partial y_j}, \quad J \le i, \tag{5.67.b}$$

$$\frac{\partial X_5}{\partial y_j} = (\frac{\partial y}{\partial y_j} y^* + y \frac{\partial y^*}{\partial y_j}) \tilde{D}_A, \quad J = i+1, \tag{5.67.c}$$

$$\frac{\partial^2 X_5}{\partial y_j \partial y_k} = yy^* \frac{\partial^2 \tilde{D}_A}{\partial y_j \partial y_k}, \quad K \le J \le i, \tag{5.67.d}$$

$$\frac{\partial^2 X_5}{\partial y_j \partial y_k} = (\frac{\partial y}{\partial y_j} y^* + y \frac{\partial y^*}{\partial y_j}) \frac{\partial \tilde{D}_A}{\partial y_k}, \quad K < J = i+1, \tag{5.67.e}$$

$$\frac{\partial^2 X_5}{\partial y_j \partial y_k} = \frac{\partial^2 yy^*}{\partial y_j \partial y_k} \tilde{D}_A, \quad K = J = i+1. \tag{5.67.f}$$

The pseudo code

$$\forall i: \quad T_7(i) = y_t(i) * D(i), \tag{5.68.a}$$

$$\forall i: \quad Z_0(i) = y(i) * T_7(i), \tag{5.68.b}$$

$$\forall i, j \ (J = i+1): \quad dZ_0(i+1, j) = T_7(i)(j), \tag{5.68.c}$$

$$\forall i, j \ (J = i+1): \quad dZ_0(i+1, j) = y(i) * D(i)(j), \tag{5.68.d}$$

$$\forall i, j \ (J \le i): \quad T_8(i, j) = y_t(i) * dD(i, j), \tag{5.68.e}$$

$$\forall i, j \ (J \le i): \quad dZ_0(i+1, j) = y(i) * T_8(i, j), \tag{5.68.f}$$

$$\forall i, j, k \ (K \le J \le i): \quad T(i) = y_t(i) * ddD(i, j, k), \tag{5.68.g}$$

$$\forall i, j, k \ (K \le J \le i): \quad ddZ_0(i+1, j, k) = y(i) * T(i), \tag{5.68.h}$$

$$\forall i, j, k \ (K < J = i+1): \quad ddZ_0(i+1, j, k) = y_y(i) * dD(i, k)(j), \tag{5.68.i}$$

$$\forall i, j, k \ (K < J = i+1): \quad ddZ_0(i+1, j, k) = T_8(i, k), \tag{5.68.j}$$

Table 5.30: Cost of X_5 in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	24	56	80	600	7 000	42 000
5	120	760	3 040	3 000	95 000	1 740 000
8	288	2 976	18 816	7 200	372 000	11 088 000
10	440	5 720	44 880	11 000	715 000	26 730 000
20	1 680	44 240	680 960	42 000	5 530 000	414 960 000

$$\forall i,j,k \ (K < J = i+1): \quad ddZ_0(i+1,j,k) = D(i)(j). \tag{5.68.k}$$

The cost of the multiplications in (a) and (b) is $p^2(i+1)$, hence

$$T(X_5) = p^2 n(n+1). \tag{5.69.a}$$

There is no cost for (c), the cost of (d) is $p^2(i+1)$, this is the same as (e), (f), hence a total of $\sum p^3(i+1) + 2p^2 ip(i+1)$.

$$T(X_5') = \frac{p^3 n(n+1)(4n-1)}{6}. \tag{5.69.b}$$

The cost of the products in (g), (h) and (i) are $p^2(i+1)$, hence a total of $\sum p^2(i+1)(ip(ip+1)+ip^2)$.

$$T(X_5'') = \frac{p^3 n(n-1)(n+1)(3pn+2p+4)}{12}. \tag{5.69.c}$$

5.4.10 Case of X_6

$$X_6 = yu^* q_A, \tag{5.70.a}$$

$$\frac{\partial X_6}{\partial y_j} = yu^* \frac{\partial q_A}{\partial y_j}, \quad J \le i, \tag{5.70.b}$$

$$\frac{\partial X_6}{\partial y_j} = \frac{\partial y}{\partial y_j} u^* q_A, \quad J = i+1, \tag{5.70.c}$$

$$\frac{\partial^2 X_6}{\partial y_j \partial y_k} = yu^* \frac{\partial^2 q_A}{\partial y_j \partial y_k}, \quad K \le J \le i, \tag{5.70.d}$$

$$\frac{\partial^2 X_6}{\partial y_j \partial y_k} = \frac{\partial y}{\partial y_j} u^* \frac{\partial q_A}{\partial y_k}, \quad K < J = i+1. \tag{5.70.e}$$

Pseudo code

$$\forall i: \quad T_{12}(i) = y(i) * u_t(i), \tag{5.71.a}$$

$$\forall i: \quad Z_0(i+1) = T_{12}(i) * q(i), \tag{5.71.b}$$

$$\forall i,j \ (J \le i): \quad dZ_0(i+1,j) = T_{12}(i) * dq(i,j), \tag{5.71.c}$$

$$\forall i,j \ (J = i+1): \quad dZ_0(i+1,j) = u_t(i)(j) * q(i), \tag{5.71.d}$$

$$\forall i,j,k \ (K \le J \le i): \quad ddZ_0(i+1,j,k) = T_{12}(i) * ddq(i,j,k), \tag{5.71.e}$$

$$\forall i,j,k \ (K < J = i+1): \quad ddZ_0(i+1,j,k) = u_t(i)(j) * dq(i,k). \tag{5.71.f}$$

The cost of (b), (c) and (e) is $p^2(i+1)$, the cost of (a) is p^2, the cost of (d) and (f) is $p(i+1)$, hence

$$T(X_6) = p^2 n(n+3)/2. \tag{5.72.a}$$

$$T(X_6') = \frac{p^3 n(n-1)(n+1)}{3} + p^2 \frac{n(n+1)}{2}. \tag{5.72.b}$$

$$T(X_6'') = \frac{p^3 n(n-1)(n+1)}{6}\left(3 + p\frac{3n-2}{4}\right). \tag{5.72.c}$$

Table 5.31: Cost of X_6 in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	20	28	40	500	2 300	13 000
5	80	380	1 520	2 000	41 500	710 000
8	176	1 488	9 408	4 400	171 600	4 872 000
10	260	2 860	22 440	6 500	335 500	12 045 000
20	920	22 120	340 480	23 000	2 681 000	196 840 000

5.4.11 Case of X_7

$$X_7 = \tilde{D}_A u.y^* \tilde{D}_A, \tag{5.73.a}$$

$$\frac{\partial X_7}{\partial y_j} = \tilde{D}_A u.\frac{\partial y^*}{\partial y_j} \tilde{D}_A, \quad J = i+1, \tag{5.73.b}$$

$$\frac{\partial X_7}{\partial y_j} = \frac{\partial \tilde{D}_A}{\partial y_j} u.y^* \tilde{D}_A u + \tilde{D}_A u.y^* \frac{\partial \tilde{D}_A}{\partial y_j}, \quad J \le i, \tag{5.73.c}$$

$$\frac{\partial^2 X_7}{\partial y_j \partial y_k} = \frac{\partial \tilde{D}_A}{\partial y_k} u.\frac{\partial y^*}{\partial y_j} \tilde{D}_A + \tilde{D}_A u.\frac{\partial y^*}{\partial y_j} \frac{\partial \tilde{D}_A}{\partial y_k}, \quad K < J = i+1, \tag{5.73.d}$$

$$\frac{\partial^2 X_7}{\partial y_j \partial y_k} = \frac{\partial^2 \tilde{D}_A}{\partial y_j \partial y_k} u.y^* \tilde{D}_A + \tilde{D}_A u.y^* \frac{\partial^2 \tilde{D}_A}{\partial y_j \partial y_k} + \frac{\partial \tilde{D}_A}{\partial y_j} u.y^* \frac{\partial \tilde{D}_A}{\partial y_k} + \frac{\partial \tilde{D}_A}{\partial y_k} u.y^* \frac{\partial \tilde{D}_A}{\partial y_j}, \quad K \le J \le i. \tag{5.73.e}$$

Pseudo code

$$\forall i : \quad X_9(i+1) = T_9(i) * T_7(i), \tag{5.74.a}$$

$$\forall i,j \ (J \le i) : \quad dX_9(i+1,j) = T_4(i,j) * T_7(i) + T_9(i) * T_8(i,j), \tag{5.74.b}$$

$$\forall i,j \ (J = i+1) : \quad dX_9(i+1,j) = T_9(i) * D(i)(j), \tag{5.74.c}$$

$$\forall i,j,k \ (K < J = i+1) : \quad ddX_9(i+1,j,k) = T_4(i,k) * D(i)(j) + T_9(i) * dD(i,k)(j), \tag{5.74.d}$$

$$\forall i,j,k \ (K \le J \le i) : \quad ddX_9(i+1,j,k) = T_4(i,j) * T_8(i,k), \tag{5.74.e}$$

$$\forall i,j,k \ (K \le J \le i) : \quad T = ddD(i,j,k) * u(i), \tag{5.74.f}$$

$$\forall i,j,k \ (K \le J \le i) : \quad ddX_9(i+1,j,k) = T * T_7(i), \tag{5.74.g}$$

$$\forall i,j,k \ (K \le J \le i) : \quad T = y_t(i) * ddD(i,j,k), \tag{5.74.h}$$

$$\forall i,j,k \ (K \le J \le i) : \quad ddX_9(i+1,j,k) = T_9(i) * T. \tag{5.74.i}$$

Complexity. For (a) we have $\sum p^2(i+1)^2$ hence

$$T(X_7) = p^2 \frac{n(n+1)(2n+1)}{6}. \tag{5.75.a}$$

Each multiplication in (b) and (c) costs $p^2(i+1)^2$. There are $2ip+p$ such products, thus

$$T(X_7') = \frac{p^3 n(n+1)(3n^2+n-1)}{6}. \tag{5.75.b}$$

Each product in (d), (e), (f), (g), (h) and (i) costs $p^2(i+1)$. For (d) we have $2ip^2$ products, for (e) we have $(ip)^2$ products, and for the other we have $ip(ip+1)/2$ each. Hence

$$T(X_7'') = p^3 n(n-1)(n+1) \frac{18pn^2 + 15pn + 15n - 2p + 10}{30}. \tag{5.75.c}$$

Table 5.32: Cost of X_7 in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	20	104	384	500	13 000	208 000
5	220	3 160	36 192	5 500	395 000	21 260 000
8	816	19 104	358 848	20 400	2 388 000	215 544 000
10	1 540	45 320	1 070 784	38 500	5 665 000	648 120 000
20	11 480	682 640	32 571 168	287 000	85 330 000	20 027 140 000

5.4.12 Case of X_8

$$X_8 = y^* \tilde{D}_A u.\tilde{D}_A, \tag{5.76.a}$$

$$\frac{\partial X_8}{\partial y_j} = \frac{\partial y^*}{\partial y_j} \tilde{D}_A u.\tilde{D}_A, \quad J = i+1, \tag{5.76.b}$$

$$\frac{\partial X_8}{\partial y_j} = y^* \frac{\partial \tilde{D}_A}{\partial y_j} u.\tilde{D}_A + y^* \tilde{D}_A u.\frac{\partial \tilde{D}_A}{\partial y_j}, \quad J \leq i, \tag{5.76.c}$$

$$\frac{\partial^2 X_8}{\partial y_j \partial y_k} = \frac{\partial y^*}{\partial y_j} \frac{\partial \tilde{D}_A}{\partial y_k} u.\tilde{D}_A + \frac{\partial y^*}{\partial y_j} \tilde{D}_A u.\frac{\partial \tilde{D}_A}{\partial y_k} + y^* \frac{\partial \tilde{D}_A}{\partial y_j} u.\frac{\partial \tilde{D}_A}{\partial y_k}, \quad K < J = i+1, \tag{5.76.d}$$

$$\frac{\partial^2 X_8}{\partial y_j \partial y_k} = y^* \frac{\partial^2 \tilde{D}_A}{\partial y_j \partial y_k} u.\tilde{D}_A + y^* \frac{\partial \tilde{D}_A}{\partial y_k} u.\frac{\partial \tilde{D}_A}{\partial y_j} + y^* \tilde{D}_A u.\frac{\partial^2 \tilde{D}_A}{\partial y_j \partial y_k}, \quad K \leq J \leq i. \tag{5.76.e}$$

Pseudo code

$$\forall i : \quad T_{10}(i) = y_t(i) * T_9(i), \tag{5.77.a}$$

$$\forall i : \quad X_9(i+1) = T_{10}(i) * D(i), \tag{5.77.b}$$

$$\forall i,j \ (J = i+1) : \quad dX_9(i+1,j) = T_9(i)(j) * D(i), \tag{5.77.c}$$

$$\forall i,j \ (J \leq i) : \quad T_{11}(i,j,J) = y_t(i) * T_4(i,j), \tag{5.77.d}$$

$$\forall i,j \ (J \leq i) : \quad dX_9(i+1,j) = T_{11}(i,j) * D(i), \tag{5.77.e}$$

$$\forall i,j \ (J \leq i) : \quad dX_9(i+1,j) = T_{10}(i) * dD(i,j), \tag{5.77.f}$$

$$\forall i,j,k \ (K < J = i+1) : \quad ddX_9(i+1,j,k) = T_4(i,k)(j) * D(i), \tag{5.77.g}$$

$$\forall i,j,k \ (K < J = i+1) : \quad ddX_9(i+1,j,k) = T_9(i)(j) * dD(i,k), \tag{5.77.h}$$

$$\forall i,j,k \ (K \leq J \leq i) : \quad T = y_t(i) * T_5(i,j,k), \tag{5.77.i}$$

$$\forall i,j,k \ (K \leq J \leq i) : \quad ddX_9(i+1,j,k) = T * D(i), \tag{5.77.j}$$

$$\forall i,j,k \ (K \leq J \leq i) : \quad ddX_9(i+1,j,k) = T_{11}(i,k) * dD(i,j), \tag{5.77.k}$$

$$\forall i,j,k \ (K \leq J \leq i) : \quad ddX_9(i+1,j,k) = T_{11}(i,j) * dD(i,k), \tag{5.77.l}$$

$$\forall i,j,k \ (K \leq J \leq i) : \quad ddX_9(i+1,j,k) = T_{10}(i) * ddD(i,j,k). \tag{5.77.m}$$

Complexity The cost of (a) is $p(i+1)$, the cost of (b) is $p^2(i+1)^2$, hence

$$T(X_8) = p\frac{n(n+1)}{2} + p^2\frac{n(n+1)(2n+1)}{6}. \tag{5.78.a}$$

The cost of (d) is $p(i+1)$, the cost of (c), (e), (f) is $p^2(i+1)^2$. Number of operations: p for (c), ip for (d), (e) and (f), hence

$$T(X_8') = p^3\frac{n(n+1)(2n+1)}{6} + p^2\frac{n(n-1)(n+1)}{3} + p^3\frac{n(n-1)(n+1)(3n+2)}{6}. \tag{5.78.b}$$

Table 5.33: Cost of X_8 in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	26	112	300	530	13 200	165 100
5	250	3 320	26 088	5 650	399 000	15 317 000
8	888	1 9776	252 000	20 760	2 404 800	151 082 400
10	1 650	46 640	744 876	39 050	5 698 000	449 971 500
20	11 900	693 280	22 203 552	289 100	85 596 000	13 631 968 000

Table 5.34: Cost of X in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	32	64	144	800	8 000	66 000
5	280	2 720	19 008	7 000	340 000	10 680 000
8	960	17 472	205 632	24 000	2 184 000	120 456 000
10	1 760	42 240	632 016	44 000	5 280 000	375 210 000
20	12 320	659 680	20 416 032	308 000	82 460 000	12 440 820 000

The cost of (i) is $p(i+1)$, the cost of (g), (h), (j), (k), (l) and (m) is $p^2(i+1)^2$. Equations (g) and (h) are executed ip^2 times, (i), (j) and (m) are executed $ip(ip+1)/2$ times, while (k,l) is executed $(ip)^2$ times (note the special case when indices k and j are equal).

Hence $\sum 2p^4 i(i+1)^2 + p^2 i(i+1)(ip+1)/2 + p^2(i+1)^2 ip(ip+1) + p^4 i^2(i+1)^2$.

$$T(X_8'') = n(n-1)(n+1)p^2 \frac{48p^2n^2 + 60p^2n + 45pn + 20 + 10p + 8p^2}{120}. \tag{5.78.c}$$

5.4.13 Cost of X

If $X_9 = X_7 - X_8$, then

$$X = X_9/q_A, \tag{5.79.a}$$

$$\frac{\partial X}{\partial y_j} = \frac{1}{q_A}\Big(\frac{\partial X_9}{\partial y_j} - X * \frac{\partial q_A}{\partial y_j}\Big), \tag{5.79.b}$$

$$\frac{\partial^2 X}{\partial y_j \partial y_k} = \frac{1}{q_A}\Big(-\frac{\partial q_A}{\partial y_k}\frac{\partial X}{\partial y_j} - \frac{\partial X}{\partial y_k}\frac{\partial q_A}{\partial y_j} - X * \frac{\partial^2 q_A}{\partial y_j \partial y_k} + \frac{\partial^2 X_9}{\partial y_j \partial y_k}\Big). \tag{5.79.c}$$

The pseudo code is trivial. Note however the following facts. When we compute X and its first derivative, we compute first the numerator, store it somewhere, divide, and store the result. This will be added to Z_0 and its derivative. However, there is no need to store the second derivative of X, we just add it.

Complexity: each polynomial division costs $(i+1)(i+2)$.

$$T(X) = p^2 \frac{n(n+1)(n+2)}{3}. \tag{5.80.a}$$

$$T(X') = p^3 n(n-1)(n+1)(3n+2)/6. \tag{5.80.b}$$

$$T(X'') = p^3 n(n+1)(n-1)\frac{8pn^2 - 5pn + 10n - 2p}{20}. \tag{5.80.c}$$

Table 5.35: Cost of Z in direct mode

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	24	32	48	600	4 000	22 000
5	120	640	2 400	3 000	80 000	1 340 000
8	288	2 688	16 128	7 200	336 000	9 408 000
10	440	5 280	39 600	11 000	660 000	23 430 000
20	1 680	42 560	638 400	42 000	5 320 000	388 360 000

5.4.14 Case of Z

Since

$$Z = (b - \tilde{b})Z_0, \tag{5.81.a}$$

we have

$$\frac{\partial Z}{\partial y_j} = (b - \tilde{b})\frac{\partial Z_0}{\partial y_j}, \tag{5.81.b}$$

$$\frac{\partial^2 Z}{\partial y_j \partial y_k} = (b - \tilde{b})\frac{\partial^2 Z_0}{\partial y_j \partial y_k}. \tag{5.81.c}$$

Each operation costs $2(i+1)p^2$.

$$T(Z) = p^2 n(n+1). \tag{5.82.a}$$

$$T(Z') = \frac{2}{3}p^3 n(n-1)(n+1). \tag{5.82.b}$$

$$T(Z'') = \frac{p^3 n(n-1)(n+1)(3pn - 2p + 4)}{12}. \tag{5.82.c}$$

This gives a total complexity which is the following.

$$T(Q) = n\left(2 + 2\,n + pn + 5\,p + 15\,p^2 n + 2\,p^2 n^2 + 15\,p^2\right)/2. \tag{5.83.a}$$

$$T(Q') = n + n^2 + p[\frac{3n^2}{2} + + \frac{11n}{6} + \frac{2n^3}{3}] + p^2[\frac{3n^2}{2} + \frac{n^3}{3} + \frac{7n}{6}] + p^3[-\frac{7n}{3} + \frac{16n^3}{3} + \frac{n^2}{2} + \frac{3n^4}{2}]. \tag{5.83.b}$$

$$T(Q'') = 2pn(n-1) + \frac{pn(n-1)(n+1)}{120}(168p^3n^2 + 30pn + 255p^3n + 165p^2n + 40 + 160p - 2p^3 + 310p^2). \tag{5.83.c}$$

On figure 5.9 we have plotted some values. On figure 5.10, we have plotted the ratio of (b) and (c) by (a) (we divided by np in the first case, and $np(np+1)/2$ in the second case). These ratios should be smaller than 2 and 4 respectively. In all equations but X_7, X_8 and X, the ratio for the second derivative is 1/2. This means that the average complexity of the second derivative of a product ab is 1/2; it is due to the fact that, in general, one of the factors is constant, and the other factor depends only on half of the input variables. In the case of X_7 and X_8, it happens that $\tilde{D}_A$ appears twice in the formula. In the case of X, we divide something non-constant by q, so that the ratio is greater. According to (5.83.c), the limit for large p and n is 14/5.

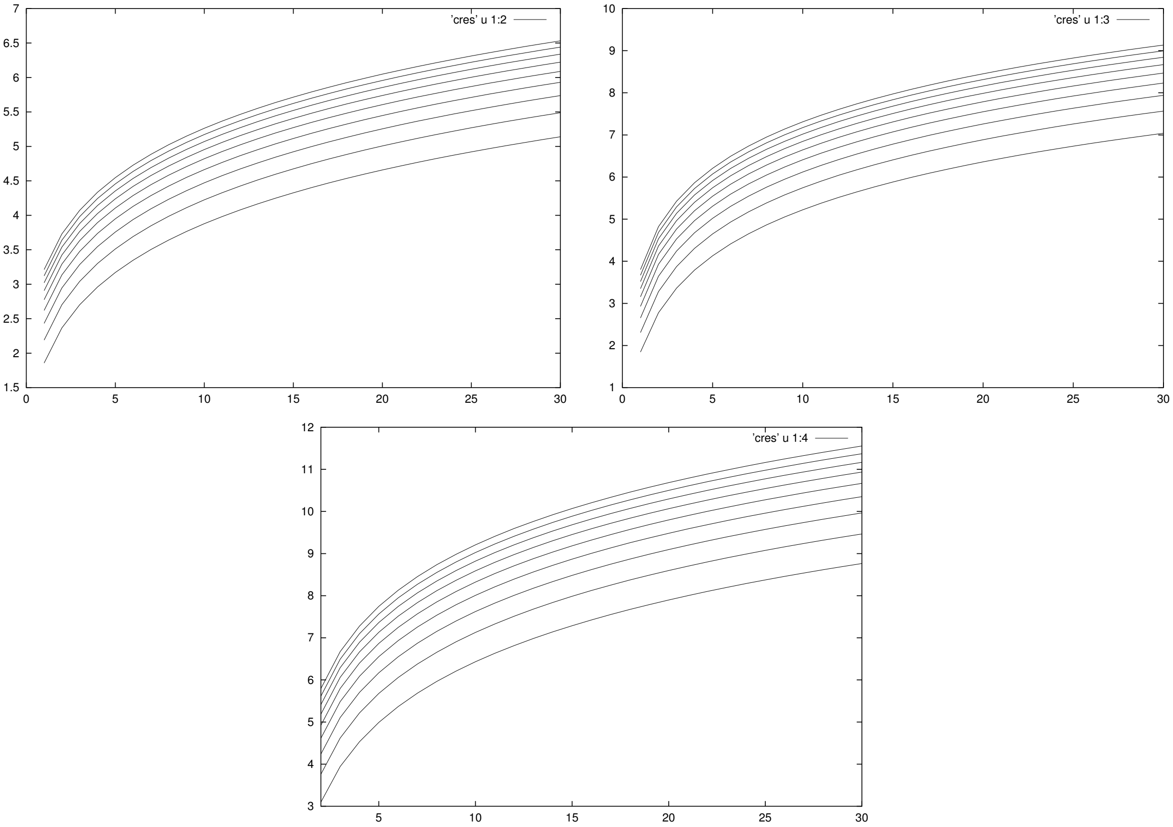

Figure 5.9: Cost of the Schur parameters direct mode. On each curve p is fixed (between 2 and 10). We give the complexity of Q, Q' and Q''. For Q'', we assume $n \geq 2$, since no multiplication is required for $n = 1$.

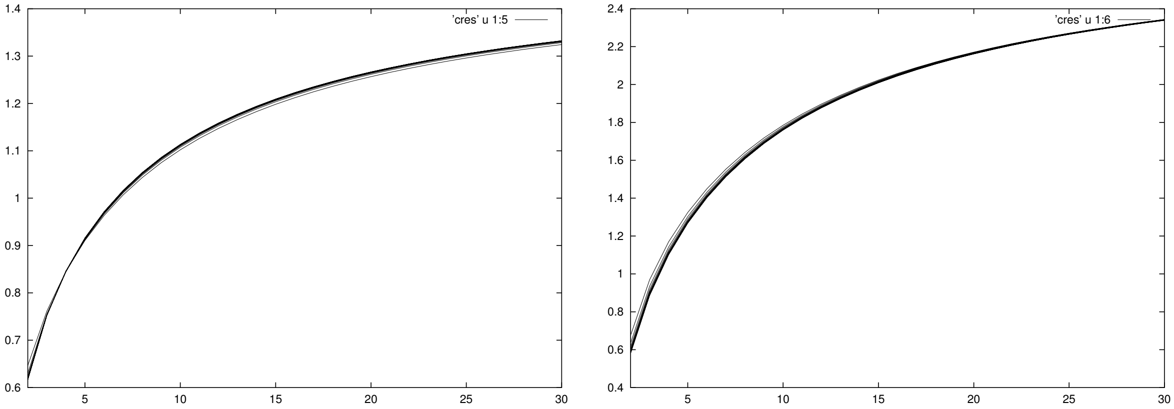

Figure 5.10: Cost of the Schur parameters direct mode. On each curve p is fixed (between 2 and 10). We give the ratio of the cost of one element of Q' or Q'' by the cost of Q.

5.4.15 Complexity of ψ

We assume here that $\tilde{D}$, q and the derivatives of these quantities have been computed and stored somewhere. We compute now ψ and its derivatives. In order to speed up things, we assume q monic. This means that we have to divide everything by the leading coefficient u of q. Assume that we divide a by u, giving b. The equations are

$$b = \frac{a}{u}. \tag{5.84.a}$$

$$\frac{\partial b}{\partial y} = \frac{1}{u}(\frac{\partial a}{\partial y} - b\frac{\partial u}{\partial y}). \tag{5.84.b}$$

$$\frac{\partial^2 b}{\partial y \partial t} = \frac{1}{u}(\frac{\partial^2 a}{\partial y \partial t} - \frac{\partial b}{\partial t}\frac{\partial u}{\partial y} - \frac{\partial b}{\partial y}\frac{\partial u}{\partial t} - b\frac{\partial^2 u}{\partial y \partial t}). \tag{5.84.c}$$

No additional memory is required. The cost of (a), (b) and (c) is 1, 2 and 4. We have $(p^2+1)(n+1)-1$ such quantities to compute, since there are $p^2(n+1)$ coefficients in $\tilde{D}$, and n coefficients in q (the leading coefficient of q is left unchanged). This gives a complexity of

$$T(\psi) = (p^2+1)(n+1)-1. \tag{5.85.a}$$

$$T(\psi') = 2np[(p^2+1)(n+1)-1]. \tag{5.85.b}$$

$$T(\psi) = 2np(np+1)[(p^2+1)(n+1)-1]. \tag{5.85.c}$$

We implement now the formula $G\tilde{D} = Vq + R$, $\psi = \|G\|^2 - \|V\|^2$. Assume that $G = \sum G_k z^k$. Let $R_m = 0$ and $\psi_m = \|G\|^2$. Consider

$$R_{k+1}z + G_k\tilde{D} = V_k q + R_k, \tag{5.86.a}$$

$$\psi_k = \psi_{k+1} - \|V\|_k^2. \tag{5.86.b}$$

In each iteration of (a), we compute the left hand side. The leading coefficient of this will be V_k. We subtract $V_k(q - z^n)$ from this. Given V_k, we compute its norm and update ψ. After that, computing $R_{k+1}z$ is nothing else than shifting R_{k+1} and ignoring the leading term V_k. Thus we need only allocate space for a matrix of polynomials of degree n.

Assume that G has g rows. This gives a space complexity of

$$S(\psi) = gp(n+1). \tag{5.87.a}$$

$$S(\psi') = npgp(n+1). \tag{5.87.b}$$

$$S(\psi'') = np(np+1)gp(n+1)/2. \tag{5.87.c}$$

If we differentiate (5.86) we get

$$\frac{\partial R_{k+1}}{\partial y}z + G_k\frac{\partial \tilde{D}}{\partial y} = \frac{\partial V_k}{\partial y}q + V_k\frac{\partial q}{\partial y} + \frac{\partial R_k}{\partial y}. \tag{5.88.a}$$

$$\frac{\partial \psi_k}{\partial y} = \frac{\partial \psi_{k+1}}{\partial y} - 2\langle V_k \mid \frac{\partial V_k}{\partial y}\rangle. \tag{5.88.b}$$

and

$$\frac{\partial R_{k+1}}{\partial y \partial t}z + G_k\frac{\partial^2 \tilde{D}}{\partial y \partial t} = \frac{\partial^2 V_k}{\partial y \partial t}q + V_k\frac{\partial^2 q}{\partial y \partial t} + \frac{\partial V_k}{\partial y}\frac{\partial q}{\partial t} + \frac{\partial V_k}{\partial t}\frac{\partial q}{\partial y} + \frac{\partial^2 R_k}{\partial y \partial t}. \tag{5.89.a}$$

$$\frac{\partial^2 \psi_k}{\partial y \partial t} = \frac{\partial^2 \psi_{k+1}}{\partial y \partial t} - 2\langle \frac{\partial V_k}{\partial y} \mid \frac{\partial V_k}{\partial t}\rangle - 2\langle V_k \mid \frac{\partial^2 V_k}{\partial y \partial t}\rangle. \tag{5.89.b}$$

Multiplying G_k by a matrix costs $gp^2(n+1)$, multiplying a matrix by a polynomial costs $gp(n+1)$, the division by q costs gpn. Hence the complexity is

$$T(\psi) = mgp(n+1)(p+1). \tag{5.90.a}$$

$$T(\psi') = npmgp(n+1)(p+2). \tag{5.90.b}$$

$$T(\psi'') = \frac{np(np+1)}{2} mgp[(n+1)(p+4)+1]. \tag{5.90.c}$$

We have another way to compute ψ, namely $G = V_1q+R_1$, $R_1\tilde{D} = V_2q+R_2$, $\psi = \|G\|^2 - \|R_1\|^2 + \|V_1\|^2$. We use the same technique as before. Once R_1 has been computed, we copy it in a matrix W, and define $W = \sum w_k z^k$. We use the same memory location for the two division loops, but in the formulas that follow, we shall add primes for the second loop.

$$S(\psi) = gp(n+2). \tag{5.90.a}$$

$$S(\psi') = npgp(n+2). \tag{5.90.b}$$

$$S(\psi'') = np(np+1)gp(n+2)/2. \tag{5.90.c}$$

The formulas are

$$R_{k+1}z + G_k = V_kq + R_k. \tag{5.91.a}$$

$$R'_{k+1}z + W_k\tilde{D} = V'_kq + R'_k. \tag{5.91.b}$$

$$\psi_k = \psi_{k+1} + \|V'_k\|^2 - \|W_k\|^2. \tag{5.91.c}$$

The formulas for the derivatives are

$$\frac{\partial R_{k+1}}{\partial y} z = \frac{\partial V_k}{\partial y} q + V_k \frac{\partial q}{\partial y} + \frac{\partial R_k}{\partial y}. \tag{5.93.a}$$

$$\frac{\partial R'_{k+1}}{\partial y} z + \frac{\partial W_k}{\partial y} \tilde{D} + W_k \frac{\partial \tilde{D}}{\partial y} = \frac{\partial V'_k}{\partial y} q + V'_k \frac{\partial q}{\partial y} + \frac{\partial R'_k}{\partial y}. \tag{5.93.b}$$

$$\frac{\partial \psi_k}{\partial y} = \frac{\partial \psi_{k+1}}{\partial y} + 2\langle V'_k \mid \frac{\partial V'_k}{\partial y}\rangle - 2\langle W_k \mid \frac{\partial W_k}{\partial y}\rangle. \tag{5.93.c}$$

The formulas for the second derivatives are

$$\frac{\partial^2 R_{k+1}}{\partial y \partial t} z = \frac{\partial^2 V_k}{\partial y \partial t} q + \frac{\partial V_k}{\partial y}\frac{\partial q}{\partial t} + \frac{\partial V_k}{\partial t}\frac{\partial q}{\partial y} + V_k \frac{\partial^2 q}{\partial y \partial t} + \frac{\partial^2 R_k}{\partial y \partial t}. \tag{5.94.a}$$

$$\begin{aligned} &\frac{\partial^2 R'_{k+1}}{\partial y \partial t} z + \frac{\partial^2 W_k}{\partial y \partial t}\tilde{D} + \frac{\partial W_k}{\partial y}\frac{\partial \tilde{D}}{\partial t} + \frac{\partial W_k}{\partial t}\frac{\partial \tilde{D}}{\partial y} + W_k \frac{\partial^2 \tilde{D}}{\partial y \partial t} = \\ &= \frac{\partial^2 V'_k}{\partial y \partial t} q + \frac{\partial W_k}{\partial y}\frac{\partial \tilde{D}}{\partial t} + \frac{\partial W_k}{\partial t}\frac{\partial \tilde{D}}{\partial y} + \frac{\partial V'_k}{\partial y}\frac{\partial q}{\partial t} + V'_k \frac{\partial^2 q}{\partial y \partial t} + \frac{\partial^2 R'_k}{\partial y \partial t}. \end{aligned} \tag{5.94.b}$$

$$\frac{\partial^2 \psi_k}{\partial y \partial t} = \frac{\partial^2 \psi_{k+1}}{\partial y \partial t} + 2(\langle V'_k \mid \frac{\partial^2 V'_k}{\partial y \partial t}\rangle + \langle \frac{\partial V'_k}{\partial y} \mid \frac{\partial V'_k}{\partial t}\rangle - \langle W_k \mid \frac{\partial^2 W_k}{\partial y \partial t}\rangle - \langle \frac{\partial W_k}{\partial y} \mid \frac{\partial W_k}{\partial t}\rangle). \tag{5.94.c}$$

The complexity is easy to compute. It is

$$T(\psi) = gpn[m+2+p(n+1)]. \tag{5.95.a}$$

$$T(\psi') = npgp[m(2n+1) + 2n(1+p(n+1))]. \tag{5.95.b}$$

$$T(\psi'') = gpn\frac{np(np+1)}{2}[m(4n+3) + 4n(1+p(n+1))]. \tag{5.95.c}$$

Table 5.36: Cost of ψ, formulas (5.95), in direct mode, $m = 400$.

n	ψ	ψ'	ψ''	ψ	ψ'	ψ''
2	3 264	32 448	356 480	86 400	4 248 000	195 216 000
5	8 280	181 200	10 406 000	231 000	25 050 000	6 642 750 000
8	13 440	454 656	63 574 016	393 600	66 048 000	43 835 904 000
10	16 960	708 800	152 208 000	512 000	106 200 000	109 282 000 000
20	35 520	2 899 200	2 403 584 000	1 224 000	496 800 000	2 013 216 000 000

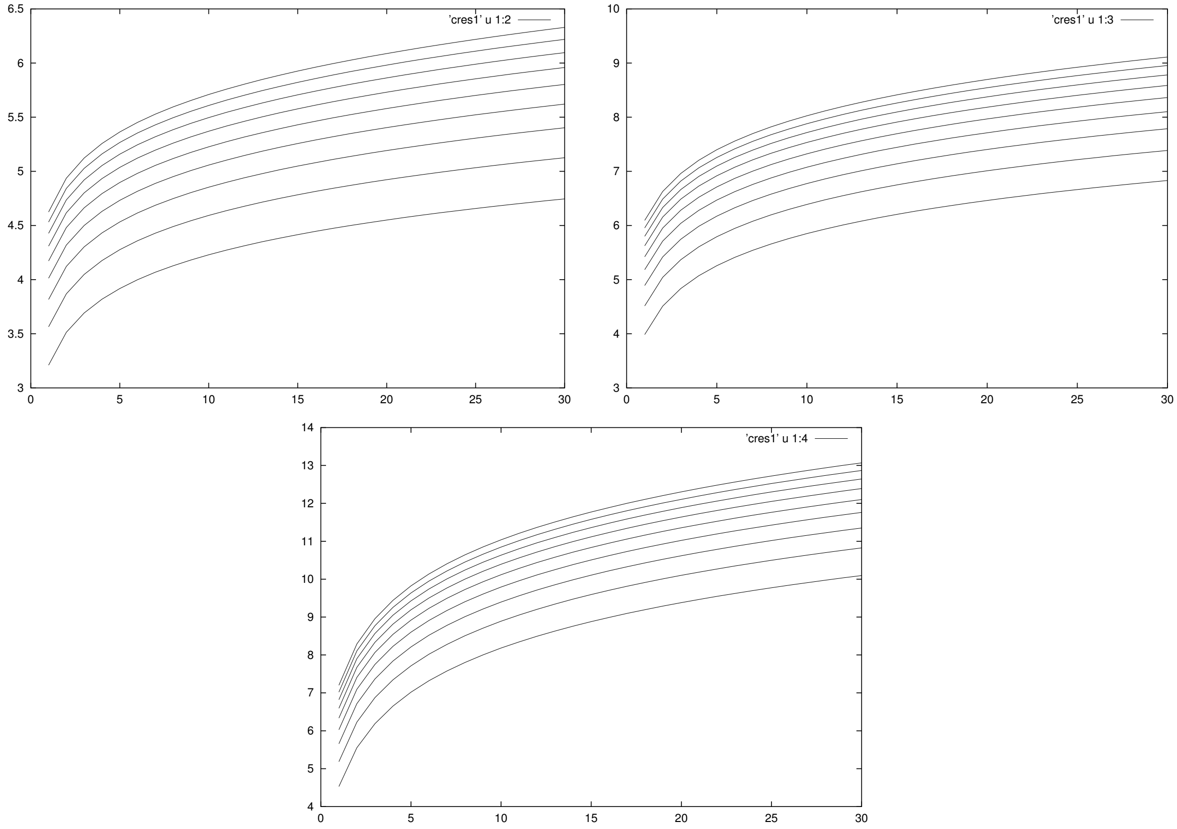

Figure 5.11: Cost of ψ in direct mode. On each curve p is fixed (between 2 and 10). We assume $g = p$ and $m = 400$. The curves correspond to the function, the first derivative and the second derivative

Let's compute the ratio of these two complexities. We introduce

$$\lambda = 1 + p(n+1), \quad t = m/n.$$

$$r = 1 + \frac{(t-1)\lambda - 1}{tn + \lambda + 1}. \tag{5.96.a}$$

$$r' = 1 + \frac{(t-2)\lambda}{2tn + t + 2\lambda}. \tag{5.96.b}$$

$$r'' = 1 + \frac{(t-4)\lambda + t}{t(4n+3) + 4\lambda}. \tag{5.96.c}$$

In case $t \geq 4$, these quantities are greater than one, and the second method is better than the first one. In the case m is much larger than n, we can simplify a bit these formulas

$$r = 1 + \frac{\lambda}{n}. \tag{5.97.a}$$

$$r' = 1 + \frac{\lambda}{2n+1}. \tag{5.97.b}$$

$$r'' = 1 + \frac{\lambda + 1}{4n+3}. \tag{5.97.c}$$

If we assume that np is much smaller than m, we get

$$r = 1 + p, \quad r' = 1 + p/2, \quad r'' = 1 + p/4. \tag{5.98.a}$$

But if p becomes large, we get

$$r = t, \quad r' = t/2, \quad r'' = t/4. \tag{5.98.b}$$

For instance, if $m = 200$ and $n = 20$, then $t = 10$. Hence r'' is $1 + p/4$ for small p, $10/4$ for large p. In case $p = 2$, we have

$$r = 2.6, \quad r' = 1.7 \quad r'' = 1.4.$$

This means that we have more than a factor two for ψ, and 40% for the second derivative.

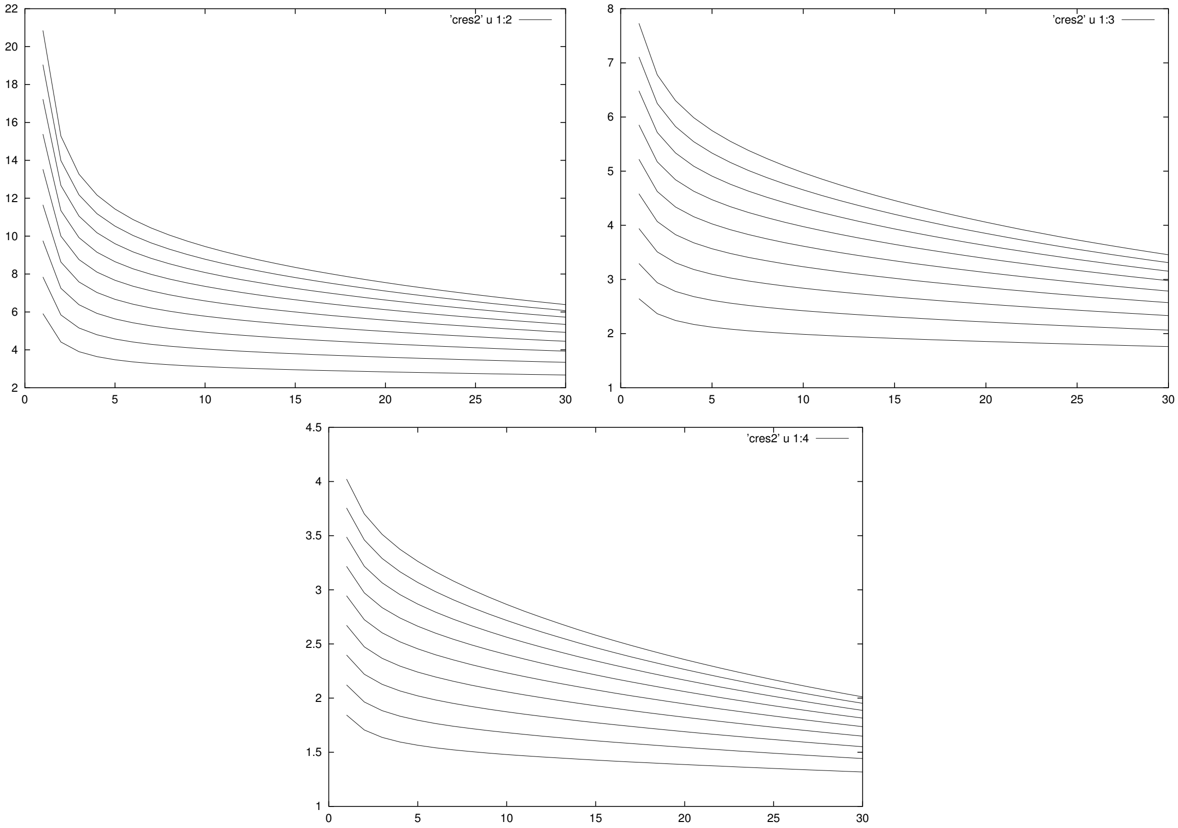

Figure 5.12: Gain. On each curve p is fixed (between 2 and 10). We assume $g = p$ and $m = 400$. Function, first derivative, second derivative.

Bibliography

[1] Daniel Alpay, Laurent Baratchart, and Andrea Gombani. On the differential structure of matrix-valued rational inner functions. *Operator Theory: Advances and Applications*, 73:30–66, 1994.

[2] Laurent Baratchart. *Sur l'approximation rationnelle L^2 pour les systèmes dynamiques linéaires.* PhD thesis, Université de Nice, 1987.

[3] Laurent Baratchart and José Grimm. An elementary proof of the nonexistence of canonical forms in the real and complex case. *Systems and Control Letters*, 3:193–196, September 1993.

[4] Michel Cardelli. *Contribution à l'approximation rationnelle L^2 des fonctions de transfert.* PhD thesis, Université de Nice-Sophia Antipolis, 1990.

[5] Jean-Pierre Dedieu and Jean-Claude Yakoubsohn. Localization of an algebraic hypersurface by the exclusion algorithm. *AAEC*, 2:239–256, 1992.

[6] C. Faure and Y. Papegay. Odyssée User's Guide. Version 1.7. Rapport technique 0224, INRIA, September 1998.

[7] Christèle Faure. Le gradient de THYC3D par Odyssée. Technical Report RR-3519, Inria, 1998.

[8] Pascale Fulcheri. *Approximation rationnelle matricielle dans H^2 et analyse de Schur. Application à l'identification des systèmes.* PhD thesis, Université de Nice-Sophia Antipolis, December 1994.

[9] Pascale Fulcheri and Martine Olivi. Matrix rational H_2 approximation: a gradient algorithm based on Schur analysis. *Siam J. on Cont. Optim.*, 36(6):2103–2127, November 1998.

[10] C. William Gear. *Numerical Initial Value Problems in Ordinary Diffe rential Equations.* Prentice Hall, 1971.

[11] G. Hall and J.M. Watt, editors. *Modern Numerical Methods for Ordinary Differential Equations.* Clarendon Press, Oxford, 1976.

[12] Donald E. Knuth and Silvio Levy. *The `CWEB` system of Structured Documentation.* Addison-Wesley, 1994.

[13] Juliette Leblond and Martine Olivi. Weighted H^2 approximation of transfer functions. Technical Report 2863, INRIA, 1996.

[14] Jacques Morgenstern. How to compute fast a function and its derivatives. *SIGACT news*, 1985.

[15] Norman Ramsey. Building a language-independent `WEB`. *Communications of the ACM*, pages 1051–1055, 1989.

[16] N. Rostaing. *Différentiation automatique: application à un problème d'optimisation en météorologie.* PhD thesis, Université de Nice-Sophia Antipolis, 1993.

[17] Walter Rudin. *Real and Complex Analysis.* McGraw-Hill, 1966.

[18] Szegö. *Orthogonal Polynomials.* Colloquium Publications, AMS, 1939.

[19] J.C. Willems. From time series to linear systems. part I: Finite dimensional linear time invariant systems. *Automatica*, 22:561–580, 1986.

List of Tables

List of Figures

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399