

E. Kaltofen
S.M. Watt
Editors

Computers and Mathematics

AD-A221 997

DTIC

ELECTE

MAY 02 1990

DTIC FILE COPY

$$\begin{aligned}
 & + \dots + \dots + \dots \\
 & \phi(v) = A(v) - 4v^{\frac{1}{2}} \cos^2 \frac{2\pi}{3} \dots (v) \\
 & + 2v^{\frac{3}{2}} \cos \frac{4\pi}{3} C(v) - 2v^{\frac{5}{2}} \\
 & \phi(v) = \{A(v) - 4 \sin^2 \frac{2\pi}{3} \phi(v)\} + v^{\frac{1}{2}} \\
 & + 2v^{\frac{3}{2}} \cos \frac{2\pi}{3} C(v) - 2v^{\frac{5}{2}} \cos \frac{2\pi}{3} \\
 & A(v) = \frac{1-v^2-v^3+v^4+\dots}{(1-v)^2(1-v^4)^2(1-v^6)^2 \dots} \\
 & B(v) = \frac{(1-v^5)(1-v^{10})(1-v^{15}) \dots}{(1-v)(1-v^4)(1-v^6) \dots} \\
 & C(v) = \frac{(1-v^5)(1-v^{10})(1-v^{15}) \dots}{(1-v^2)(1-v^3)(1-v^4) \dots} \\
 & D(v) = \frac{1-v-v^4+v^7+\dots}{(1-v)^2(1-v^3)^2(1-v^6) \dots} \\
 & \phi(v) = -1 + \left\{ \frac{1}{1-v} + \frac{(1-v)C(v)}{v^{20}} \right. \\
 & \left. + \frac{v^{20}}{(1-v)(1-v^3)(1-v^6)} \right\}
 \end{aligned}$$

$$\psi(v) = -1 + \left\{ \frac{1}{1-v^2} + \frac{(1-v^2)C(v)}{v^{20}} \right.$$

$$\left. + \frac{(1-v^2)(1-v^3)}{v^{20}} \right\}$$

$$\frac{v}{1-v} + \frac{v^3}{(1-v^2)(1-v^3)} +$$

$$= 3\phi(v) + 1 - A(v),$$

$$\frac{v}{1-v} + \frac{v^3}{(1-v^2)(1-v^3)} + \frac{v}{(1-v^2)(1-v^4)(1-v^6)}$$

$$= 3\psi(v),$$

$$\frac{v^3}{1-v} +$$

$$= \phi(v) -$$

$$\frac{v}{1-v}$$

$$= \psi(v) + v \cdot \frac{1+v^5+v^{\dots}}{(1-v^2)(1-v^3)(1-v^6) \dots}$$

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

Springer-Verlag

90 04 30 060

Computers and Mathematics

Erich Kaltofen Stephen M. Watt
Editors

Computers and Mathematics



Springer-Verlag
New York Berlin Heidelberg
London Paris Tokyo

Erich Kaltofen
Rensselaer Polytechnic
Department of Computer Science
Troy, NY 12180, U.S.A.

Stephen M. Watt
IBM Watson Research Center
Yorktown Heights, NY 10598, U.S.A.

Library of Congress Cataloging-in-Publication Data

Computers and mathematics / Erich Kaltofen, Stephen M. Watt, editors.
p. cm.

To be used at conference on computers & mathematics at
Massachusetts Institute of Technology, June 12, 1989.

I. Mathematics—Data processing—Congresses. I. Kaltofen, Erich.
II. Watt, Stephen M. III. Massachusetts Institute of Technology.

QA76.95.C64 1989
510'.28'5—dc20

89-6259

Printed on acid-free paper.

© 1989 by Springer-Verlag New York Inc.

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer-Verlag, 175 Fifth Avenue, New York, NY 10010, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use of general descriptive names, trade names, trademarks, etc. in this publication, even if the former are not especially identified, is not to be taken as a sign that such names, as understood by the Trade Marks and Merchandise Marks Act, may accordingly be used freely by anyone.

Camera-ready text provided by authors
Printed and bound by R.R. Donnelley & Sons, Harrisonburg, Virginia
Printed in the United States of America.

9 8 7 6 5 4 3 2 1

ISBN 0-387-97019-3 Springer-Verlag New York Berlin Heidelberg
ISBN 3-540-97019-3 Springer-Verlag Berlin Heidelberg New York

PRICE-\$39.00 per Springer Verlag
 TELECON 4/30/90
 175 Fifth Ave.
 N.Y. N.Y. 10010

VG

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By 39.00	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	21



About the Cover

Page from Ramanujan's Lost Notebook. Ramanujan's Lost Notebook is a collection of pages of formulas and calculations done by Ramanujan during the last year of his life. It was apparently in the possession of G. H. Hardy and G. N. Watson between 1920 and 1965; however neither one ever mentioned it in print. R. Rankin and J. M. Whittaker assisted Watson's widow in placing the Lost Notebook in the Wren Library in Cambridge. It was examined by G. E. Andrews in 1976, and he published the first discussion of its contents in the *American Mathematical Monthly* in 1979.

This is one of the most amazing pages in the Lost Notebook. The last four formulas are examples of the Mock Theta Conjectures (settled in 1988 by D. R. Hickerson). The formulas for $F(q^{1/5})$ and $f(q^{1/5})$ are, in fact, crucial to the explanation of certain partition congruences found by Dyson, Atkin, Swinnerton-Dyer and Garvan. (This page was reproduced by courtesy of Narosa Publishing House, New Delhi.)

Trefoil Tube. Building a tube around a space curve provides a powerful technique for analyzing local properties like curvature and torsion as well as global properties such as knottedness. The tube around a trefoil knot, produced by Thomas Banchoff and associates at Brown University, is related to the stereographic projection of an orbit of a dynamical system on the unit sphere in 4-dimensional space. This example was studied in collaboration with Hüseyin Koçak, Fred Bishopp, David Laidlaw and David Margolis.

Cover design by Alexandra Gatje/Richard Jenks

(P)

Preface

Computers and Mathematics '89 is the third in a series of conferences devoted to the use of computers in mathematics and the mathematical sciences. This is interpreted in a broad sense; computers are used in mathematics not just for approximate numerical calculation but in virtually every area of pure and applied mathematics, including algebra, geometry, number theory, group theory, integration and differential equations.

Each of the conferences in this series has had a strong, interdisciplinary program of invited speakers. In *Computers and Mathematics '89* the contributed research papers have assumed an equally important role. This volume contains the contributed papers accepted for presentation, selected from 85 drafts submitted in response to the call for papers.

The program committee wishes to thank all who submitted papers for consideration and all who assisted in the selection process. The program committee chairman would like to express his thanks to Mrs. Donna Carr for her untiring assistance in the secretarial work.

Johannes Buchmann (Saarbrücken)
Herbert Edelsbrunner (Illinois)
John Fitch (Bath, England)
Keith Geddes (Waterloo)
Erich Kaltofen (Rensselaer), Chairman
Daniel Lazard (Paris)
Michael Overton (New York University)
Fritz Schwarz (GMD Bonn)
Neil Soiffer (Tektronix, Beaverton)
Evelyne Tournier (Grenoble)
Stephen Watt (IBM Research)
Franz Winkler (Linz, Austria)

Portugal Conference

Conferences in the Computers and Mathematics Series

Computer Algebra as a Tool for Research in Mathematics and Physics,
April 5-6, 1984, Courant Institute of Mathematical Sciences, New York.

Computers and Mathematics '86,
July 30-August 1, 1986, Stanford University, California.

Computers and Mathematics '89,
June 13-17, 1989, Massachusetts Institute of Technology.

*next
page*

(B)

Table of Contents

Tuesday June 13, 1989

Session 1, Track A (10:45am-12:30pm)

Chair: Erich Kaltofen

A Completion Procedure for Computing a Canonical Basis for a k -Subalgebra

D. Kapur, SUNY at Albany

K. Madlener, Universität Kaiserslautern 1

Summation of Harmonic Numbers

D. Y. Savio, E. A. Lamagna, S.-M. Lin, The University of Rhode Island 12

Algorithm and Implementation for Computation of Jordan Form Over $A[x_1, \dots, x_m]$

N. Strauss, Pontificia Universidade Catolica do Rio de Janeiro 21

Fast Group Membership Using a Strong Generating Test for Permutation Groups

G. Cooperman, L. Finkelstein, Northeastern University

P. W. Purdom Jr., Indiana University 27

Finite-Basis Theorems and a Computation-Integrated Approach to Obstruction Set Isolation

M. R. Fellows, University of Idaho

N. G. Kinnarsley, M. A. Langston, Washington State University 37

Session 1, Track B (10:45am-12:30pm)

Chair: Stephen Watt

Practical Determination of the Dimension of an Algebraic Variety

A. Galligo, University of Nice and INRIA/Sophia Antipolis, France

C. Traverso, University of Pisa, Italy 46

A Computer Generated Census of Cusped Hyperbolic 3-Manifolds

M. Hildebrand, Harvard University

J. Weeks, Ithaca, New York 53

Classicality of Trigonal Curves of Genus Five

P. Viana, MIT and P.U.C. Pontificia Universidade Católica, Rio de Janeiro 60

Symmetric Matrices with Alternating Blocks

A. Hefez, Univ. Fed. do Esp. Santo, Vitoria, Brazil

A. Thorup, Copenhagen University 66

Cohomology to Compute

D. Leites, Stockholm University

G. Post, University of Twente, The Netherlands 73

Wednesday June 14, 1989

Session 2, Track A (4:00pm-5:45pm)

Chair: Wolfgang Lassner

Use of Symbolic Methods in Analyzing an Integral Operator

H. F. Trotter, Princeton University 82

Computer Algebraic Methods for Investigating Plane Differential Systems of Center and Focus Type

Dongming Wang, Academia Sinica, Beijing 91

An Example of Computer Enhanced Analysis of

P. J. Costa, R. H. Westlake, Raytheon Company, Wayland, MA 100

(KR) ←

<i>An Algorithm for Symbolic Computation of Hopf Bifurcation</i> E. Freire, E. Gamero, E. Ponce, University of Sevilla, Spain	109
<i>Application of the Reduce Computer Algebra System to Stability Analysis of Difference Schemes</i> V. G. Ganzha, Inst. of Theoret. and Appl. Mechanics, Novosibirsk, and T. U. München R. Liska, Technical University of Prague	119

Session 2, Track B (4:00pm-5:45pm)

Chair: Johannes Buchmann

<i>Signs of Algebraic Numbers</i> T. Sakkalis, New Mexico State University, Las Cruces	130
<i>Efficient Reduction of Quadratic Forms</i> N. W. Rickert, Northern Illinois University, DeKalb	135
<i>A Story About Computing with Roots of Unity</i> F. Bergeron, Université du Québec à Montréal	140
<i>Exact Algorithms for the Matrix-Triangularization Subresultant PRS Method</i> A. G. Akritas, University of Kansas	145
<i>Computation of Fourier Transforms on the Symmetric Group</i> D. Rockmore, Harvard University	156

Thursday June 15, 1989

Session 3, Track A (9:00am-10:20am)

Chair: Evelyne Tournier

<i>Integration in Finite Terms and Simplification with Dilogarithms: A Progress Report</i> J. Baddoura, Massachusetts Institute of Technology	166
<i>Why Integration is Hard</i> H. J. Hoover, University of Alberta	172
<i>Liouvillian Solutions of Linear Differential Equations with Liouvillian Coefficients</i> M. F. Singer, North Carolina State University	182
<i>Recipes for Classes of Definite Integrals Involving Exponentials and Logarithms</i> K. O. Geddes, T. C. Scott, University of Waterloo	192

Session 3, Track B (9:00am-10:20am)

Chair: Franz Winkler

<i>Logic and Computation in MATHPERT: An Expert System for Learning Mathematics</i> M. J. Beeson, San Jose State University	202
<i>Representation of Inference in Computer Algebra Systems with Applications to Intelligent Tutoring</i> T. A. Ager, R. A. Ravaglia, Stanford University S. Dooley, University of California at Berkeley	215
<i>Bunny Numerics: A Number Theory Microworld</i> C. Graci, J. Narayan, R. Odendahl, SUNY College at Oswego	228
<i>Advanced Mathematics from an Elementary Viewpoint: Chaos, Fractal Geometry, and Nonlinear Systems</i> W. Feurzeig, P. Horwitz, A. Boulanger, BBN Laboratories, Cambridge, MA	240

Session 4, Track A (10:45am-12:05pm)

Chair: Fritz Schwarz

*Iterated Function Systems and the Inverse Problem of Fractal Construction
Using Moments*

E. R. Vrscay, C. J. Roehrig, University of Waterloo 250

Working with Ruled Surfaces in Solid Modeling

J. K. Johnstone, The Johns Hopkins University 260

Using Macsyma to Calculate the Extrinsic Geometry of a Tube in a Riemannian Manifold

H. S. D. Mills, M. H. Vernon, Lewis Clark State College, Lewiston, Idaho 269

Computer Algebra in the Theory of Ordinary Differential Equations of Halphen Type

V. P. Gerdt, N. A. Kostov, Joint Institute for Nuclear Research, Dubna 279

Session 4, Track B (10:45am-12:05pm)

Chair: Neil Soiffer

Symbolic Derivation of Equations for Mixed Formulation in Finite Element Analysis

H. Q. Tan, The University of Akron 289

Semantics in Algebraic Computation

D. L. Rector, University of California at Irvine 299

Symbolic Computation with Symmetric Polynomials: An Extension to Macsyma

A. Valibouze, LITP, Paris 308

Simultaneous Computations in Fields of Different Characteristics

D. Duval, Université de Grenoble I 321

List of Contributors

T. A. Ager	215	R. Liska	119
A. G. Akritas	145	S.-M. Liu	12
J. Baddoura	166	K. Madlener	1
M. J. Beeson	202	H. S. D. Mills	269
F. Bergeron	140	J. Narayan	228
A. Boulanger	240	R. Odendahl	228
G. Cooperman	27	E. Ponce	109
P. J. Costa	100	G. Post	73
S. Dooley	215	P. W. Purdom Jr.	27
D. Duval	321	R. A. Ravaglia	215
M. R. Fellows	37	D. L. Rector	299
W. Feurzeig	240	N. W. Rickert	135
L. Finkelstein	27	D. Rockmore	156
E. Freire	109	C. J. Roehrig	250
A. Galligo	46	T. Sakkalis	130
E. Gamero	109	D. Y. Savio	12
V. G. Ganzha	119	T. C. Scott	192
K. O. Geddes	192	M. F. Singer	182
V. P. Gerdt	279	N. Strauss	21
C. Graci	228	H. Q. Tan	289
A. Hefez	66	A. Thorup	66
M. Hildebrand	53	C. Traverso	46
H. J. Hoover	172	H. F. Trotter	82
P. Horwitz	240	A. Valibouze	308
J. K. Johnstone	260	M. H. Vernon	269
D. Kapur	1	P. Viana	60
N. G. Kinnersley	37	E. R. Vrscay	250
N. A. Kostov	279	Dongming Wang	91
E. A. Lamagna	12	J. Weeks	53
M. A. Langston	37	R. H. Westlake	100
D. Leites	73		

A Completion Procedure for Computing a Canonical Basis for a k -Subalgebra

Deepak Kapur
Department of Computer Science
State University of New York at Albany
Albany, NY 12222
kapur@albanycs.albany.edu

Klaus Madlener
Fachbereich Informatik
Universität Kaiserslautern
D-6750, Kaiserslautern, W. Germany

Abstract

A completion procedure for computing a canonical basis for a k -subalgebra is proposed. Using this canonical basis, the membership problem for a k -subalgebra can be solved. The approach follows Buchberger's approach for computing a Gröbner basis for a polynomial ideal and is based on rewriting concepts. A canonical basis produced by the completion procedure shares many properties of a Gröbner basis such as reducing an element of a k -subalgebra to 0 and generating unique normal forms for the equivalence classes generated by a k -subalgebra. In contrast to Shannon and Sweedler's approach using tag variables, this approach is direct. One of the limitations of the approach however is that the procedure may not terminate for some orderings thus giving an infinite canonical basis. The procedure is illustrated using examples.

1 Introduction

A procedure and related theory for computing a canonical basis for a finitely presented k -subalgebra are presented. With a slight modification, the procedure can also be used for the membership problem of a unitary subring generated by a finite basis using a Gröbner basis like approach.

The procedure is based on the rewriting approach following Buchberger [1965, 1976, 1985] and Knuth and Bendix [1970]. The structure of the procedure is the same as that of Buchberger's algorithm for computing a Gröbner basis of a polynomial ideal. The definitions of reduction and critical pairs (also called S -polynomials) are different; they can be considered as a generalization of these concepts in Buchberger's algorithm. This approach for solving the membership problem for a k -subalgebra is quite different from the approach taken by Shannon and Sweedler [1987, 1988] in which tag variables are used to transform the subalgebra membership problem to the ideal membership problem. The proposed approach is direct, more in the spirit of the recent work of Robbiano and Sweedler [1988]. However, it is based on rewriting concepts and employs completion using critical pairs.

a G is called a *canonical basis* (or even a *Gröbner basis*) of the k -subalgebra generated by F . The unique normal form of a polynomial p with respect to G is called the *canonical form* of the polynomial p with respect to G . For definitions of various properties of rewriting relations, the reader may consult [Loos and Buchberger]. Below, we assume that polynomials are in the sum of products form and they are simplified (i.e., in a polynomial, there are no terms with zero coefficients, monomials with identical terms are collected together using the operations over the field k).

3 . Making Rules from Polynomials

Let $<$ be a total admissible term ordering which extends to a well-founded ordering on polynomials [Buchberger, 1985]. Let $ht(f)$ be the head-term of f with respect to $<$. For each polynomial f , we can define a rewrite rule (*simplification rule*) as follows (for making a rule, we can assume without any loss of generality that the head-coefficient of f is 1):

$$ht(f) \rightarrow -(f - ht(f)).$$

Associated with a basis $\{f_1, \dots, f_m, \dots\}$ of polynomials is a set $\mathcal{R} = \{L_1 \rightarrow R_1, \dots, L_m \rightarrow R_m, \dots\}$ of rules made as above. We will also use $k[\mathcal{R}]$ to stand for the k -subalgebra generated by $\{f_1, \dots, f_m, \dots\}$. We define a reduction relation induced by $\mathcal{R}$ on polynomials as follows:

$$p \rightarrow q \quad \text{if and only if}$$

- i. $p = ct + p'$, where ct is a monomial in p ($c \in k$, $c \neq 0$, and t is a term) and p' does not have any monomial whose term is t ,
- ii. there are $1 \leq j_1 < j_2 < \dots < j_l$, $l \geq 0$, natural numbers $d_{j_1}, \dots, d_{j_l}$ such that $t = L_{j_1}^{d_{j_1}} L_{j_2}^{d_{j_2}} \dots L_{j_l}^{d_{j_l}}$,
- iii. the term t' in any monomial bigger than ct in p cannot be expressed as a product of powers of the left sides of a non-empty subset of the rules in $\mathcal{R}$, and
- iv. $q = p - c(L_{j_1} - R_{j_1})^{d_{j_1}} (L_{j_2} - R_{j_2})^{d_{j_2}} \dots (L_{j_l} - R_{j_l})^{d_{j_l}}$.

It is easy to see that $p - q \in k[\mathcal{R}]$.

Unlike in Gröbner basis algorithms for polynomial ideals or in term rewriting systems, a single step reduction can thus simultaneously involve many rules.

The third condition above is strictly not necessary but is motivated by implementation concerns. If this condition is not imposed and a weaker reduction relation is defined using (i), (ii) and (iv), in which any monomial (instead of the biggest possible monomial) can be reduced, the results below work also (some proofs may have to be modified though). Using the above definition of a reduction relation, it is possible to consider monomials in descending order for rewriting since any monomial in p once reduced will not reappear in the polynomials obtained by rewriting p .

Even if t satisfies condition (ii) above, we cannot rewrite a proper subterm of t ; we must always rewrite the whole term t . This is so because the polynomial $p - q$ must be in $k[\mathcal{R}]$. Also observe that an element of k always reduces to 0 using any basis by taking $d_{j_1} = \dots = d_{j_l} = 0$. Thus $\mathcal{R}$ need not contain rules corresponding to elements of k as well as the right sides of rules need not have elements of k as monomials.

The proposed approach has a disadvantage however over the indirect approach of Shannon and Sweedler in the sense that for some orderings on indeterminates and terms, the completion procedure may not terminate and thus generate an infinite canonical basis. This raises an interesting open question: Given a finitely presented k -subalgebra, does there exist an ordering on terms for which the completion procedure will terminate? If so, how can such an ordering be computed?

In the next section, definitions are given. Section 3 discusses how rules are made from polynomials, and a reduction relation is defined using a set of rules corresponding to a k -subalgebra basis. Properties of this reduction relation are stated and it is shown that the reduction relation is strong enough so that its reflexive, symmetric and transitive closure is precisely the equivalence relation induced by the associated k -subalgebra. A canonical basis of a k -subalgebra is defined. Section 4 defines superpositions, critical pairs and S-polynomials which lead to a finite test for checking whether a given finite basis of a k -subalgebra is a canonical basis. Section 5 is the main result which shows that if all S-polynomials corresponding to critical pairs of a set of rules reduce to 0, then the corresponding basis is canonical. Section 6 outlines a completion procedure based on the test of Section 5, and properties of canonical bases generated by a completion procedure are discussed. A finite canonical basis always exists for a k -subalgebra over $k[x]$. A number of examples taken from papers by Shannon and Sweedler as well as Robbiano and Sweedler are discussed illustrating the procedure. Some comments on how this approach can be modified to be applicable to unitary subrings are given in the final section. Further details and proofs are given in an expanded version of this paper [Kapur and Madlener, 1989].

2 k -Subalgebras and Canonical bases

Let $k[x_1, \dots, x_n]$ be the polynomial ring over a field k with $x_1, \dots, x_n$ as indeterminates. A unitary subring generated by a finite basis $F = \{f_1, \dots, f_m\}$, each $f_i \in k[x_1, \dots, x_n]$, is the smallest subring containing 1 and the elements of F (i.e., if p and q are in the subring, then $p+q$ as well as $p \cdot q$ are in the subring¹). A k -subalgebra generated by F is the smallest unitary subring generated by F and containing k (see Zariski and Samuel for definitions). Following Shannon and Sweedler, we write this k -subalgebra as $k[f_1, \dots, f_m]$. It is easy to see that a k -subalgebra $k[f_1, \dots, f_m]$ defines an equivalence relation on the polynomial ring $k[x_1, \dots, x_n]$, just like a congruence relation defined by an ideal. Polynomials p and q are equivalent modulo $k[f_1, \dots, f_m]$ if and only if $p - q \in k[f_1, \dots, f_m]$.

Our goal is to compute canonical forms for equivalence classes induced by a k -subalgebra $k[f_1, \dots, f_m]$. We follow the approach proposed by Buchberger [1965, 1985] for computing canonical forms for congruence classes defined by a polynomial ideal. As in Buchberger's approach, with each basis F , we associate a reduction relation $\rightarrow_F$; we will often omit the subscript whenever it is obvious from the context. This reduction relation is associated after first defining a total well-founded ordering on polynomials in $k[x_1, \dots, x_n]$. Such an ordering can be defined in the same way as is usually done in the case of the Gröbner basis algorithm for polynomial ideals using admissible orderings on terms [Buchberger, 1985].

From a given basis F , the goal is to compute another basis (preferably finite) $G = \{g_1, \dots, g_r\}$ such that (i) $k[f_1, \dots, f_m] = k[g_1, \dots, g_r]$, (ii) for every element $p \in k[f_1, \dots, f_m]$, $p \rightarrow_G^* 0$, and (iii) for every element $q \in k[x_1, \dots, x_n]$, q has a unique normal form with respect to $\rightarrow_G$. Further, for any p and q , p and q have the same normal form if and only if $p - q \in k[f_1, \dots, f_m]$. Such

¹Contrast this definition with that of an ideal which is closed under multiplication with respect to any element of the polynomial ring $k[x_1, \dots, x_n]$ instead of only the elements of the subring.

We believe that Robbiano and Sweedler [1988] defined the reduction relation in a similar way except that they consider only the head-term of p instead of any term in p . In their approach, if the head-term cannot be reduced (i.e., cannot be expressed as a product of powers of the left sides of any subset of rules), then the polynomial p cannot be reduced.

Consider the following example from Shannon and Sweedler [1988].

Let $F = \{1. \ x^3 - x, \ 2. \ x^2\}$ be a basis over $Q[x]$. Using the degree ordering, the rules corresponding to the above polynomials are: $\mathcal{R} = \{1. \ x^3 \rightarrow x, \ 2. \ x^2 \rightarrow 0\}$. Any polynomial which has a term whose degree is a multiple of 3 or a multiple of 2, can be reduced using the rule 1 or rule 2 respectively. A polynomial containing x^5 or x^7 as a term can also be reduced using both the rules 1 and 2. However, a monomial with term x cannot be reduced by $\mathcal{R}$. For example, $x^7 - 2x^6 + 3x^5 - 2 \rightarrow -2x^6 + 4x^5 - 2 \rightarrow 4x^5 - 2 \rightarrow 4x^3 - 2 \rightarrow 4x - 2 \rightarrow 4x$. The polynomial $4x$ cannot be reduced further.

3.1 Properties of Reduction Relations

Proposition 3.1: The reduction relation $\rightarrow$ is terminating.

This follows from the fact that (i) the left side of a rule is the head-term of a polynomial with respect to an admissible ordering $<$ which is well-founded and (ii) the reduction relation always completely reduces a monomial by replacing it by a strictly smaller polynomial.

A polynomial p is said to be *irreducible* (or *in normal form*) if and only if there is no q such that $p \rightarrow q$. A polynomial p has a *normal form* q if and only if $p \rightarrow^* q$ and q is in normal form. For example, $4x$ above is a normal form of $x^7 - 2x^6 + 3x^5 - 2$. Thus,

Proposition 3.2: Every $p \in k[x_1, \dots, x_n]$ has a normal form with respect to the reduction relation $\rightarrow$ defined by a set of rules $\mathcal{R}$.

Theorem 3.3: The relation $\leftrightarrow^*$, the reflexive, symmetric and transitive closure of $\rightarrow$, is the k -subalgebra equivalence relation induced by $k[\mathcal{R}]$ associated with $\mathcal{R}$, i.e. for any p and q , $p \leftrightarrow^* q$ if and only if $p - q \in k[\mathcal{R}]$.

The proof of this theorem is very similar to those given in [Buchberger, 1976] for the congruence relation defined by an ideal over a polynomial ring over a field and in [Kandri-Rody and Kapur; 1984] for the congruence relation defined by an ideal over a polynomial ring over a Euclidean domain.

A reduction relation $\rightarrow$ is said to be *canonical* if and only if $\rightarrow$ is terminating and is *confluent*, i.e., for every polynomial p , p has a unique normal form (called the *canonical form* of p) with respect to $\rightarrow$. A basis is called a *canonical basis* if and only if the associated reduction relation $\rightarrow$ is canonical. In the next section, we discuss a finite test for checking whether a basis is a canonical basis using the concepts of superpositions, critical pairs and S-polynomials.

4 Superposition, Critical-pair and S-polynomial

We now define the notions of *superposition* and *critical pairs* for rules in $\mathcal{R} = \{L_1 \rightarrow R_1, \dots, L_m \rightarrow R_m, \dots\}$. Just as a reduction relation is defined using many rules, the critical pair and S-polynomial are defined, in general, using more than two rules (equivalently, polynomials). This is quite different from the definitions of critical pair and S-polynomial in [Buchberger, 1976; 1985] as well as in [Kandri-Rody and Kapur, 1984], or for that matter in term rewriting systems. These definitions can in fact be considered generalizations of the definitions of M -polynomials given in [Kapur and Narendran, 1985]. Below, we give two different ways to define superpositions and S-polynomials; the first one is intuitively appealing whereas the

second one is suitable for computations and proofs.

A finite non-empty subset $\{L_{j_1}, L_{j_2}, \dots, L_{j_l}\}$ of $\mathcal{R}$ is said to *superpose* (or *overlap*) with another disjoint subset $\{L_{i_1}, L_{i_2}, \dots, L_{i_{l'}}\}$ of $\mathcal{R}$ (i.e., rule numbers j_j 's and i_i 's are disjoint) if and only if there is a *minimal* vector of positive natural numbers $\langle d_{j_1}, d_{j_2}, \dots, d_{j_l} \rangle$, which are exponents associated with rules $\{L_{j_1}, L_{j_2}, \dots, L_{j_l}\}$, and another vector of positive numbers $\langle e_{i_1}, e_{i_2}, \dots, e_{i_{l'}} \rangle$, exponents associated with rules $\{L_{i_1}, L_{i_2}, \dots, L_{i_{l'}}\}$, such that

$$L_{j_1}^{d_{j_1}} L_{j_2}^{d_{j_2}} \dots L_{j_l}^{d_{j_l}} = L_{i_1}^{e_{i_1}} L_{i_2}^{e_{i_2}} \dots L_{i_{l'}}^{e_{i_{l'}}}.$$

The vector $\langle d_{j_1}, d_{j_2}, \dots, d_{j_l} \rangle$ is minimal in the sense that for no vector that is smaller than it, there are positive numbers $\langle e_{i_1}, e_{i_2}, \dots, e_{i_{l'}} \rangle$ satisfying the above property about the left sides of the rules ($\langle c_1, c_2, \dots, c_l \rangle$ is smaller than $\langle c'_1, c'_2, \dots, c'_l \rangle$ if and only if they are distinct and each $c_i \leq c'_i, 1 \leq i \leq l$). It is possible to have two non-comparable l' vectors $\langle e_{i_1}, e_{i_2}, \dots, e_{i_{l'}} \rangle$ and $\langle e'_{i_1}, e'_{i_2}, \dots, e'_{i_{l'}} \rangle$ for the same minimal k -vector $\langle d_{j_1}, d_{j_2}, \dots, d_{j_l} \rangle$ satisfying the above property about the left sides of the rules. In that case, the rule subset $\{L_{j_1}, L_{j_2}, \dots, L_{j_l}\}$ is said to superpose in more than one ways.

The *critical pair* associated with this superposition is:

$$\begin{aligned} & \langle L_{j_1}^{d_{j_1}} L_{j_2}^{d_{j_2}} \dots L_{j_l}^{d_{j_l}} - (L_{j_1} - R_{j_1})^{d_{j_1}} (L_{j_2} - R_{j_2})^{d_{j_2}} \dots (L_{j_l} - R_{j_l})^{d_{j_l}}, \\ & L_{i_1}^{e_{i_1}} L_{i_2}^{e_{i_2}} \dots L_{i_{l'}}^{e_{i_{l'}}} - (L_{i_1} - R_{i_1})^{e_{i_1}} (L_{i_2} - R_{i_2})^{e_{i_2}} \dots (L_{i_{l'}} - R_{i_{l'}})^{e_{i_{l'}}} \rangle. \end{aligned}$$

The *S-polynomial* corresponding to the critical pair is:

$$(L_{j_1} - R_{j_1})^{d_{j_1}} (L_{j_2} - R_{j_2})^{d_{j_2}} \dots (L_{j_l} - R_{j_l})^{d_{j_l}} - (L_{i_1} - R_{i_1})^{e_{i_1}} (L_{i_2} - R_{i_2})^{e_{i_2}} \dots (L_{i_{l'}} - R_{i_{l'}})^{e_{i_{l'}}}.$$

It is obvious that the *S-polynomials* of $\mathcal{R}$ belong to $k[\mathcal{R}]$.

The set of critical pairs for a finite set $\mathcal{R}$ of rules is always finite; a bound can be computed using the degree of the indeterminates appearing in the left sides of rules [Stickel, 1981; Huet, 1978]. The finiteness of the number of critical pairs also follows from the fact that the vectors of exponents $\langle d_1, \dots, d_m, e_1, \dots, e_m \rangle$, with $d_j, e_i \geq 0$, satisfying the following equation, form an abelian monoid which has a finite basis.

$$L_1^{d_1} L_2^{d_2} \dots L_m^{d_m} = L_1^{e_1} L_2^{e_2} \dots L_m^{e_m}$$

An alternate way of computing the exponents of the left sides of rules above is to set up a finite set of diophantine equations from the left sides of rules for a finite $\mathcal{R}^2$. For each indeterminate x_i , there is a linear diophantine equation

$$d_1 v_{i_1} + d_2 v_{i_2} + \dots + d_m v_{i_m} = e_1 v_{i_1} + e_2 v_{i_2} + \dots + e_m v_{i_m},$$

where $v_{i_1}, \dots, v_{i_m}$ are, respectively, the degrees of x_i in the left sides of rules $1, \dots, m$. So there are n such linear diophantine equations. These equations are solved for $d_1, \dots, d_m$ and $e_1, \dots, e_m$ and a basis of *minimal* non-zero simultaneous solutions in which if $d_j \neq 0$, then $e_j = 0$ and if $e_i \neq 0$, then $d_i = 0$, can be computed. Using these basis vectors, any solution to these simultaneous equations can be obtained as a nonnegative linear combination of the vectors in the basis (i.e., the multipliers are nonnegative). Further, only one of the two solutions $\langle d_1, \dots, d_m, e_1, \dots, e_m \rangle$ and $\langle e_1, \dots, e_m, d_1, \dots, d_m \rangle$ need to be considered because of the symmetric nature of the diophantine equations. These equations can be solved using algorithms proposed for solving linear diophantine equations arising in associative-commutative unification problems [Stickel, 1981; Huet, 1978]. The finiteness of a basis from which all solutions to the above set of equations can be generated, also follows from the results related to these algorithms.

It will be interesting to compare these definitions with the corresponding concepts in Robbiano and Sweedler's approach.

²This formulation however extends to be applicable to an infinite $\mathcal{R}$ also.

5 A Test for a Canonical Basis

The following is a Church-Rosser theorem for k -subalgebras.

Theorem 5.1: The set $\mathcal{R} = \{L_1 \rightarrow R_1, \dots, L_m \rightarrow R_m, \dots\}$ is canonical or equivalently, the corresponding basis $\{f_1, \dots, f_m, \dots\}$ is a canonical basis if and only if all S-polynomials generated using every finite subset of $\mathcal{R}$ reduce to 0.

The proof of the theorem uses the following lemmas.

Lemma 5.2: If $p \rightarrow^* 0$, then for any $L_i \rightarrow R_i \in \mathcal{R}$, $(L_i - R_i)p \rightarrow^* 0$.

Note that it is not necessarily the case that $t p \rightarrow^* 0$ for any term t or even for $t = L_i$, the left side of a rule in $\mathcal{R}$.

It follows by induction from the above lemma that

Corollary 5.3: If $p \rightarrow^* 0$, for any $c \in k$, and any natural numbers $d_{j_1}, \dots, d_{j_l}$,
 $(c(L_{j_1} - R_{j_1})^{d_{j_1}} \dots (L_{j_l} - R_{j_l})^{d_{j_l}} p) \rightarrow^* 0$.

In addition, we have:

Lemma 5.4: If $p_1 - p_2 \rightarrow^* 0$, then p_1 and p_2 are joinable, i.e., there is a q such that $p_1 \rightarrow^* q$ and $p_2 \rightarrow^* q$.

Sketch of Proof of Theorem 5.1: The only if part of the proof is easy and is omitted. The proof of the if part follows. Consider a polynomial p that can be reduced in two different ways. Since the reduction is defined by rewriting the biggest possible monomial which can be reduced using a finite subset of rules in $\mathcal{R}$, the only case to consider is when p has a monomial ct' which can be reduced in two different ways and no monomial greater than ct' in p can be reduced. So there exist two m -vectors $\langle a_1, \dots, a_m \rangle$ and $\langle b_1, \dots, b_m \rangle$ with some a_i and b_j possibly 0, such that $t' = L_1^{a_1} \dots L_m^{a_m} = L_1^{b_1} \dots L_m^{b_m}$ and $p \rightarrow p_1 = p - c(L_1 - R_1)^{a_1} \dots (L_m - R_m)^{a_m}$ as well as $p \rightarrow p_2 = p - c(L_1 - R_1)^{b_1} \dots (L_m - R_m)^{b_m}$. Let $c_1 = \min(a_1, b_1), \dots, c_m = \min(a_m, b_m)$; c_i 's correspond to the common powers of the rules applied on both sides. Let $d_1 = a_1 - c_1, \dots, d_m = a_m - c_m$ and $e_1 = b_1 - c_1, \dots, e_m = b_m - c_m$. Because of Corollary 5.3, and Lemma 5.4, it suffices to show that

$$q = (L_1 - R_1)^{d_1} \dots (L_m - R_m)^{d_m} - (L_1 - R_1)^{e_1} \dots (L_m - R_m)^{e_m} \rightarrow^* 0, \quad (*)$$

such that for any i , if $d_i \neq 0$ then $e_i = 0$, and if $e_i \neq 0$ then $d_i = 0$.

This is shown by Noetherian induction using the well-founded ordering $<$ on the head-term $t = L_1^{d_1} \dots L_m^{d_m} = L_1^{e_1} \dots L_m^{e_m}$. The basis step of $t = 1$ is trivial. The induction hypothesis is to assume that $(*)$ holds for $t' < t$.

There are two cases: (i) t cannot be decomposed into $t_1 \neq 1$ and $t_2 \neq 2$ such that $t = t_1 t_2$, and both t_1 and t_2 can be reduced by the rules in $\mathcal{R}$. This implies that the exponent vector $\langle d_1, \dots, d_m, e_1, \dots, e_m \rangle$ belongs to a minimal basis set of solutions obtained from diophantine equations associated with $\mathcal{R}$ since this exponent vector cannot be expressed as a sum of two non-zero exponent vectors. The S-polynomial corresponding to this exponent vector reduces to 0 by the assumption that all the S-polynomials of $\mathcal{R}$ reduce to 0.

(ii) $t = t_1 t_2$, and $t_1, t_2 \neq 1$: By the induction hypothesis, for $i = 1, 2$,

$$s_i = (L_1 - R_1)^{d_{i1}} \dots (L_m - R_m)^{d_{im}} - (L_1 - R_1)^{e_{i1}} \dots (L_m - R_m)^{e_{im}} \rightarrow^* 0,$$

where $t_i = L_1^{d_{i1}} \dots L_m^{d_{im}} = L_1^{e_{i1}} \dots L_m^{e_{im}}$. Obviously, $d_j = d_{1j} + d_{2j}$ and $e_j = e_{1j} + e_{2j}$.

If s_1 reduces to 0 in l_1 reduction steps by reducing terms $r_{11}, \dots, r_{1l_1}$ in the first, $\dots$, l_1 -th step, respectively, then by Corollary 5.3, $(L_1 - R_1)^{d_{21}} \dots (L_m - R_m)^{d_{2m}} s_1$ also reduces to 0 in exactly

l_1 steps by reducing terms $t_2 r_{11}, \dots, t_2 r_{1l_1}$ in the respective steps. A similar reduction sequence can be obtained for $(L_1 - R_1)^{e_1} \dots (L_m - R_m)^{e_{1m}} s_2$ reducing to 0 from the reduction sequence $s_2 \rightarrow^2 0$. Now $q = (L_1 - R_1)^{e_1} \dots (L_m - R_m)^{e_{2m}} s_1 + (L_1 - R_1)^{e_{11}} \dots (L_m - R_m)^{e_{1m}} s_2$ and a reduction sequence from q to 0 can be constructed by appropriately mixing the reduction steps from the above reduction sequences and additional reduction sequences available using the induction hypothesis. These details can be found in the proof given in an expanded version of this paper [Kapur and Madlener, 1989].

Thus $\mathcal{R}$ is a canonical basis. $\square$

6 Completion Procedure

From the above theorem, one also gets a completion procedure similar to Buchberger's Gröbner basis algorithm [1985] or the Knuth-Bendix procedure [1970] (see also Huet, 1981) whose correctness can be established using methods similar to the one given in Buchberger's papers. If a given basis of a k -subalgebra is not a canonical basis, then it is possible to generate a canonical basis equivalent to a given basis of a k -subalgebra using the completion procedure. For every S-polynomial of a basis that does not reduce to 0, the current basis is augmented with a normal form of the S-polynomial and the basis is inter-reduced. This process of generating S-polynomials, checking whether they reduce to 0, and augmenting the basis with normal forms of S-polynomials is continued until all S-polynomials of the final basis reduce to 0. Optimizations and heuristics can be introduced into the completion procedure in regards to the order in which various finite subsets of a basis are considered; further, since a finite subset of a basis may result in many S-polynomials, if some S-polynomial results in a new rule which simplifies any rule in the subset under consideration, then the subset does not have to be considered.

Unlike Gröbner basis algorithms, this process of adding new polynomials to a basis may not always terminate. An example below illustrates the divergence of the completion procedure. We consider this a major limitation of this approach in contrast to Shannon and Sweedler's approach. However, the following results are immediate consequences of general results in term rewriting theory [Huet, 1981; Butler and Lankford, 1980; Avenhaus, 1985; Dershowitz et al, 1988] since orderings on polynomials are total, thus a rule can always be made from a polynomial, and the completion procedure will never abort because of the inability to make a rule.

Theorem 6.1: If a completion procedure follows a *fair strategy* in computing superpositions and critical pairs, then the completion procedure serves as a semi-decision procedure for k -subalgebra membership even when the completion procedure does not terminate.

Theorem 6.2: Given a polynomial ordering $<$, if a k -subalgebra has a finite canonical basis with respect to $<$, then a completion procedure with a fair strategy would generate a finite canonical basis.

Further, such a finite canonical basis is unique with respect to $<$ if it is reduced (i.e., for every polynomial in the basis, none of its monomials can be reduced using the remaining set of polynomials in the basis).

A strategy is called *fair* if and only if all superpositions among all possible finite subsets of rules are eventually considered. There can be many ways to generate superpositions and critical pairs which would constitute a fair strategy. A simple fair strategy is to consider superpositions in the degree ordering irrespective of the ordering $<$ used for making rules from polynomials.

For the univariate case, the completion procedure always terminates.

Theorem 6.3: A k -subalgebra over $k[x]$ always has a finite canonical basis which is gener-

ated by the completion procedure.

Sketch of Proof: Suppose r polynomials with the degrees $d_1, \dots, d_r$ are already generated in a basis. There is a number $\text{bound}(d_1, \dots, d_r)$ that is a multiple of $d = \gcd(d_1, \dots, d_r)$ such that every multiple of d which is $\geq \text{bound}(d_1, \dots, d_r)$ can be expressed as a nonnegative linear combination of $\{d_1, \dots, d_r\}$. Since a polynomial will be added to the basis only if the degree of its head-term cannot be expressed as a nonnegative linear combination of $\{d_1, \dots, d_r\}$, one can only add to the basis, polynomials of degree $< \text{bound}(d_1, \dots, d_r)$ or of degree d_{r+1} which is not a multiple of d . In the second case, the $\gcd(d_1, \dots, d_r, d_{r+1}) < d$. $\square$.

6.1 Examples

Example 1: Consider the example from Shannon and Sweedler [1988] which was discussed earlier. The basis is $F_1 = \{1. x^3 - x, 2. x^2\}$ and the rules corresponding to the basis are: $\mathcal{R}_1 = \{1. x^3 \rightarrow x, 2. x^2 \rightarrow 0\}$. A critical pair can be obtained by solving the following diophantine equation:

$$3d_1 + 2d_2 = 3e_1 + 2e_2.$$

The basis of the solutions to this equation is: $\langle 2, 0, 0, 3 \rangle$. The superposition is x^6 and the critical pair is: $\langle 2x^4 - x^2, 0 \rangle$. The S-polynomial $2x^4 - x^2$ reduces to 0. So, F is a canonical basis. In contrast, Shannon and Sweedler's approach using tagged variables will have to perform more complex computations to get a Gröbner basis involving tag variables.

Example 2: Let us consider an example given by Robbiano, $F_2 = \{x^3, x^4, x^5 + x^2 + x\}$. The rules corresponding to them are:

$$\mathcal{R}_2 = \{1. x^3 \rightarrow 0, 2. x^4 \rightarrow 0, 3. x^5 \rightarrow -x^2 - x\}.$$

Superpositions and critical pairs can be computed by setting up a diophantine equation:

$$3d_1 + 4d_2 + 5d_3 = 3e_1 + 4e_2 + 5e_3.$$

A minimal basis for the solutions to the above equation is:

$$\{\langle 1, 0, 1, 0, 2, 0 \rangle, \langle 0, 1, 1, 3, 0, 0 \rangle, \langle 0, 0, 2, 2, 1, 0 \rangle, \langle 4, 0, 0, 0, 3, 0 \rangle, \langle 5, 0, 0, 0, 0, 3 \rangle, \langle 1, 3, 0, 0, 0, 3 \rangle, \langle 0, 5, 0, 0, 0, 4 \rangle\}.$$

Corresponding to the first solution, the critical pair is obtained by a superposition generated by the product of the left sides of rules 1 and 3 which is equal to the square of the left side of rule 2. The superposition is x^8 , and the critical pair is $\langle x^5 + x^4, 0 \rangle$. The S-polynomial $x^5 + x^4$ can be reduced to its normal form $-x^2 - x$. This means that the given basis is not a canonical basis.

A canonical basis can be obtained however by augmenting the original basis with normal forms of S-polynomials thus computed and repeating this process. So the superposition x^8 gives an additional rule

$$4. x^2 \rightarrow -x.$$

It is always better to use this rule to simplify the existing rules. Rule 2 gets simplified to x thus giving

$$2'. x \rightarrow 0.$$

Rule 2' deletes every other rule. As a result, we did not have to consider critical pairs generated by the rules 1, 2, and 3, which got deleted. This is in contrast to having to consider all superpositions generated from the basis solutions of the above diophantine equation which would have resulted in unnecessary computations.

The basis $\{x\}$ is a canonical basis for the k -subalgebra generated by $\{x^3, x^4, x^5 + x^2 + x\}$. It should be easy to see from Theorem 5.1 that a singleton basis is always a canonical basis.

Example 3: Let us consider another example given by Robbiano, $F_2 = \{x, y^2 - xy, xy^2\}$. Using the total degree ordering defined by the ordering $y > x$, we get

$$\mathcal{R}_3 = \{1. x \rightarrow 0, 2. y^2 \rightarrow xy, 3. xy^2 \rightarrow 0\}.$$

Rule 3 is normalized using rules 1 and 2 to 3'. $x^2y \rightarrow 0$.

This set of rules gives the following diophantine equations for generating superpositions:

$$d_1 + 2d_3 = e_1 + 2e_3,$$

$$2d_2 + d_3 = 2e_2 + e_3.$$

A basis of common solutions to these equations is: $\{< 4, 1, 0, 0, 0, 2 >\}$. So, the only superposition is generated by squaring the left side of rule 3' which is equal to the product of the fourth power of the left side to rule 1 and the left side of rule 2. The critical pair is $< 0, x^5y >$ and its S-polynomial x^5y reduces to 0. So, $\{x, y^2 - xy, x^2y\}$ is a canonical basis for the k -subalgebra generated by $\{x, y^2 - xy, xy^2\}$.

For the same basis, if we use a different total degree ordering defined by the ordering $x > y$, then the rule set is slightly different:

$$\{1. x \rightarrow 0, 2. xy \rightarrow y^2, 3. xy^2 \rightarrow 0\}.$$

A critical pair is generated by identifying that the square of the left side of rule 2 is equal to the product of the left sides of rules 1 and 3. The S-polynomial in this case is $2xy^3 - y^4$, which gives a new rule:

$$4. xy^3 \rightarrow 1/2 y^4.$$

There is no other superposition from the first three rules.

Rules 1 and 4 superpose with rules 2 and 3. The superposition is x^2y^3 and the critical pair is $< 1/2xy^4, xy^4 >$. The corresponding S-polynomial $1/2xy^4$ does not reduce any further, which gives a rule:

$$5. xy^4 \rightarrow 0.$$

Also, the cube of the left side of rule 2 is equal to the product of the left side of rule 4 with the square of the left side of rule 1. This gives the critical pair $< y^6 - 3x^2y^4 + 3xy^5, 1/2 x^2y^4 >$. If the corresponding S-polynomial is reduced, we obtain $y^6 - 3xy^5$ which gives another new rule:

$$6. xy^5 \rightarrow 1/3 y^6.$$

There is again a superposition since the fifth power of the left side of rule 2 is equal to the product of the left side of rule 5 and the fourth power of the left side of rule 1. This gives another new rule. It can be shown that this process of generating new rules continue and does not terminate thus resulting in an infinite canonical basis, which is $\{xy^{2j+1} - 1/(j+1)y^{2j+2}, xy^{2j} \mid j \geq 0\}$.

This example illustrates that unlike in the case of ideals over Noetherian rings, this completion procedure need not terminate. Further, there are finitely generated k -subalgebras which have finite canonical basis under one ordering but do not have a finite canonical basis under a different ordering. These observations are similar to the ones made by Robbiano and Sweedler. The situation here is very similar to the one encountered in generating canonical basis for finitely presented monoids (Thue systems) or for first-order equational theories; the reduction orderings there also affect the termination of completion procedures.

It will be interesting to compare the computational performance of the above procedure with Shannon and Sweedler's method.

7 Extension to Unitary Subrings

The approach discussed in this paper can be generalized to compute canonical bases for unitary subrings. The coefficient field k is not included in every unitary subring even though Z is included in every unitary subring if k is of characteristic 0. When rules are made, the coefficient of the left side cannot be made 1, and the definition of reduction relation must be modified to take into consideration the coefficients of the head-terms also. Similarly, the definitions of superpositions, critical pairs and S-polynomials have to be changed to deal with the coefficients. We are also investigating the extensions of this approach to unitary subrings of $R[x_1, \dots, x_n]$, where R is a commutative ring.

Acknowledgement: Most of this work was initiated while the first author visited University of Kaiserslautern during the summer of 1986 and during the winter of 1986 at the Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS) at New Delhi. The first author would like to thank Lorenzo Robbiano for his encouragement to write this work; this paper would not have been written but for Robbiano's prodding after his talk on this subject at a Gröbner basis workshop in October 1988 at the Mathematical Sciences Institute of Cornell University. We also thank Moss Sweedler for many helpful comments on the first draft of this paper.

References

1. Avenhaus, J. (1985). On the termination of the Knuth-Bendix algorithm. Report 120/84, Universität Kaiserslautern, West German.
2. Buchberger, B. (1965). *An algorithm for finding a basis for the residue class ring of a zero-dimensional polynomial ideal.* (in German) Ph.D. Thesis, Univ. of Innsbruck, Austria.
3. Buchberger, B. (1976). A theoretical basis for the reduction of polynomials to canonical forms. *ACM-SIGSAM Bulletin* 10/3, 39, 19-29.
4. Buchberger, B., and Loos, R. (1982). Algebraic simplification. In: *Computer algebra: Symbolic and algebraic computation.* (eds. Buchberger, Collins, and Loos), Computing Suppl. 4, Springer Verlag, 11-43.
5. Buchberger, B. (1985). Gröbner bases: An algorithmic method in polynomial ideal theory. In: N.K. Bose (ed.) *Multidimensional Systems Theory*, Reidel, 184-232.
6. Butler, G., and Lankford, D.S. (1980). Experiments with computer implementations of procedures which often derive decision algorithms for the word problem in abstract algebras. Memo MTP-7, Dept. of Mathematics, Louisiana Tech. University, Ruston, LA, August 1980.
7. Dershowitz, N., Marcus, L., and Tarlecki, A. (1988). Existence, uniqueness, and construction of rewrite systems. *SIAM J. of Computing*, to appear.
8. Huet, G. (1978). An algorithm to generate the basis of solutions to homogeneous linear diophantine equations. *Information Processing letters*, 7, 3, 144-147.
9. Huet, G. (1981). A complete proof of correctness of the Knuth-Bendix completion procedure. *J. of Computer and Systems Sciences*, 23, 1, 11-21.
10. Kandri-Rody, A., and Kapur, D. (1984). An algorithm for computing a Gröbner basis of a polynomial ideal over a Euclidean ring. G.E. Corporate Research and Development Report 84C&D045, Schenectady, NY. A revised version appeared in *J. of Symbolic Computation*, August 1988.

11. Kapur, D., and Madlener, K. (1989). A completion procedure for computing a canonical basis for a k -subalgebra. Technical Report 89-5, Department of Computer Science, State University of New York at Albany, NY. An expanded version of this paper.
12. Kapur, D., and Narendran, P. (1985). Existence and construction of a Gröbner basis for a polynomial ideal. Presented at a workshop on *Combinatorial algorithms in algebraic structures*, Europäische Akademie, Otzenhausen, W.-Germany.
13. Knuth, D., and Bendix, P. (1970). Simple word problems in universal algebras. In: J. Leech (ed.) *Computational Problems in Abstract Algebras*, Pergamon Press.
14. Lankford, D. S., and Ballantyne, A. M. (1983). On the uniqueness of term rewriting systems. Unpublished note, Dept. of Mathematics, Louisiana Tech. University, Ruston, LA.
15. Robbiano, L., and Sweedler, M. (1988). Computing a canonical basis of a k -subalgebra. Presented at a Mathematical Sciences Institute workshop on Gröbner bases, Cornell University.
16. Shannon, D., and Sweedler, M. (1988). Using Gröbner bases to determine algebra membership, split surjective algebra homomorphisms and determine birational equivalence. *J. of Symbolic Computation*, 6, 267-273.
17. Shannon, D., and Sweedler, M. (1988). Using Gröbner bases to determine subalgebra membership. Unpublished Manuscript, Dept. of Mathematics, Cornell University.
18. Stickel, M.E. (1981). A unification algorithm for associative-commutative unification. *Journal ACM*, 28, 3, 423-434.
19. Zariski, O., and Samuel, P. (1958). *Commutative Algebra, Vol. I*. Springer Verlag, New York.

Summation of Harmonic Numbers

Dominic Y. Savio, Edmund A. Lamagna, and Shing-Min Liu

Department of Computer Science and Statistics

The University of Rhode Island

Kingston, Rhode Island 02881

Abstract. *The problem of finding closed forms for a summation involving harmonic numbers is considered. Solutions for $\sum_{i=1}^n p(i)H_i^{(k)}$, where $p(i)$ is a polynomial, and $\sum_{i=1}^n H_i/(i+m)$, where m is an integer, are given. A method to automate these results is presented. This is achieved by using Moenck's algorithm and by exploiting the relationship between polygamma functions and harmonic numbers.*

1. Introduction

We are interested in developing an interactive system which will aid both experts and students in the analysis of algorithms. This system should be capable of assisting analysts in dealing with summation of functions, solution of recurrence relations, manipulation of generating functions, and asymptotic analysis. As a first step, one of the members of our research group has implemented several basic tools including Gosper's summation algorithm, a definite summation procedure for binomial sums, a series of methods for determining the convergence of infinite sums, and several basic techniques for solving large classes of recurrence equations [3].

The indefinite summation problem is concerned with finding closed form solutions $S(n)$ to

$$S(n) - S(0) = \sum_{i=1}^n f(i),$$

where $f(i)$ is an arbitrary expression from some class. Three procedures have been developed, each of which is capable of producing results the others cannot. Moenck's algorithm considers the case where the summand, $f(i)$, is a polynomial or a rational function and expresses the transcendental part of the answer in terms of polygamma functions [11]. Karr's procedure poses the problem in an algebraic way and then derives conditions for summability [6,7]. Starting with a field of constants, larger fields are constructed by the formal adjunction of symbols which behave like solutions to first

order linear equations. Then, in these extension fields, the difference equations are posed and solutions are sought. Gosper's procedure works whenever $S(n) = \sum_{i=1}^n f(i)$ is such that $S(n)/S(n-1)$ is a rational function of n [1,2].

One is usually concerned with definite sums in the analysis of algorithms. When the indefinite sum is known, the problem is trivially solved. To complicate matters, we frequently encounter special classes of functions in the analysis of algorithms like binomial coefficients, harmonic numbers, and Fibonacci numbers. We have implemented algorithms to solve many types of definite summation problems involving binomial coefficients [4] and have considered the problem of deriving closed forms for the summation of harmonic numbers [10].

The average number of comparisons to find the maximum of n items, is given by the n th harmonic number, $H_n = 1 + 1/2 + 1/3 + \dots + 1/n$. The expected performance of many sorting algorithms is expressed in terms of these functions. Thus, we are motivated to solve summation problems involving them. Graham, Knuth, and Patashnik [9] mention that it would be "nice" to automate the derivation of such formulas. Karr's procedure will yield formulas, as a rational function of n and the symbol H_n , for the sums $\sum_{i=1}^n H_i$ and $\sum_{i=1}^n iH_i$. However, it will give no formula for the sum $\sum_{i=1}^n H_i/i$, since the generalized harmonic number $H_n^{(2)}$ must be adjoined to the field. Gosper's procedure will not handle harmonic numbers at all since they are not rational functions of n . Moenck's algorithm cannot solve the sum $\sum_{i=1}^n H_i$ since it involves a double summation.

In this paper, we will investigate a method to find closed forms for certain types of summation involving harmonic numbers. This will be achieved by employing a relationship between harmonic numbers and polygamma functions and by using Moenck's algorithm.

2. Summation of Harmonic Numbers

As defined in *The Art of Computer Programming* by Knuth [8],

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \sum_{k=1}^n \frac{1}{k}, \quad n \geq 1. \quad (1)$$

It may seem at first glance that H_n is bounded, even when n has a large value, since we are always adding smaller and smaller terms. But it is not hard to see that H_n can get as large as we please if we take n to be big enough. In a sense, H_n "just barely" goes to infinity as n gets larger because it can be proved that the sum $1 + \frac{1}{2^k} + \frac{1}{3^k} + \dots + \frac{1}{n^k}$ stays bounded for all $k > 1$. The generalized harmonic numbers $H_n^{(k)}$ are defined as

$$H_n^{(k)} = 1 + \frac{1}{2^k} + \frac{1}{3^k} + \dots + \frac{1}{n^k}, \quad k \geq 1. \quad (2)$$

From the viewpoint of realistic algorithm analysis and from the problems encoun-

tered in textbooks, we have good reason to categorize the commonly occurring sums for expressions involving harmonic numbers into one of two cases. They are either of the form

$$\sum_{i=1}^n p(i)H_i, \quad (3)$$

where $p(i)$ is a polynomial in i , or of the form

$$\sum_{i=1}^n f(i)H_i, \quad (4)$$

where $f(i)$ is a rational function in i .

Consider the case when the sum is in the form of equation (3) and $p(i) = a_m i^m + \dots + a_1 i + a_0$, $a_i \in \mathbb{C}$. As long as we can work the sum for any power of i times a harmonic number (i.e., $\sum_{i=1}^n i^m H_i$, $m \in \mathbb{N}$), we can also solve the problem of (3). On the other hand, in the case of (4), we need to consider the rational function $1/(i+m)$, $m \in \mathbb{I}$.

Let us look at the sum $\sum_{i=1}^n H_i^{(k)}$, where $k \in \mathbb{N}$. We first treat the smallest case, the sum of harmonic numbers $\sum_{i=1}^n H_i$. Writing $\sum_{j=1}^i 1/j$ in place of H_i and changing the order of summation, we find that

$$\begin{aligned} \sum_{i=1}^n H_i &= \sum_{i=1}^n \sum_{j=1}^i \frac{1}{j} = \sum_{j=1}^n \sum_{i=j}^n \frac{1}{j} = \sum_{j=1}^n \frac{(n+1-j)}{j} \\ &= \sum_{j=1}^n \frac{(n+1)}{j} - \sum_{j=1}^n 1 \\ &= (n+1) \sum_{j=1}^n \frac{1}{j} - \sum_{j=1}^n 1 \\ &= (n+1)H_n - n. \end{aligned} \quad (5)$$

Using the same rule of transformation of sums, we also see that

$$\begin{aligned} \sum_{i=1}^n H_i^{(2)} &= \sum_{i=1}^n \sum_{j=1}^i \frac{1}{j^2} = \sum_{j=1}^n \sum_{i=j}^n \frac{1}{j^2} = \sum_{j=1}^n \frac{n+1-j}{j^2} \\ &= \sum_{j=1}^n \frac{n+1}{j^2} - \sum_{j=1}^n \frac{j}{j^2} \\ &= (n+1) \sum_{j=1}^n \frac{1}{j^2} - \sum_{j=1}^n \frac{1}{j} \\ &= (n+1)H_n^{(2)} - H_n. \end{aligned} \quad (6)$$

From (5) and (6), it is obvious that

$$\sum_{i=1}^n H_i^{(k)} = (n+1)H_n^{(k)} - H_n^{(k-1)}, \quad (7)$$

where $k \in \mathbb{N}$.

Next we consider $\sum_{i=1}^n i^m H_i$, where $m \in \mathbb{N} \cup \{0\}$. We have seen the simplest case, $m = 0$, before in equation (5). When $m = 1$, we can apply the same technique of interchanging the order of summation to obtain

$$\begin{aligned} \sum_{i=1}^n i H_i &= \sum_{i=1}^n i \sum_{j=1}^i \frac{1}{j} = \sum_{j=1}^n \frac{1}{j} \sum_{i=j}^n i \\ &= \sum_{j=1}^n \frac{1}{j} \left[\frac{n(n+1)}{2} - \frac{j(j-1)}{2} \right] \\ &= \frac{n(n+1)}{2} \sum_{j=1}^n \frac{1}{j} - \frac{1}{2} \sum_{j=1}^n (j-1) \\ &= \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}. \end{aligned} \quad (8)$$

This technique also works for the general case, i.e., summation of any power of i times a harmonic number,

$$\sum_{i=1}^n i^m H_i = H_n \sum_{i=1}^n i^m - \sum_{j=1}^n \frac{1}{j} \sum_{i=1}^{j-1} i^m, \quad (9)$$

where m is a non-negative integer. In a manner similar to the derivation of equations (8) and (9), it can be shown that $\sum_{i=1}^n i^m H_n^{(2)}$ is of the form

$$H_n^{(2)} \sum_{i=1}^n i^m - \sum_{j=1}^n \frac{1}{j^2} \sum_{i=1}^{j-1} i^m. \quad (10)$$

In fact, equations (5) - (10) lead to the following theorem.

Theorem 1.

$$\sum_{i=1}^n i^m H_i^{(k)} = H_n^{(k)} \sum_{i=1}^n i^m - \sum_{j=1}^n \frac{1}{j^k} \sum_{i=1}^{j-1} i^m \quad (11)$$

where $m \in \mathbb{N} \cup \{0\}$ and $k \in \mathbb{N}$.

Next we consider

$$\sum_{i=1}^n \frac{H_i}{i+m}, \quad m \in \mathbb{I}.$$

First let us treat the case when $m = 0$, $\sum_{i=1}^n H_i/i$. Again writing the definition of H_i and transforming the sums, we have

$$\begin{aligned} \sum_{i=1}^n \frac{H_i}{i} &= \sum_{i=1}^n \frac{1}{i} \sum_{j=1}^i \frac{1}{j} = \sum_{j=1}^n \frac{1}{j} \sum_{i=j}^n \frac{1}{i} \\ &= \sum_{j=1}^n \frac{1}{j} [H_n - H_{j-1}] \end{aligned}$$

$$\begin{aligned}
&= H_n \sum_{j=1}^n \frac{1}{j} - \sum_{j=1}^n \frac{1}{j} \left[H_j - \frac{1}{j} \right] \\
&= H_n H_n - \sum_{j=1}^n \frac{H_j}{j} + \sum_{j=1}^n \frac{1}{j^2},
\end{aligned}$$

so

$$2 \sum_{i=1}^n \frac{H_i}{i} = H_n^2 + H_n^{(2)}$$

and

$$\sum_{i=1}^n \frac{H_i}{i} = \frac{1}{2} (H_n^2 + H_n^{(2)}). \quad (12)$$

Now let us see what happens to $\sum_{i=1}^n H_i/(i+2)$.

$$\begin{aligned}
\sum_{i=1}^n \frac{H_i}{i+2} &= \sum_{i=1}^n \frac{1}{i+2} \sum_{j=1}^i \frac{1}{j} = \sum_{j=1}^n \frac{1}{j} \sum_{i=j}^n \frac{1}{i+2} \\
&= \sum_{j=1}^n \frac{1}{j} [H_{n+2} - H_{j+1}] \\
&= H_{n+2} \sum_{j=1}^n \frac{1}{j} - \sum_{j=1}^n \frac{1}{j} \left(H_j + \frac{1}{j+1} \right) \\
&= H_{n+2} H_n - \sum_{j=1}^n \frac{H_j}{j} - \sum_{j=1}^n \frac{1}{j(j+1)} \\
&= H_{n+2} H_n - \frac{1}{2} (H_n^2 + H_n^{(2)}) - \sum_{j=1}^n \frac{1}{j(j+1)} \\
&= H_{n+2} H_n - \frac{1}{2} (H_n^2 + H_n^{(2)}) - \frac{n}{n+1}.
\end{aligned} \quad (13)$$

Equations (12) and (13) can be generalized by the following theorems.

Theorem 2.

$$\sum_{i=1}^n \frac{H_i}{i+m} = H_n H_{n+m} - \frac{1}{2} (H_n^2 + H_n^{(2)}) - \sum_{i=1}^n \frac{1}{i} \sum_{j=1}^{m-1} \frac{1}{i+j}, \quad (14)$$

where m is a positive integer.

Theorem 3.

$$\sum_{i=1+m}^n \frac{H_i}{i-m} = \frac{1}{2} (H_{n-m}^2 + H_{n-m}^{(2)}) + \sum_{i=1}^{n-m} \frac{1}{i} \sum_{j=1}^m \frac{1}{i+j}, \quad (15)$$

where m is a non-negative integer and $n > m$.

3. Polygamma Functions and Harmonic Numbers

Moenck's procedure is to follow, as closely as possible, the techniques used for the integration of rational functions. Using the method of summation by parts, this algorithm represents the sum of a rational function as a rational part plus a transcendental part. The transcendental part is expressed in terms of polygamma functions. Most of Moenck's procedure can be traced back to Jordan's book on finite differences [5]. In this section, we will explore some of the properties of polygamma functions and derive a relationship between them and the harmonic numbers.

The polygamma functions are the logarithmic derivatives of the gamma function. These functions are denoted by $\psi_m(x)$, where

$$\psi_m(x) = D^m \log \Gamma(x+1), \quad m \geq 1. \quad (16)$$

Since

$$\Gamma(x+1) = \lim_{n \rightarrow \infty} \left[\frac{n! n^{(x+1)}}{(x+1)(x+2) \cdots (x+n+2)} \right],$$

we have

$$\log \Gamma(x+1) = \lim_{n \rightarrow \infty} \left[\log n! + (x+1) \log n - \sum_{k=1}^{n+2} \log(x+k) \right]$$

for all x .

Consider the case when $m = 1$.

$$\begin{aligned} \psi_1(x) &= D \log \Gamma(x+1) \\ &= \lim_{n \rightarrow \infty} \left[\log n - \sum_{k=1}^{n+2} \frac{1}{(x+k)} \right], \\ \psi_1(0) &= \lim_{n \rightarrow \infty} \left[\log n - \sum_{k=1}^{n+2} \frac{1}{k} \right] = -\nu \quad (\nu = \text{Euler's constant}). \end{aligned} \quad (17)$$

Let us apply the difference operator Δ , where $\Delta f(x) = f(x+1) - f(x)$, to $\psi_1(x)$.

$$\begin{aligned} \Delta \psi_1(x) &= D \Delta \log \Gamma(x+1) \\ &= D \log(x+1) \\ &= \frac{1}{x+1}. \end{aligned} \quad (18)$$

Now if we apply the inverse difference operator Δ^{-1} to (18), we have

$$\Delta^{-1} \frac{1}{x+1} = \psi_1(x)$$

Since the inverse difference is the indefinite sum, by evaluating from 0 to $x-1$, we obtain

$$\sum_{i=0}^{x-1} \frac{1}{i+1} = \psi_1(x-1) - \psi_1(0).$$

By (1), the L.H.S. of the above equation is H_x if x is an integer. So

$$\psi_1(n-1) = -\nu + H_n, \quad n \geq 1. \quad (19)$$

Next, consider the case when $m=2$.

$$\begin{aligned} \psi_2(x) &= D \psi_1(x) \\ &= \lim_{n \rightarrow \infty} \left[- \sum_{k=1}^{\infty} \frac{-1}{(x+k)^2} \right] \\ &= \sum_{k=1}^{\infty} \frac{1}{(x+k)^2}, \\ \psi_2(0) &= \sum_{k=1}^{\infty} \frac{1}{k^2} = \zeta(2) \quad (\zeta = \text{Riemann zeta function}). \end{aligned} \quad (20)$$

Again if we apply the difference operator Δ to $\psi_1(x)$, we get

$$\begin{aligned} \Delta \psi_1(x) &= D \left(\frac{1}{x+1} \right) = \frac{-1}{(x+1)^2}, \\ \psi_2(x) &= \Delta^{-1} \frac{-1}{(x+1)^2} \\ \psi_2(n-1) &= - \left(1 + \frac{1}{2^2} + \dots + \frac{1}{n^2} \right) + \psi_2(0) \\ &= \psi_2(0) - H_n^{(2)}. \end{aligned} \quad (21)$$

In general, we have the following theorem.

Theorem 4. For integers $n > 1$ and $m \geq 1$,

$$\psi_m(n-1) = \psi_m(0) + (-1)^{m-1} (m-1)! H_n^{(m)}. \quad (22)$$

It is interesting to note that whenever m is even, $\psi_m(0) = 1/2 | B_r | (2\pi)^r / r!$, where B_r is a Bernoulli number. For example, $\psi_2(0) = \pi^2/6$, $\psi_4(0) = \pi^4/90$, ... There is no known formula for $\psi_m(0)$ when m is an odd integer greater than one.

4. Examples

The results derived above imply that when we want to find the closed form for a summation involving harmonic numbers, we perform the following steps:

- i) Replace the harmonic number by its definition.
- ii) Interchange the order of summation.
- iii) Apply Moenck's algorithm to the inner sum.

- iv) Apply Moenck's algorithm to the resulting sum.
- v) Use the transformation (22) to convert the polygamma functions to harmonic numbers.

We illustrate this procedure with two examples.

Example 1. Find the sum $\sum_{i=1}^n i^4 H_i^{(2)}$.

$$\begin{aligned}\sum_{i=1}^n i^4 H_i^{(2)} &= \sum_{i=1}^n i^4 \sum_{j=1}^i \frac{1}{j^2} \\ &= \sum_{j=1}^n \frac{1}{j^2} \sum_{i=j}^n i^4.\end{aligned}$$

We apply Moenck's algorithm to $\sum_{i=j}^n i^4$ to get

$$\left(\frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n\right) - \left(\frac{1}{5}j^5 - \frac{1}{2}j^4 + \frac{1}{3}j^3 - \frac{1}{30}j\right).$$

Applying Moenck's algorithm again to the product of $1/j^2$ and the above expression, we obtain

$$\begin{aligned}&\left(\frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n\right) [\psi_2(n-1) - \psi_2(0)] \\ &\quad - \frac{1}{20}n^4 + \frac{1}{15}n^3 + \frac{1}{30}n^2 - \frac{1}{12}n + \frac{1}{30} [\psi_1(n-1) - \psi_1(0)].\end{aligned}$$

Using the transformation (22) on this formula, we arrive at the answer,

$$\left(\frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n\right) H_n^{(2)} - \frac{1}{20}n^4 + \frac{1}{15}n^3 + \frac{1}{30}n^2 - \frac{1}{12}n + \frac{1}{30} H_n.$$

Example 2. Find the sum $\sum_{i=1}^n i^2 H_{n+i}$.

$$\begin{aligned}\sum_{i=1}^n i^2 H_{n+i} &= \sum_{i=1}^n i^2 \sum_{j=1}^{n+i} \frac{1}{j} \\ &= \sum_{i=1}^n i^2 \left[H_n + \sum_{j=1}^i \frac{1}{n+j} \right] \\ &= H_n \sum_{i=1}^n i^2 + \sum_{j=1}^n \frac{1}{n+j} \sum_{i=j}^n i^2 \\ &= H_n \left[\frac{n(n+1)(2n+1)}{6} \right] \\ &\quad + \sum_{j=1}^n \frac{1}{n+j} \left[\frac{n(n+1)(2n+1)}{6} - \frac{(j-1)j(2j-1)}{6} \right].\end{aligned}\tag{23}$$

Applying Moenck's algorithm to the second term of (23) yields

$$\begin{aligned}&\frac{n(n+1)(2n+1)}{6} [\psi_1(2n-1) - \psi_1(n-1)] - \left[\frac{5}{18}n^3 + \frac{1}{4}n^2 - \frac{1}{36}n \right. \\ &\quad \left. - \frac{n(n+1)(2n+1)}{6} (\psi_1(2n-1) - \psi_1(n-1)) \right].\end{aligned}$$

Simplifying the above expression gives

$$\frac{n(n+1)(2n+1)}{6} \left[2\psi_1(2n-1) - 2\psi_1(n-1) \right] - \frac{n}{36}(10n^2 + 9n - 1),$$

and using the transformation (22), we get

$$\frac{n(n+1)(2n+1)}{6} \left[2H_{2n} - 2H_n \right] - \frac{n}{36}(10n^2 + 9n - 1).$$

Now combining the first term of (23) and the above expression produces the result,

$$\frac{n(n+1)(2n+1)}{6} \left[2H_{2n} - H_n \right] - \frac{n}{36}(10n^2 + 9n - 1).$$

5. References

- [1] R. W. Gosper, "Indefinite hypergeometric sums in MACSYMA," *Proc. MACYSMA User's Conference* (1977), Berkeley CA, 237-252.
- [2] R. W. Gosper, "Decision procedures for indefinite hypergeometric summation," *Proc. Nat. Acad. Sciences, USA* 75 (1978), 40-42.
- [3] M. B. Hayden, "Automated tools for the analysis of algorithms," Masters Thesis, University of Rhode Island, Kingston RI (1986).
- [4] M. B. Hayden and E. A. Lamagna, "Summation involving binomial coefficients using hypergeometric functions," *Proc. ACM Symposium on Symbolic and Algebraic Computation* (1986), 367-372.
- [5] C. Jordan, *Calculus of Finite Differences*. Chelsea, New York NY (1965).
- [6] M. Karr, "Summation in finite terms," *Journal of the ACM* 28 (1981), 305-350.
- [7] M. Karr, "Theory of summation in finite terms," *Journal of Symbolic Computation* 1 (1985), 303-315.
- [8] D. E. Knuth, *The Art Of Computer Programming, Vol. 1: Fundamental Algorithms* (Second Edition). Addison-Wesley, Reading MA (1973).
- [9] R. L. Graham, D. E. Knuth and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Reading MA (1989).
- [10] S.-M. Liu, "Symbolic expressions for summation of harmonic numbers," Masters Thesis, Brown University, Providence RI (1986).
- [11] R. Moenck, "On computing closed forms for summations," *Proc. MACYSMA User's Conference* (1977), Berkeley CA, 225-236.

Algorithm and Implementation for Computation of Jordan Form over $A[x_1, \dots, x_m]$

Nicholas Strauss
Departemento de Matematica
Pontificia Universidade Catolica
Rio De Janeiro, R.J., Brasil

Abstract.

I outline a sequential algorithm for computation of the Jordan form for matrices in $K = A[x_1, \dots, x_m]$, with A an unique factorization domain with separability. The algorithm has average cost (for K integers) of $O(n^4 L(d)^2)$. I have implemented this algorithm in MACSYMA and it is currently distributed as part of the Climax system.

Introduction.

The Jordan form is the generalization of the Eigen problem with respect to a finite dimensional vector space over an algebraically closed field. Other forms, the Smith or Rational, are useful as canonical representatives with respect to the equivalence induced by similarity of linear transformations. Kannan and Bachem [3] give a sequential algorithm for computing the Smith form for integer nonsingular matrices, of cost $O(n^4 \log^2(n))$, for a matrix of dimension n . The algorithm relies on subcalculation of the Hermite normal form. Recently, Kaltofen, Krishnamoorthy, and Saunders [2] have developed a parallel probabilistic algorithm for computing the Jordan form using a Smith form calculation. Their Smith form algorithm is in *Las Vegas - RNC*².

Here I outline another sequential algorithm for computation of the Jordan form for matrices in $K = A[x_1, \dots, x_m]$, with A either the integers or a finite field, more generally any integral domain with unique factorization and separability. The algorithm has average cost of order $O(n^4 L(d)^2)$ with $L(d)$ the length of the coefficients (assumed integral), with a worst case of $O(n^{12} + n^9 L(d)^3)$. I have implemented this algorithm in MACSYMA, and it is currently distributed as part of the Climax system. [7]

Algorithm.

Let J be a $n \times n$ matrix with coefficients in K .

Form $\text{Determinant}(J - \lambda I) = p(x_1, \dots, x_m, \lambda)$.

Then the irreducible factors of $p(x_1, \dots, x_m, \lambda)$ given by FACTOR are irreducible over $K[\lambda]$. Hence they are either of form $p_i(x_1, \dots, x_m)$ irreducible over K , or of the form $p_i(x_1, \dots, x_m, \lambda)$ irreducible over $K'[\lambda]$, with K' the field of quotients of K .

One can form the algebraic closure $CL(K')$ of the field K' . In this field, the Jordan form exists. [5] Theorems 1 and 2 hold in this generality, thus the algorithm works for matrices with coefficients in K . In practice, this is dependent upon the zero-equivalence problem within RANK.

1. Compute the characteristic polynomial $p(\lambda)$ of J by computation of $\text{DET}(J - \lambda I)$.
2. FACTOR $p(\lambda)$ into powers of irreducible polynomials over $K'[\lambda]$, $p(\lambda) = \prod_{i=1}^k p_i(\lambda)^{r_i}$.
3. Form matrix polynomials $p_i(J)$, $i = 1, \dots, k$.
4. Use Rank Space decomposition on each $p_i(J)$ for $i = 1, \dots, k$ searching from 1 to r_i to find $B_r(p_i(\lambda))$, the number of Jordan r -blocks for irreducible factor p_i .
5. For each eigenvalue $\lambda_{i,l}$ for $l = 1, \dots, r_i$, the number of Jordan r -blocks will be $B_r(\lambda_{i,l}) := B_r(p_i(\lambda))/r_i$.
6. Find eigenvalues, $\lambda_{i,l}$ if possible.
7. Compute similarity transform and (optional) check if Jordan form is correct.

Rank Space Decomposition.

Let λ be an eigenvalue. As is well known, letting $M = J - \lambda I$,

$$\begin{aligned} \delta(0) &= n - \text{rank}(M), \\ \delta(r) &:= \text{rank}(M^r) - \text{rank}(M^{r+1}) \\ &= \text{nullspace}(M^{r+1}) - \text{nullspace}(M^r) \\ &= \sum_{j>r} B_j(\lambda). \end{aligned}$$

with $B_j(\lambda)$ is the number of $j \times j$ Jordan blocks for eigenvalue λ . This formula can be used with a binary search to determine $B_j(\lambda)$.

```

Search( $r_1, r_2$ ) := Midpoint =  $1/2(r_1 + r_2)$ 
    If  $r_1 - r_2 \leq 2$  then Search1( $r_1, r_2$ )
    else
    If  $\delta(r_1) - \delta(\text{Midpoint}) \neq 0$ 
    then Search( $r_1, \text{Midpoint}$ )
    If  $\delta(\text{Midpoint}) - \delta(r_2) \neq 0$ 
    then Search( $\text{Midpoint}, r_2$ )

```

```

Search1( $r_1, r_2$ ) := For  $i = r_1 + 1$  thru  $r_2 + 1$  do
    If  $\delta(i - 1) - \delta(i) \neq 0$ 
    then let  $B_i(\lambda) = \delta(i - 1) - \delta(i)$ 

```

In general, we can perform Rank Space decomposition on irreducible factors of $p(\lambda)$ to find the number of $j \times j$ Jordan blocks for groups of eigenvalues. This allows us to avoid finding the roots and increases the efficiency.

Theorem 1. Suppose $\lambda_{i,1}, \dots, \lambda_{i,r_i}$ are the roots of the irreducible factor $p_i(\lambda)$ of the characteristic polynomial $p(\lambda)$ of the matrix J . Then Rank Space decomposition of $p_i(J)$ will give $\sum_{l=1}^{r_i} B_j(\lambda_{i,l})$.

Proof. $p_i(J) \cong \oplus_{i=1}^k \oplus_{l=1}^{r_i} \oplus_{j=1}^s p_i(b_j(\lambda_{i,l}))$ with $b_j(\lambda_{i,l})$ is Jordan form for eigenvalue $\lambda_{i,l}$, k is the number of irreducible factors of $p(\lambda)$, and s is the largest Jordan block for $\lambda_{i,l}$.

Each $p_i(b_j(\lambda_{i,l}))$ is nilpotent. The t -powers of this matrix have column rank t , since the super-diagonal $(i, i+t)$ for $i = 1, \dots, n$, is $\frac{d(p_i(\lambda_{i,l}))}{d\lambda} \neq 0$ for the characteristic polynomial is separable. $\square$

Theorem 2. If λ and μ are both roots of irreducible factor $r(x)$ of a characteristic polynomial of matrix J , over K , they share the same Jordan structure,

$$B_j(\lambda) = B_j(\mu),$$

for $j = 1, \dots, n$.

Proof. Consider the splitting field L of $r(x)$ as a Galois extension of the field of fractions K' of K . Let $\sigma \in \text{GAL}(L/K')$ be an element of the Galois group of automorphisms of L which fix K' . Then $\sigma(J) = J$. Apply σ to similarity equation,

$$P^{-1}JP = J$$

for $\hat{J}$ the Jordan form, and P has coefficients in L .

$$\sigma(P^{-1})\sigma(\hat{J})\sigma(P) = J.$$

Rearranging, $\sigma(\hat{J})$ is similar to $\hat{J}$. Hence by uniqueness of Jordan form (up to permutation of blocks), we have that λ and μ share the same Jordan structure.

□

Complexity.

The complexity of this procedure is the sum of the computation of characteristic polynomial, factorization, and the Rank Space decomposition for each factor. The algorithm is very efficient as showing diagonalizability, since only two ranks must be computed for each irreducible factor. Total cost = cost(DET) + cost(FACTOR) + cost(Rank Space decomposition). Cost(DET) is cost of one elimination, over $K[\lambda]$. Using Kannan's Hermite form algorithm, gives cost $O(n^4 L(d)^2)$. Cost(FACTOR) is cost of a univariate factorization of characteristic polynomial of degree n (assume K is integer ring). The average cost is $O(n^3 \log(n)^3 \log(p))$ using Rabin's randomized modification of Berlekamp's factoring algorithm. The worst case is $O(n^{12} + n^9 L(d)^3)$ using the Lenstra, Lenstra, and Lovasz factoring algorithm. Finally, the cost of Rank Space decomposition is $O(n^4)$.

Theorem 3. The cost of rank space decomposition is bounded by n^4 for K integers and base field operations of unit cost.

Proof. Matrix multiplication over K has cost $O(n^2)$, hence the sequential cost of matrix powering is $O(n^2 \log(n))$. In the worst case, the whole rank space must be searched giving a cost of $O(n^3 \log(n))$.

The rank space decomposition technique will take logarithmic time to search the rank space, which is r_i . One rank calculation costs $O(n^3)$, for K integer ring. Thus the technique for one factor $p_i(x)$ will cost $n^3 \log(r_i)$.

The complete cost will then be

$$\sum_{i=1}^k n^3 \log(r_i) = n^3 \log\left(\prod_{i=1}^k r_i\right).$$

But $\sum_{i=1}^k r_i = n$. Since $\log$ is an increasing function, it is enough to maximize $\prod_{i=1}^k r_i$ given the constraint $\sum_{i=1}^k r_i = n$. By the geometric-arithmetic mean inequality, $(\prod_{i=1}^k r_i)^{1/k} \leq \frac{1}{k} \sum_{i=1}^k r_i$, or $\log(\prod_{i=1}^k r_i) \leq k \log\left(\frac{n}{k}\right) \leq \frac{n}{e}$. Hence the cost is bounded by,

$$\leq n^3 k \log\left(\frac{n}{k}\right) \leq \frac{n^4}{e}.$$

k is the number of distinct irreducible factors of the characteristic polynomial. $\square$

In the average case, the characteristic polynomial computation dominates the algorithm cost. In the worst case, however, the factoring computation will. This analysis is for integral ground ring, for enlarged rings the computation will become significantly more costly, due to the rank computations needed over the ground ring.

Similarity Transform.

To compute the similarity transform matrix one simply solves the eigen equation, for each height-one eigenvector, then computes the cyclic vectors which span each eigenspace. Again by Galois theory, conjugate eigenvectors correspond with conjugate eigenvalues. Hence one solves the set of linear equations,

$$(J - \lambda I)e_i = 0,$$

over $K[\lambda]/p_i(\lambda)$ for the unknown e_i as $1 \leq i \leq k$. The cyclic vectors (or height- α) are found by computing,

$$e_i^\alpha := (J - \lambda I)^\alpha e_i.$$

The complete set of height- α eigenvectors $e_{i,l}^\alpha$ for $1 \leq i \leq k$ and $1 \leq l \leq \deg(p_i(\lambda))$ is then found by the appropriate substitution for λ . One point is worth mentioning. The eigen equation introduces degrees of freedom for each eigenvector. In the implementation, each degree of freedom is preserved as a new parameter r_i . Hence the similarity transform has entries in $K[\lambda_{i,l}][r_1, r_2, \dots, r_\mu]$, with $1 \leq \mu \leq n\nu$ for ν the number of jordan blocks total.

Conclusion.

Consider the primitive element ω which can generate the algebraic extension, $K(\omega) = K(\lambda_{i,l})$. Then computations involving the similarity matrix can be performed over $K(\omega)(r_1, r_2, \dots, r_\mu)$. However, computations using (or to find) a primitive element are costly. [6] Note that computations with the Jordan matrix $\hat{J} = \bigoplus_{i=1}^k b(p_i(\lambda))$ can be performed for each of the direct summand over $K[\lambda]/p_i(\lambda)$, avoiding the use of a primitive element.

References.

1. Buchberger B., Collins G.E., and Loos R., "Computer Algebra: Symbolic and Algebraic Manipulation", Springer, Wien, 1982.

2. Kaltofen E., Krishnamoorthy M., and Saunders B.D., "Fast Parallel Algorithms for Similarity of Matrices", SYMSAC 1986, Proc. of 1986 Symposium on Symbolic and Alg. Computation, July 21-23, Waterloo, Ontario, B. Char ed., 1986, ACM.
3. *ibid.*, "Fast Parallel Computation of Hermite and Smith Forms of Polynomial Matrices", SIAM J.Alg.Disc.Math., vol. 8, no. 4, October 1987, pp. 683-690.
4. Kannan R., Bachem A., "Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix", SIAM J. Computing, vol. 8, no. 4, November 1979, pp. 499-507.
5. Lang S., "Algebra", Addison Wesley, Reading, Massachusetts, 1974.
6. Najid-Zejli H., "Computations in Radical Extensions", in Proc. Eurosam 84:Springer Lecture Notes in Computer Science 174, Springer-Verlag, Berlin, 1984, pp. 115-122.
7. Strauss N., "Jordan Form and Eigen Finite Field", The Macsyma Newsletter, Symbolics Inc., Cambridge, Massachusetts, October, 1985.
8. *ibid.*, "Jordan Form of a Binomial Coefficient Matrix over $\mathbb{Z}_p$ ", Linear Algebra and Its Applications, 90:65-72, no. 7, Elsevier.

Fast Group Membership Using a Strong Generating Test for Permutation Groups

Gene Cooperman and Larry Finkelstein*

College of Computer Science
Northeastern University
360 Huntington Ave.
Boston, Mass. 02115

Paul Walton Purdom, Jr.

Department of Computer Science
Indiana University
101 Lindley Hall
Bloomington, In. 47505

Abstract. Many important algorithms in computations with permutation groups require an efficient solution to the *group membership problem*. This requires deciding if a given permutation is an element of a permutation group G specified by a set of generators. Sims [8] developed an elegant solution to this problem. His method relies on the construction of an alternative generating set for G known as a *strong generating set* which can be easily used to test membership of an arbitrary permutation in G . This algorithm was shown to have worst case time $O(n^6)$. Later versions [1, 5, 6] have improved the theoretical worst case time but without necessarily improving the performance in practice. An algorithm is presented here which has an observed running time of $O(n^4)$ for all permutations groups for which it has been tested. (The worst-case time for this algorithm is $O(n^5)$.) The key idea is a new test [3] for whether a set of generators is a strong generating set. Each call to this last test has worst case time $O(n^4)$. A further reduction in time is achieved by using a fast algorithm for finding reduced generating sets. For groups with small bases, the running time is $O(n^2)$, which is optimal for the data structure used.

1. Introduction

Given a permutation group G acting on n letters defined by a generating set S , the *group membership problem* is to decide if an arbitrary permutation is an element of G . Efficient algorithms for solving the membership problem are crucial for many important group computations. Sims [8] gave the first polynomial time algorithm for solving this problem. His algorithm has worst-case time $O(n^6)$, but it is quite practical when a small *base* is present. Since then Knuth [6] and Jerrum [5] have developed practical algorithms with worst-case times of $O(n^5)$. The algorithm of Babai, Luks and Seress [1] has worst-case time $O(n^4 \log^c(n))$ with a large constant coefficient. Their algorithm introduces several ideas of theoretical interest. A variation on Sims' original method, based on coset enumeration, is presented in [7].

Most algorithmic solutions to the membership problem identify a collection of subsets $\{U_1, \dots, U_k\}$ of G and then attempt to express g in the factored form $g = g_k g_{k-1} \dots g_1$, where each $g_i \in U_i$, $1 \leq i \leq k$. Each U_i arises as a set of (right) cosets of $G^{(i+1)}$ in $G^{(i)}$, where $G = G^{(1)} \supseteq G^{(2)} \supseteq \dots \supseteq G^{(k+1)} = \{e\}$ is a chain of subgroups of G . If the attempt to express g in this form fails, then g cannot be a member of G . Generally speaking, if U_i is a set of distinct coset representatives for $G^{(i+1)}$ in $G^{(i)}$, then we call $U = \{U_1, U_2, \dots, U_{n-1}\}$ a *partial family of cosets*. U is said to be *complete* if each U_i is in fact a full set of coset representatives for $G^{(i+1)}$ in $G^{(i)}$. A set S of generators for G is said to be a *strong generating set* if $G^{(i)} = \langle S \cap G^{(i)} \rangle$ for all i . If S is a strong generating set, then it is easy to construct a complete family of cosets.

The traditional choice for the subgroups $G^{(i)}$ is the *point stabilizer sequence*. Given a fixed ordering $\alpha = \alpha_1, \dots, \alpha_n$ of $\{1, 2, \dots, n\}$, the i^{th} element of this sequence is given by $G^{(i)} = G_{\{\alpha_1, \dots, \alpha_{i-1}\}}$, the subgroup consisting of all elements of G which fix each of the points $\alpha_1, \alpha_2, \dots, \alpha_i$, $1 \leq i \leq n-1$. This paper uses this sequence.

* The work of this author was supported in part by the National Science Foundation under grant number DCR-8603293.

A set of m points such that only the identity element of G fixes each point is called a *base*. All permutation groups have a base of size at most $n - 1$, and many have a very small base. Most permutation group algorithms, including ours, find a small base if one exists. The base we find is at most $\log_2(n)$ times the size of the smallest base. We analyze running times in terms of n , the number of points, and m , the size of the base found by the algorithm.

In [3], we introduced the notion of a *basic generator*. Basic generators are formed from the generating set S for G and elements of the partial family of cosets. Subject to certain conditions, we proved that the partial family of cosets is complete if and only if each basic generator factors through the family. This is the heart of our strong generating test. This test can be efficiently implemented using the labelled branching data structure of Jerrum [5]. The first step of our test is to build a labelled branching which contains all the orbit information for the point stabilizer sequence implicit in S . This can be accomplished in time $O(n|S| + n^2)$, by using a fast algorithm described in [3] and generalized in section 4. The strong generating test may then be applied with respect to this branching and S in time $O(mn^2|S|)$ where m is the number of internal nodes of the branching. If a small base is known in advance, then this time can be reduced to $O(m^2n|S| + n^2)$. A specific idea for incorporating this test using the Schreier vector data structure of Sims is described in [3].

In our initial implementation of an algorithm for constructing a strong generating set using the labelled branching data structure, it was experimentally observed that the strong generating set is usually present early in the computation. The running time of the algorithm is then dominated by the time to verify the strong generating property for this existing set of generators. An efficient strong generating test allows early termination. This is similar in spirit to the approach taken by Leon [7] in his Random Schreier-Sims algorithm and suggests that the strong generating test could be of great value in many traditional implementations of group membership algorithms.

There are two advantages to the labelled branching data structure. First, the labelled branching data structure tends to "fill up" very rapidly, i.e., the partial family of cosets it defines tends to be complete shortly after the computation has begun. Second, there is little extra overhead in applying the strong generating test, since the partial family of cosets will automatically satisfy a necessary condition for use of the strong generating test. Our algorithm for constructing a strong generating set runs in time $O(m^2n^2)$ in practice. The worst-case time is $O(m^3n^2)$. When $m = O(n)$, the worst-case time is the same as previous practical algorithms, but previous algorithms often have an average time close to the worst-case time.

2. Labelled Branchings

Let G be a permutation group acting on $\Omega = \{1, 2, \dots, n\}$ and let $\alpha = \alpha_1, \alpha_2, \dots, \alpha_n$ be an ordering of the points of Ω . Define the point stabilizer chain of subgroups for G (with respect to α) by setting $G^{(i)} = G_{(\alpha_1, \dots, \alpha_{i-1})}$ (the subgroup of G which fixes $\alpha_1, \dots, \alpha_{i-1}$ pointwise), $i = 2, \dots, n - 1$, and $G^{(1)} = G$. A *branching* on Ω relative to α is a directed forest with nodes $\alpha_1, \dots, \alpha_n$, in which each edge has the form (α_i, α_j) for $i < j$. A branching $\mathcal{B}$ is said to be a *labelled branching* for G relative to α , if each edge (α_i, α_j) is labelled by a permutation σ_{ij} so that the following properties hold:

- (i) $\sigma_{ij} \in G^{(i)}$ and moves α_i to α_j .
- (ii) The set of edge labels of $\mathcal{B}$ generates G .

A labelled branching $\mathcal{B}$ is said to be *complete* if the following additional property holds:

- (iii) If α_k is in the $G^{(i)}$ orbit of α_i , then there is a (directed) path in $\mathcal{B}$ from α_i to α_k .

Property (iii) ensures that the edge labels of $\mathcal{B}$ form a strong generating set for G relative to the ordering α . For simplicity of notation, we will assume from now on that α is the identity ordering.

A labelled branching can be implemented as an array of structures using $O(n^2)$ storage. Node labels are used to implicitly store the edge labels, and an edge label may be recovered at the cost of one permutation multiply. Let r be the root of the connected component of $\mathcal{B}$ containing j . Associate with each node a *node label* $\tau(j)$, where $\tau(j)$ is the product of the path labels from the root r to j if $r \neq j$, and is the identity if $r = j$. Since $\tau(j)^{-1}$ moves j to its root r , we may recover the label for edge (i, j) as $\sigma_{ij} = \tau(i)^{-1}\tau(j)$. Furthermore, if there is a path from k to j in $\mathcal{B}$, then $\tau(k)^{-1}\tau(j)$ is the product of the edge labels along the path from k to j . This means that coset representatives can also be recovered at the cost of one multiply.

Let $\mathcal{E}(\mathcal{B})$ denote the set of edge labels of $\mathcal{B}$.

Proposition 2.1. *If $\mathcal{B}$ is a complete labelled branching for G , then $\mathcal{E}(\mathcal{B})$ is a strong generating set for G .*

A labelled branching, $\mathcal{B}'$, dominates $\mathcal{B}$ if $G' = \langle \mathcal{E}(\mathcal{B}') \rangle$, $G = \langle \mathcal{E}(\mathcal{B}) \rangle$ and $\langle G'^{(i)} \cap \mathcal{E}(\mathcal{B}') \rangle \supseteq \langle G^{(i)} \cap \mathcal{E}(\mathcal{B}) \rangle$ for $1 \leq i \leq n-1$. As a simple consequence of Proposition 2.1, a complete labelled branching for G dominates all other labelled branchings for G .

Given a labelled branching $\mathcal{B}$ for G , if i is connected to j in $\mathcal{B}$, $i < j$, then we denote the product of the edge labels along the (unique) path from i to j by p_{ij} . As noted before, $p_{ij} = \tau(i)^{-1}\tau(j)$.

Given $g \in \text{Sym}(\Omega)$, we say that g *factors through* $\mathcal{B}$ if g can be written

$$g = p_{i_k j_k} \cdots p_{i_2 j_2} p_{i_1 j_1},$$

where $i_k > \cdots > i_2 > i_1$.

Procedure Factor. *Input:* an element $g \in S_n$ and a labelled branching $\mathcal{B}$ for G . *Output:* true if g factors through $\mathcal{B}$, and false otherwise.

Initialize h to g

For $i \leftarrow 1$ to $n-1$ do if $i^h \neq i$ then

 If there exists a path in $\mathcal{B}$ from i to i^h then

 Set $u \leftarrow \tau(i)^{-1}\tau(i^h)$; (u maps i to i^h)

 Set $h \leftarrow hu^{-1}$

 Else return(FALSE)

Return(TRUE)

Note that Factor may return false even if $g \in G$, in the case where $\mathcal{B}$ is not complete. However, if $\mathcal{B}$ is complete, then Factor returns true if and only if $g \in G$.

If $\mathcal{B}$ is a complete labelled branching for G , then the set of internal nodes of $\mathcal{B}$ forms a *base* for G . A base for G is a set of points with the property that only the identity element of G fixes each point in the set. If there are m internal nodes of $\mathcal{B}$, then the Factor algorithm takes time $O(mn)$.

The main operation on a labelled branching $\mathcal{B}$ for G is "sifting" a permutation g as described by Jerrum [5]. Sifting is closely tied to factoring in the sense that Sift does not transform $\mathcal{B}$ if Factor returns true. Sift returns true if and only if Factor returns true. Otherwise, Sift returns false and we have the following situation. There exists an index i_k such that $g' = gp_{i_1 j_1}^{-1} \cdots p_{i_2 j_2}^{-1} p_{i_{k-1} j_{k-1}}^{-1}$, where $i_k > i_{k-1} > \cdots > i_2 > i_1$, g' fixes $1, \dots, i_{k-1}$ and there is no path in $\mathcal{B}$ from i_k to $j_k = i_k^{g'}$. Sift then attempts to create a new edge (i_k, j_k) with edge label g' . If $\text{indegree}(j_k) = 0$, then this can be done directly. However, if $\text{indegree}(j_k) \neq 1$, then care must be taken not to destroy the properties of the labelled branching. This is accomplished by adding a new edge with label g' , and then removing a newly redundant edge. A new element derived from g' and the redundant edge label is recursively sifted into the new labelled branching. The recursive Sift is the key to maintaining the properties of the labelled branching.

The next two propositions describe the relationship between a branching $\mathcal{B}$ and the branching $\mathcal{B}'$ obtained by sifting an element g into $\mathcal{B}$. (The proof is omitted.)

Proposition 2.2. *If $\mathcal{B}'$ is the labelled branching resulting from sifting g into $\mathcal{B}$, then $\mathcal{B}'$ dominates $\mathcal{B}$.*

Proposition 2.3. *If $\mathcal{B}'$ is the labelled branching resulting from sifting g into $\mathcal{B}$ and there is a path in $\mathcal{B}$ from p to q , then there is a path in $\mathcal{B}'$ from p to q .*

3. Basic Generators

The following discussion of basic generators is specialized for the labelled branching data structure. It can be made independent of the particular data structure used to hold information about the group [3].

Let S be a set of generators for G with $e \notin S$ and let $S^{(i)} = S \cap G^{(i)}$, $1 \leq i \leq n-1$. We say that the branching $\mathcal{B}$ is *fully augmented* for S , if for each $j \in i^{(S^{(i)})}$, $1 \leq i \leq n-1$, there is a path in $\mathcal{B}$ from node i to node j with the associated path product an element of $\langle S^{(i)} \rangle$ and each path product in $\mathcal{B}$ arises precisely in this way. In this case, $\langle \mathcal{E}(\mathcal{B}) \rangle \subseteq \langle S \rangle$. The inclusion is not always strict. If S is a strong generating set for G , then clearly a labelled branching fully augmented for S is complete for G . In section 4, we will show how to construct a branching which is fully augmented for S in time $O(|S|n + n^2)$.

For each pair of points (i, j) with $i < j$, i connected to j in $\mathcal{B}$, let p_{ij} be the product of the edge labels along the path from i to j . If this path has length 1, then p_{ij} is just an edge label σ_{ij} and we identify it in this way. Let U be the set of all such products, which we call *path products*. Define the set $T = T_1 \cup T_2$ of *basic generators* of G relative to S and U as follows:

$$\begin{aligned} (T_1) \quad & \sigma_{ij}g, \quad g \in G^{(j+1)}, \text{ or} \\ (T_2) \quad & p_{ij}gkl, \quad \text{where an edge label } \sigma_{ii_1} \text{ exists with } i < i_1, \text{ either } i_1 = j \\ & \text{or } i_1 \text{ is connected to } j \text{ in } \mathcal{B}, \text{ and } g \in G^{(i)} - G^{(i_1)} \end{aligned}$$

where $g \in S$ and $p_{ij}, \sigma_{ij} \in U$.

In [3] it is shown that $|T| \leq (n-1)|S|$ and that the following strong generating test for S takes time $O(|S|mn + \min(|S|m^2n, m^2n^2))$.

Strong Generating Test.

- (1) Create a labelled branching $\mathcal{B}$ fully augmented for S . (We will give a fast algorithm for this in the next section).
- (2) Factor each element of S through $\mathcal{B}$. If any element fails to factor, then the test fails. (This guarantees that $\mathcal{B}$ is a labelled branching for G .)
- (3) If $|S| > n-1$ then replace S by the edge labels of $\mathcal{B}$.
- (4) Factor each basic generator through $\mathcal{B}$. If every basic generator factors then the test succeeds. Otherwise, the test fails.

4. Construction of a Fully Augmented Branching and Reduced Generating Set

Let S be a generating set for a permutation group. This section describes a procedure for constructing (in $O(|S|m + mn)$ time and $O(|S|m + mn)$ space) a subset S' of S and a labelled branching $\mathcal{B}$ fully augmented for S' which is also fully augmented for S . S' is first constructed independently of $\mathcal{B}$. Thus, if S is a strong generating set, then the routine can be used to construct a reduced strong generating set.

We will describe the procedure and omit the proof that the algorithm is correct and the time and space bounds are as stated. We use two intermediate data structures. $\mathcal{I}$ is an acyclic (undirected) graph with each edge $\{i, j\}$ labelled by an element $\sigma_{ij} \in S$ which moves i to j and $\sigma_{ji} = \sigma_{ij}^{-1}$. If $k = \min(i, j)$, then we do not require that $\sigma_{ij} \in \langle S^{(k)} \rangle$. The set S' will consist of the edge labels of $\mathcal{I}$. *root* is an array of size n giving for each $i \in \Omega$ the smallest node, *root*[i]

of $\mathcal{I}$, such that i and $\text{root}[i]$ are in the same connected component. The procedure proceeds in a bottom up fashion decrementing i from $n - 1$ downto 1. After the i^{th} iteration, the connected components of $\mathcal{I}$ represent the orbits of $\langle S^{(i)} \rangle$, $\text{root}[j]$ is the smallest node in the $\langle S^{(i)} \rangle$ orbit containing j , and the product of the edge labels along the path from $\text{root}[j]$ to j represents an element of $\langle S^{(i)} \rangle$ which moves $\text{root}[j]$ to j . In particular, since $\text{root}[i] = i$, all the descendants of i in $\mathcal{B}$ are known. Thus we are able to set the parent field of $\mathcal{B}$, for all nodes $j > i$ with $\text{parent}(j) \geq i$. After these data structures have been built, we can use $\mathcal{I}$ and the parent field information already stored in $\mathcal{B}$ to compute the τ fields of $\mathcal{B}$.

Procedure Augment. *Input:* A set S of generators for the permutation group G . *Output:* A subset S' of S and a labelled branching $\mathcal{B}$ with the property that $\mathcal{B}$ is fully augmented for both S and S' . *Intermediate Data Structures:* $\mathcal{I}$ and root as described above.

[Build $\mathcal{I}$, root and the parent field of $\mathcal{B}$.]

```

Initialize  $\mathcal{B}$  to a trivial labelled branching on  $\Omega$  ( $\forall i, \text{parent}(i) \leftarrow \text{NIL}$ )
Initialize  $\mathcal{I}$  to a trivial labelled graph on  $\Omega$ 
Set  $S' \leftarrow \emptyset$  and  $\text{root}[n] \leftarrow n$ 
For  $i \leftarrow n - 1$  downto 1 do
  Set  $T \leftarrow S^{(i)} - S^{(i+1)}$ 
  If  $T \neq \emptyset$  then
    For  $j \leftarrow i$  to  $n$  do set  $\text{closed}[j] \leftarrow \text{NIL}$ 
    Set  $\text{root}[i] \leftarrow i$ 
    For  $i' \leftarrow i$  to  $n - 1$  such that  $\text{root}[i'] = i'$  do
      [Use breadth first search to merge the connected components of  $\mathcal{I}$  under  $T$ .]
      Set  $\text{open\_set} \leftarrow \{i'\}$  and  $\text{closed}[i'] \leftarrow \text{TRUE}$ 
      For  $j \in \text{open\_set}$  do
        Remove  $j$  from  $\text{open\_set}$ 
        [Check the image under  $T$ .]
        For  $\rho \in T$  do
          Set  $k \leftarrow j^\rho$ 
          If  $\text{root}[k] \neq i'$ 
            [A new generator is found.]
            Add  $\rho$  to  $S'$ ; delete  $\rho$  from  $T$ 
            [Update  $\text{root}$  to be compatible with  $\rho$ .]
            For  $\ell \leftarrow i'$  to  $n$  do
              If  $\text{root}[\ell] \neq \text{root}[\ell^\rho]$  then [true at least for  $\ell = k$ ]
                [Merge two orbits of  $\mathcal{I}$ .]
                Let  $m = \min(\text{root}[\ell], \text{root}[\ell^\rho])$ ,
                 $M = \max(\text{root}[\ell], \text{root}[\ell^\rho])$  and
                 $\mathcal{M}$  be the set of nodes in  $\mathcal{I}$  connected to  $M$ 
                Foreach  $m' \in \mathcal{M}$  do
                  Set  $\text{root}[m'] \leftarrow m$ 
                  If  $\text{parent}(m') = \text{NIL}$  and  $m' = i$  then
                    Set  $\text{parent}(m') \leftarrow i$  in  $\mathcal{B}$ 
                Add edge  $\{\ell, \ell^\rho\}$  to  $\mathcal{I}$  with labels  $\sigma_{\ell, \ell^\rho} = \rho$ ,  $\sigma_{\ell^\rho, \ell} = \rho^{-1}$ 
            [Update  $\text{open\_set}$ .]
          Foreach  $j' \in \mathcal{I}$  adjacent to  $j$  do
            If  $\text{closed}[j'] = \text{NIL}$  then
              Set  $\text{closed}[j'] \leftarrow \text{TRUE}$  and add  $j'$  to  $\text{open\_set}$ 

```

The three promised structures have now been created. It remains to assign $\tau(j)$ for each node j of $\mathcal{B}$. This is done by the following code fragment. Note that we require knowledge of all descendants of i for i a root of $\mathcal{B}$. This information is stored in an array *descendants*.

[Set the τ fields of $\mathcal{B}$.]

```

For  $j \leftarrow 1$  to  $n$  do
  If  $\text{parent}[j] = \text{NIL}$  then set  $\tau(j) \leftarrow \text{identity}$  else set  $\tau(j) \leftarrow \text{NIL}$ 
For  $i \leftarrow n$  downto 1 and  $\text{parent}[i] = \text{NIL}$  do
  [Compute descendants for  $i$ .]
  Set  $\text{descendants}[i] \leftarrow \text{TRUE}$ 

```

```

For  $j \leftarrow i + 1$  to  $n$  do
  Set  $\text{descendants}[j] \leftarrow \text{NIL}$ 
For  $j \leftarrow i + 1$  to  $n$  do
  If  $\text{descendants}[\text{parent}[j]] = \text{TRUE}$  then set  $\text{descendants}[j] \leftarrow \text{TRUE}$ 
[Use breadth first search in  $\mathcal{T}$  to compute  $\tau(j)$ .]
Set  $\text{open\_set} \leftarrow \{i\}$ 
For  $j \in \text{open\_set}$  do
  Remove  $j$  from  $\text{open\_set}$ 
  Let  $\mathcal{J}$  be the set of nodes adjacent to  $j$  in  $\mathcal{T}$ 
  Foreach  $k \in \mathcal{J}$  such that  $\text{descendants}[k] = \text{TRUE}$  do
    If  $\tau(k) = \text{NIL}$  then set  $\tau(k) \leftarrow \tau(j)\sigma_{jk}$  and add  $k$  to  $\text{open\_set}$ 

```

5. Two Algorithms for Constructing Strong Generating Sets

A crucial step for group membership is obtaining generators for $G^{(i+1)}$ from generators for $G^{(i)}$. The first step is to construct a set of coset representatives for $G^{(i+1)}$ in $G^{(i)}$. This set is in one-one correspondence with the points in the orbit $i^{G^{(i)}}$. Both can be found by a simple transitive closure argument. Generators for $G^{(i+1)}$ may then be constructed by an application of Schreier's Lemma [4, Lemma 7.2.2]. This may be stated in more generality as follows.

Proposition 5.1. (Schreier's Lemma) *Let $G = \langle S \rangle$, and U be a set of coset representatives for H in G . (We assume that $e \in U$.) For each $g \in G$, let $\bar{g}$ be the (unique) element of U such that $Hg = H\bar{g}$. Set $T = \{(hg)(\bar{h}g)^{-1} : h \in U, g \in S\}$. Then T generates H .*

The machinery necessary for the first algorithm has now been developed. This algorithm is a generalization of the one first presented by Jerrum [5]. The generalization has the advantage that it may complete early. Further, the subprocedure Augment is new, and can accelerate the convergence to a complete labelled branching. Early completion is a novel feature, which will be exploited only in the more sophisticated algorithm to follow.

Completion Algorithm (simple version). *Input:* a list S of permutations from S_n , with $G = \langle S \rangle$. *Output:* a complete labelled branching $\mathcal{B}$ for G . *Auxiliary Functions:* $\text{Pt-stabilizer}(\mathcal{B}, i)$ returns a list of edge labels of $\mathcal{B}$ which fix $1, \dots, i - 1$. These are the edge labels of $\mathcal{B}$ of the form σ_{jk} with $\text{parent}(j) \geq i$. $\text{Augment}(S)$ assumes that $\langle S \rangle = G^{(i)}$, and sifts into $\mathcal{B}$ those elements of S which move i to $j \in i^{G^{(i)}}$ and for which i and j are not yet connected in $\mathcal{B}$.

[Initialize.]

Initialize a branching $\mathcal{B}$ to be the trivial branching

Sift each generator of G onto $\mathcal{B}$ [This guarantees that $\mathcal{E}(\mathcal{B})$ generates G .]

[Main loop.]

For $i \leftarrow 1$ to $n - 1$ do

If i is not a leaf then

[Augment $\mathcal{B}$.]

Set $S \leftarrow \text{Pt-stabilizer}(\mathcal{B}, i)$

If $S = \emptyset$ then return($\mathcal{B}$) [the remaining vertices are leaves]

Augment(S) [modified $\mathcal{B}$ dominates old $\mathcal{B}$]

[Sift generators for $G^{(i+1)}$.]

Set $U \leftarrow$ the set of path products in $\mathcal{B}$ starting at i

Sift into $\mathcal{B}$ all the Schreier generators for $G^{(i+1)}$ using S and U

[At this point, $\mathcal{E}(\mathcal{B}) \cap G^{(i+1)}$ generates $G^{(i+1)}$, see section 2.]

Coming into the main loop, we know that the edge labels of $\mathcal{B}$ which fix $1, \dots, i - 1$ generate $G^{(i)}$. In order for $\mathcal{B}$ to be a complete labelled branching, we require that if $j = i^\sigma$ for some $\sigma \in G^{(i)}$ then there exists a path in $\mathcal{B}$ from i to j . This is accomplished by Augment. Propositions 2.2 and 2.3 then guarantee that the final branching will retain a path from i to j .

The second version of the completion algorithm incorporates the strong generating test and reduced generating sets. A key procedure is Reduce-Gen. The input to Reduce-Gen is the branching $\mathcal{B}$ and the current level i . Let S be the edge labels of $\mathcal{B}$ which fix $1, \dots, i - 1$ (and

which generate $G^{(i)}$). Reduce-Gen calls $\text{augment}(S)$ which returns a subset S' of S and a labelled branching B' which is fully augmented for S' and also for S . Ideally, we would like B' to be a complete labelled branching for $G^{(i)}$. This can be verified by using the strong generating test with respect to S' and B' . However, before this test is applied we perform two preliminary steps. We use the notation σ'_j and $\text{parent}'(i)$ to distinguish edge labels and parent fields in B' from those in B .

The first step is to update B with the orbit information contained in B' . This is accomplished by checking in B' for each non-root node k , whether $\text{parent}'(k) = \text{parent}(k)$. If not, then we set $j = \text{parent}'(k)$ and sift σ'_j into B . After this step is accomplished we double check that for each non-root node k , $\text{parent}'(k) = \text{parent}(k)$. If there is a node $k > i$ with the property that $k > \text{parent}(k) > \text{parent}'(k)$, then B' cannot be complete and so we call Augment again and obtain a new S' and B' . We eventually complete this phase and move on to the next step.

We now want to ensure that the edge labels of B' (and hence S') generates $G^{(i)}$. Thus we attempt to factor each element of S through B' . If each element factors through then we are ready to begin the completion test. Otherwise, we take the residue σ' of a $\sigma \in S$ which didn't factor and sift σ' into B and repeat starting with the first step above. By construction, for $\sigma' \in G^{(i)} - G^{(i+1)}$, $(i, i\sigma')$ was not an edge of B' . Since B' and B have the same graph structure, $(i, i\sigma')$ is not an edge of B , and sifting σ' must yield a new labelled branching which strictly dominates the old B .

At this point, we can apply the completion test directly to S' and B' . However, the following alternative appears to be more efficient. Copy B onto B' . Since each edge label of B factors through the old B' , we know that the new B' and old S' satisfy the properties for the completion test. Now first sift into B the basic generators of (B', S') (for level i), which are also Schreier generators. If they all sift into B (equivalently factor through B), then finish the completion test by attempting to sift the remaining basic generators of (B', S') through B . If any fails, i.e. Sift returns false, then sift into B , the remaining Schreier generators for $G^{(i+1)}$ formed by using (B', S') . Note that we choose to call Sift rather than Factor, since if Sift fails, then it performs positive work towards completing the branching.

The motivation for the above algorithm is to delay as long as possible the decision whether to sift Schreier generators or basic generators for the completion test. In empirical studies, the first time the algorithm commits itself to applying the full completion test, it almost always succeeds.

Completion Algorithm (sophisticated version). *Input:* A positive integer n and a set S of generators for a subgroup G of S_n . *Output:* A complete labelled branching B for G . *Auxiliary Functions:* *Sift* and *Reduce-Gen* (which itself uses *Augment*).

[Initialization]

Let B be an empty branching of degree n

Sift each element of S into B

[Main loop.]

For $i \leftarrow 1$ to n do

Set $(S', B') \leftarrow \text{Reduce-Gen}(B, i)$

Copy B onto B'

Let $\text{Schreier}(i)$ be the Schreier generators for $G^{(i+1)}$ formed from B', S'

Let $\text{Basic}(i)$ be the basic generators formed from B', S'

Set $\text{continue-test} \leftarrow \text{TRUE}$

[Sift all Basic generators which are also Schreier generators into B .]

Foreach $\sigma \in \text{Schreier}(i) \cap \text{Basic}(i)$

If not $\text{Sift}(\sigma, B)$ then set $\text{continue-test} \leftarrow \text{FALSE}$

If continue-test then

Foreach $\sigma \in \text{Basic}(i) - \text{Schreier}(i)$

If not $\text{Sift}(\sigma, B)$ then goto("Test Failed")

Return(DONE) ["Test Succeeded"]

"Test Failed": [Continue sifting in Schreier generators.]

Foreach $\sigma \in \text{Schreier}(i) - \text{Basic}(i)$ do $\text{Sift}(\sigma, B)$

Clearly, one does not have to explicitly construct the sets $Schreier(i)$ and $Basic(i)$. Finding the elements in the three sets $Basic(i) \cap Schreier(i)$, $Basic(i) - Schreier(i)$, and $Schreier(i) - Basic(i)$ can be done on the fly. In the section on measurements, the loops over $Basic(i) \cap Schreier(i)$ is called the common part, the loop over $Basic(i) - Schreier(i)$ is called the completion part and the loop over $Schreier(i) - Basic(i)$ is called the Schreier part.

Procedure Reduce-Gen. *Input:* A labelled branching B and the current level i . *Output:* A reduced set of generators S' which generate $G^{(i)}$ (computed from the edge labels of B which fix $1, \dots, i-1$) and a labelled branching B' which is fully augmented for S' . *Side Effect:* B is modified so that the path information for paths starting at a node $j \geq i$ coincides with that for B' . *Auxiliary Functions:* *Test-Orbits* has input parameters B, B' and returns true if $parent(k) = parent'(k)$ for all non-root nodes k of B' , and false otherwise. $parent(k)$ refers to B and $parent'(k)$ refers to B' . *Test-Generators* has input parameters S, B, B' and returns true if each generator factors through B' . Otherwise, *Test-Generators* returns false and has the following side effect. If $\sigma \in S$ fails to factor through B' and $\sigma = \sigma' \rho'_{i_r, j_r} \dots \rho'_{i_1, j_1}$ where $i_r > \dots > i_1 \geq i$ and $\sigma' \in G^{(i-1)}$ fails to factor through B' , then we sift σ' into B . (This adds a new connection in B .) *Augment* and *Sift* are also used.

```

Repeat
  Repeat
    Let  $S$  be the edge labels of  $B$  which fix  $1, \dots, i-1$ 
    Let  $(S', B') \leftarrow \text{Augment}(S)$ 
    [ $B'$  is fully augmented for  $S'$  and also for  $S$ .]
    [Update  $B$ .]
    For  $k \leftarrow i+1$  to  $n$  and  $k$  a non-root node of  $B'$  do
      If  $parent(k) < parent'(k)$  then
        Let  $j = parent'(k)$  and let  $\rho$  be the edge label for  $(j, k)$  in  $B'$ 
        Sift  $\rho$  into  $B$ 
    Until  $\text{Test-Orbits}(B, B')$ 
  Until  $\text{Test-Generators}(S, B, B')$ 
Return( $S', B'$ )

```

6. Measurements

The running times for both the simple and the more sophisticated algorithm were measured for a wide selection of permutation groups. The programs were tested on 187 sets of generators, which generated about 178 distinct permutation groups with 125 distinct orders. In what follows it is convenient to refer to the input as a set of groups, even though strictly speaking it is a set of generator sets. The complete set of groups and generators and the complete set of running times are available from the authors.

The measured running times were analyzed using three parameters: (1) n , the number of points, (2) g , the number of coset representatives, and (3) m , the size of the base. The value of g is n minus the number of orbits in $G^{(1)}$. For transitive groups this is $n-1$, but our set contains a few groups where g is much smaller than n . The value of m is between 1 and $n-1$. Our set of groups covers a wide range of values of m .

The running time of the completion algorithms depend on five factors: (1) the time to multiply a permutation, $O(n)$, (2) the number of multiplications needed to factor a permutation, m or less, (3) the number of coset representatives, g (on the first level, less on higher levels), (4) the number of generators for $G^{(i)}$, $n-i$ or less on level i , and (5) the number of levels, m or less. The product of these five factors yields a worst-case running time of $O(n^2 g m^2)$ ($O(n^3 m^2)$ after applying the inequality $g < n$). The completion test speeds up the simple algorithm by reducing the fifth factor, m . So far as is known, the completion test does not improve the theoretical worst-case time, but in nearly all cases that were measured, the completion test reduced the number of levels to 2 or less. The reduced generating set speeds up the simple algorithm by

reducing the fourth factor. It results in no more than $\min\{m \log_2(n-i), n-i\}$ for level i , yielding a worst-case running time of $O(n g m^3 \log_2(n))$ (or $O(n^2 m^3 \log_2(n))$ using $g < n$), which is an improvement for $m < n/\log_2(n)$. For most of the multiplications in the completion algorithms, the time to multiply permutations can be reduced to $O(m)$ if a base is known in advance, but finding a base appears to be as difficult as the rest of the problem.

Groups are usually specified with a small set of generators (never more than 6 in our set of groups). After the coset representatives are added, the generating set is often of size $n-1$. The first level of the completion algorithm runs much more rapidly if a small generating set is used the first level. For some groups, most of the time is spent on the first level.

Table 1 shows the distribution of number of levels of Schreier generation used by the completion algorithm with the strong generating test. Completing with zero levels indicates that no Schreier generators (other than those that were part of the completion test) were sifted, completing with one level indicates that one set of Schreier generators were sifted, etc. The group that required 6 levels to complete was specially designed to require a large number of levels. (This group had one generator with 7 disjoint cycles, and for 6 of the cycles the cycle length was relatively prime to the product of the previous cycle lengths.)

level	0	1	2	3	4	5	6	Total
cases	76	103	5	1	1	0	1	187

Table 1. Distribution of Schreier levels.

These results show that the branching is usually complete after a small constant number of levels. This early completion is not caused by groups having a small base because 106 of the groups had a base of 4 or more points, and 61 of them had a base of 10 or more points.

For each group, the common part was done at most one more time than the Schreier part. The completion part was usually done once. Table 2 gives the distribution for the number of times the completion part was done.

calls	0	1	2	3	Total
cases	0	184	3	0	187

Table 2. Invocations of completion test.

The running times were measured for Pascal programs compiled by the Berkeley Pascal compiler with optimization turned on, and running on a VAX8800. Table 1 shows that completion algorithm with the strong generating test requires only a fixed number of levels for most groups. The observed running times for the simple algorithm were all less than $2.6n^2gm^2$ microseconds. The observed running times for the sophisticated algorithm were all less than $6.0ngm \max\{m \log_2(n), n-1\}$. Theoretical worst-case times would increase the second formula by a factor of m , but this large set of data suggests that such cases are rare.

Table 3 shows the running time for each algorithm for selected groups, including those that lead to the results of the previous paragraph. (The worst-case combinations are marked with asterisks.)

Algorithm			Simple	Sophisticated
Group	n	m		
$P\Gamma L_2(32)$	33	4	1.467*	0.233
C_{189}	189	1	1.733	1.667*
$B_{4,12}$	48	11	0.967	0.500
J_2	100	4	12.883	2.267
M_{24}	24	7	1.550	0.433
Rubik's cube	48	18	23.267	3.600
Magic Domino	40	14	1.683	0.817

Table 3. Measured running times in seconds for various groups.

Several of the groups came in regular families. Table 4 gives formulas to statistically model the observed running times as a function of n . The best integer power of n was selected based mainly on a worst-case analysis and verified with the data. The coefficient was fit to the data. All the algorithms had special code to recognize when the tail of the branching had become a straight line. This code sped up the algorithms for the symmetric group but did not help in other cases. There is a similar technique for alternating groups (based on recognizing when the branching becomes a straight line with a fork at the end), but that technique was not used.

Algorithm			Simple	Sophisticated
Group	n	m		
A_a	a	$a - 2$	$1.5n^5$	$3.7n^4$
C_a	a	1	$49.n^2$	$6.n^2 \log_2(n)$
D_a	a	2	$62.n^2$	$9.n^2 \log_2(n)$
S_a	a	$a - 1$	$0.3n^4$	$18.n^3$

Table 4. Fitted running times in microseconds for families of groups.

The sophisticated algorithm is fast under a wide range of conditions. For groups with a small base, it usually runs in time $O(n^2)$ with a small constant of proportionality. This is near-optimal since a labelled branching has $n^2 - n$ entries (for a transitive group). For groups with large bases there appears to be room for improvement.

One way that often quickly leads to complete a branching is to sift random products until the branching does not change for several sifts [7]. However, if one wants to be sure that the branching is complete one still needs prove that the random sifting has lead to a complete branching. The rapid completion of the sophisticated algorithm suggest that such approaches are unlikely to lead to significant improvement unless they use a faster approach to prove completeness.

Acknowledgements.

The authors would like to thank Namita Sarawagi for programming early versions of many of the algorithms discussed, and Cynthia Brown for many fruitful discussions.

References

1. L. Babai, E. Luks, and A. Seress, "On Managing Permutation Groups in $O(n^4 \log^c n)$ ", *Proc. 28th IEEE FOCS* (1988), 272-282.
2. C. A. Brown, G. Cooperman, and L. Finkelstein, "Solving Permutation Problems Using Rewriting Rules", to appear in *Proc. of the International Symposium on Symbolic and Algebraic Computation* (ISSAC 88, Rome, Italy), Springer Verlag Lecture Notes in Computer Science.
3. G. Cooperman and L.A. Finkelstein, "Short Presentations for Permutation Groups and a Strong Generating Test", submitted to *J. Symbolic Computation*.
4. M. Hall, Jr., *The Theory of Groups*, Macmillan, New York, 1959.
5. M. Jerrum, "A Compact Representation for Permutation Groups", *J. Algorithms* 7 (1986), 60-78.
6. D.E. Knuth, "Notes on Efficient Representation of Permutation Groups" (1981), unpublished manuscript.
7. J. Leon, "On an Algorithm for Finding a Base and Strong Generating Set for a Group Given by a Set of Generating Permutations", *Math. Comp.* 35 (1980), 941-974.
8. C.C. Sims, "Computation with Permutation Groups", in *Proc. Second Symposium on Symbolic and Algebraic Manipulation*, edited by S.R. Petrick, ACM, New York, 1971.

Finite-Basis Theorems and a Computation-Integrated Approach to Obstruction Set Isolation

(Extended Abstract)

Michael R. Fellows*, Nancy G. Kinnersley† and Michael A. Langston‡

Abstract. Recent advances in well-partial-order theory, especially the seminal contributions of Robertson and Seymour, make tools available that establish only the *existence* of polynomial-time decision algorithms. The finite-basis theorems that are the engine of these developments are inherently *nonconstructive*, providing no effective means for capturing the obstruction sets on which these polynomial-time algorithms are based. In this paper, we use a well-studied matrix permutation problem to describe an approach to obstruction set isolation that makes essential use of the computer in a mathematical proof. (In fact, the task of identifying such obstruction sets appears to pose many “4CT-like” problems for which computational assistance is vital.) We also discuss an approach based on a computational “learning” paradigm that incorporates a fundamental component of computer-aided obstruction identification with self-reduction to obtain *known* polynomial-time algorithms that do not depend on the knowledge of an entire obstruction set.

Key words. discrete computational mathematics, nonconstructive proofs, polynomial time complexity, well-partial-order theory

AMS subject classifications. 68C25, 68E10, 68K05

* Department of Computer Science, University of Idaho, Moscow, ID 83843. This author's research is supported in part by the National Science Foundation under grant MIP-8603879, by the Office of Naval Research under contract N00014-88-K-0456, and by the National Aeronautics and Space Administration under engineering research center grant AGN-1406.

† Department of Computer Science, Washington State University, Pullman, WA 99164-1210. This author's research is supported in part by the Washington State University Graduate Research Assistantship Program.

‡ Department of Computer Science, Washington State University, Pullman, WA 99164-1210. This author's research is supported in part by the National Science Foundation under grant MIP-8603879 and by the Office of Naval Research under contract N00014-88-K-0343. A portion of this author's work was done while visiting the Institute for Mathematics and its Applications at the University of Minnesota, Minneapolis, MN, and while visiting the Coordinated Science Laboratory at the University of Illinois, Urbana, IL.

1. Introduction

Nonconstructive methods based on advances in well-partial-order theory have recently been employed to prove low-degree polynomial-time decision complexity for a variety of combinatorial problems. Some of these problems were not previously known to be decidable in polynomial time at all; others were only known to be decidable in polynomial time with algorithms having unboundedly high-degree polynomial running times. A brief background is provided in the next section. For more details, we refer the reader to [RS1-RS4] for the relevant advances in graph theory and graph algorithms, and to [FL1-FL3] for sample applications of these novel tools.

Unlike algorithms devised with traditional methods, in which search or optimization routines are almost always used to provide some form of "positive evidence" for making a decision, algorithms based on applications of well-partial-order theory rely instead on the existence of a basis of "negative evidence." Curiously, all that is directly provided by the underlying theory is that the basis is *finite*. That is, we are guaranteed a Kuratowski characterization as the foundation for an efficient algorithm, but the proof of this is nonconstructive in that the characterization itself is *not* provided!

How then does one isolate the basis (aka obstruction set) and hence a decision algorithm? Since no completely general method is possible [FL4], one goal is to bring the enormous computational power now available to bear on specific problems. In this paper, we use the well-known GATE MATRIX LAYOUT problem to describe one such approach that incorporates traditional proof techniques with computation-based obstruction verification, and explain how the computer has played a fundamental role in obstruction set isolation. We observe that an analogy can be drawn between this approach and the proof of the Four Color Theorem: computational assistance is essential for certain steps of the proof that require exhaustive case-checking, but mathematical arguments are necessary outside of these computationally intensive steps to delimit and structure the search space. We also discuss a constructive "learning" strategy by which known polynomial-time algorithms can be devised based on only *partial* knowledge of an obstruction set.

2. Background: Nonconstructive Tools and Their Application

Graphs we consider are finite and undirected, but may have loops and multiple edges. A graph H is less than or equal to a graph G in the *minor* order, written $H \leq_m G$, if and only if a graph isomorphic to H can be obtained from G by a series of these two operations: taking a subgraph and contracting an edge. A family F of graphs is said to be *closed* under the minor ordering if the facts that G is in F and that $H \leq_m G$ together imply that H must be in F . The *obstruction*

set for a family F of graphs is defined to be the set of graphs in the complement of F that are minimal in the minor ordering. If F is closed under the minor ordering, it has the following characterization: G is in F if and only if there exists no H in the obstruction set for F such that $H \leq_m G$.

Theorem 1. [RS4] Any set of graphs contains only a finite number of minor-minimal elements. That is, graphs are *well-partially-ordered* by $\leq_m$.

Theorem 2. [RS3] For every fixed graph H , the problem that takes as input a graph G and determines whether $H \leq_m G$ is solvable in polynomial time. That is, graphs under $\leq_m$ possess *polynomial-time order tests*.

We term a well-partially-ordered set with polynomial-time order tests a *Robertson Seymour poset*, or *RS poset* for short. Remarkably, Theorems 1 and 2 guarantee only the existence of a polynomial-time decision algorithm for any minor-closed family of graphs. It has been shown that Theorem 1 is independent of constructive axiomatic systems and, indeed, any proof of Theorem 1 must use impredicative arguments [FRS].

An interesting feature of Theorem 2 is the low degree of the polynomials bounding the decision algorithms' running times. Letting n denote the number of vertices in G , the general bound is $O(n^3)$. If F excludes a planar graph, then the bound is $O(n^2)$. These polynomials possess outrageously large constants of proportionality, rendering them impractical for problems of any nontrivial size [RS2].

For an application of Theorems 1 and 2, and to define an illustrative problem that will be addressed in the remainder of this paper, consider GATE MATRIX LAYOUT [DKL]. From a practical standpoint, this combinatorial problem has been the focus of much recent attention, arising in several VLSI layout styles including gate matrix, PLAs under multiple folding, Weinberger arrays and others. From a more theoretical perspective, this problem has recently been shown [FL4] to be identical to that of determining the *path-width* of a graph, an important structural metric in the Robertson-Seymour theory [RS1]. Formally, we are given an $n \times m$ Boolean matrix M and a positive integer k , and are asked whether we can permute the columns of M so that, if in each row we change to * every 0 lying between the row's leftmost and rightmost 1, then no column contains more than k 1s and *s.

The general problem is NP-complete and no brute force polynomial-time method is known even when k is fixed. Nevertheless, it has been shown that, for any fixed value of k , an arbitrary instance can be mapped to an equivalent instance with only two 1s per column, then modeled as a graph on n vertices such

that the family of “yes” instances is closed under the minor order and excludes a planar graph.

Theorem 3. [FL1] For any fixed k , GATE MATRIX LAYOUT can be decided in $O(n^2)$ time.

But what are the finite obstruction sets on which the last theorem relies? In what follows, we shall focus almost exclusively on the fixed value $k = 3$. As will be seen, this case alone is sufficiently nontrivial and in fact surprisingly challenging.

3. An Assault on the $k = 3$ Case: The Structural Component

For $k = 1$, it is trivial that the only obstruction is K_2 . For $k = 2$, it is known that there are only two obstructions, namely, K_3 and $S(K_{1,3})$ [BFKL, FL1]. Thus one might hope that the obstruction sets grow with k in a predictable pattern, eliminating a need for computational tools. Such optimism is quickly shattered, however, when one considers the $k = 3$ case; it turns out that there are 110 obstructions! Our proof of this is built on both structure theory and computation. The former, as outlined in this section, is based on proving that the search space for the obstruction set can be bounded using arguments that focus on the structure of the graphs in the set. The latter, as discussed in the next section, involves an efficient dynamic programming formulation and well-chosen minimality tests to help identify the elements of the set. Together, this integrated attack solves the problem.

We now list a few of our most useful results. Some of these help to identify directly certain members of the obstruction set.

Theorem 4. If three (nondistinct) obstructions for $k = 2$ are connected by identifying a vertex of each with a distinct leaf of a ternary tree of height one, then twenty distinct obstructions for $k = 3$ are obtained: ten trees, six graphs containing a single triangle, three containing two triangles, and one with three triangles. Moreover, the ten trees so obtained are the only tree obstructions for $k = 3$.

Theorem 5. The obstruction set for $k = 3$ contains only five graphs that are not outerplanar.

Other results help to identify local reduction and replacement rules that preserve “yes” instances.

Theorem 6. Suppose G contains a vertex u of degree one adjacent to a vertex v of degree four or more. If the edge (u, v) is contracted (equivalently, if the vertex

u is deleted), then the resulting graph is a “yes” instance for $k = 3$ if and only if G is.

Theorem 7. Suppose G contains a triangle with vertices u , v , and w in which only vertex w has degree three or more. If the triangle is replaced with $S(K_{1,3})$ (that is, the three edges of the triangle are contracted and an arbitrary vertex of $S(K_{1,3})$ is identified with w), then the resulting graph is a “yes” instance for $k = 3$ if and only if G is.

Many results can be generalized to arbitrary values of k .

Theorem 8. Suppose G contains a cycle with exactly one vertex of degree greater than two. If the cycle is contracted to a triangle, then the resulting graph is a “yes” instance for any fixed value of k if and only if G is.

This makes it possible to reduce the number of graphs that need to be considered, so that more powerful statements are possible.

Theorem 9. When embedded in the plane, no obstruction for $k = 3$ has more than four internal faces.

The proofs of these kinds of theorems are often laborious, especially for those that analyze the possible face patterns of obstructions. For example, the proof of Theorem 9 alone requires over twenty pages [Ki]. But, the result is that we can bound the search space for $k = 3$ obstructions.

By extensions of theorems such as those listed in this section, a significant subset of the $k + 1$ obstruction set can be obtained in a systematic manner from the k obstruction set, for any fixed k . Theorem 4, for example, can be extended in this fashion. Interestingly, of the 110 obstructions captured and subsequently enumerated with the aid of the work described in the next section, at most 105 satisfy a conjecture that was based on these extensions and that was long suspected to be sufficient to enumerate all obstructions. This illustrates the difficulty of ruling out the existence of “sporadic” obstructions without computer verification.

4. An Assault on the $k = 3$ Case: The Computational Component

Once a candidate graph G has been identified as a possible obstruction, we encounter a task that is extremely time-consuming (even on the fastest computers available), namely, determining whether G is truly an obstruction. In this regard, we must of course answer two questions:

- (1) is G a “no” instance and, if so,

(2) is every proper minor of G a "yes" instance?

To answer the first question, we must somehow consider the $O(m!)$ possible column permutations. Obviously, brute force alone is not enough. Furthermore, for graphs that pass the first test, we must do a lot better than mere brute force if we are to answer the second question as well!

In order to accomplish this task, we exploit what we already know from our face-pattern analysis and trade space for time. In particular, we take advantage of and improve on the dynamic programming formulation devised for this problem in [DKL]. This gives rise to a time complexity figure of only $O(m^2 2^m)$. Our algorithms are coded in Pascal and implemented on an IBM 3090-300E, with 12 megabytes of *real* (nonvirtual) memory used for the huge dynamic programming tables required.

To perform the second step efficiently, observe that minor-minimal graphs must be connected, so that we need only check single edge deletions and single edge contractions (more complex operations and vertex deletions are unnecessary). Furthermore, we are able to deduce from the structure theorems, a few of which were mentioned in the last section, that the contraction of an edge on a pendant path is equivalent to the removal of the pendant edge, so that only the edges of faces need to be contracted.

We have tailored the dynamic programming formulation to take advantage of structural features common to obstructions. For example, we know that no obstruction can contain a pendant path of length exceeding two. Moreover, whenever there is a pendant path of length exactly two, then if there exists any satisfactory permutation of the columns of the corresponding matrix, then there exists one in which the edges of the path are mapped to adjacent columns. We can therefore treat these edges as a unit when evaluating permutations (this is possible during both steps). Similarly, whenever a triangle is encountered, we collapse the three columns to which its edges are mapped to a single column with three 1s (this makes sense in the first step only).

The result is that, on a typical candidate obstruction with 13 vertices and 15 edges, less than 40 seconds of CPU time has usually been sufficient to verify that an obstruction has been found. For a graph with 18 vertices and 20 edges, a figure of around 30 CPU minutes has been more the norm. When we move up to a tree with 22 vertices (and hence 21 edges), this task has generally taken just over 2 CPU hours. (These figures reflect run times on actual obstructions. Nonobstructions frequently take much less.) To get a feel for the way resources scale, a $k = 4$ candidate obstruction with 24 vertices and 39 edges recently required a little under 9 hours of CPU time to perform the first step alone; we are fairly certain that this graph is an obstruction and anticipate that the minimality tests needed for the second step will take approximately 120 CPU hours. Thus, we are

truly working at the cutting edge of feasible computation.

5. An Automated Learning System

We now outline a novel approach to deciding membership for a minor-closed family of graphs. This scheme is a natural candidate for parallelization. (At this time, the aforementioned astronomical constants associated with arbitrary fixed minor tests preclude the implementation of such a system, even on the fastest-available computers. Optimistically, however, we observe that much work currently underway has reported that these constants can be drastically reduced. See [FL4] for details.) The approach we shall describe is especially attractive in light of the following two factors: (1) we have recently shown that there are more than a million obstructions for $k = 4$ and (2) empirical evidence suggests that only a handful of obstructions are needed in most applications. Thus, as k grows, we find ourselves in a remarkable situation: it appears to get *much* harder to isolate an obstruction set and at the same time it seems that *much* less than the full set is required in practice.

Our system is composed of these three major subsystems:

- (1) a *decision* component that establishes whether a satisfactory column permutation may be possible,
- (2) a *construction* component that attempts to produce an appropriate permutation when the decision component has specified that it may be possible, and
- (3) a *learning* component that is invoked whenever the construction component fails.

Each invocation of the learning component is guaranteed to increase the knowledge base, thereby improving the expected future performance of both the decision and construction components.

If the decision component reports "no," then it is certain that no permutation is possible. This component is simply a battery of minor-containment tests, one for each known obstruction. If, however, it reports "yes," then either "yes" is in fact the correct answer, or there is insufficient data in the system's knowledge base to determine why "no" is instead the right response. That is, a "yes" answer may merely reflect that not enough obstructions are known. Given a "yes" answer, we next employ the construction component, which attempts to produce an appropriate permutation with a technique known as *self-reduction*. Construction proceeds by selectively modifying the input and repeatedly reinvoking the decision component so as to isolate a satisfactory permutation when any exist. If the construction should fail, then our knowledge base is missing at least one obstruction and the learning component is invoked to find out why. The learning component can even be brute-force in nature, as long as we are sure *never* to invoke it when

all obstructions are known.

Space restrictions rule out more details here, but the system we have suggested inherently relies on a more general result, a simplified version of which follows.

Theorem 10. [FL4] Let F denote a closed family in an RS poset and suppose the following are known:

- (1) an algorithm that checks candidate solutions to the search version of F in $O(T_1(n))$ time,
- (2) order tests that require $O(T_2(n))$ time,
- (3) a self-reduction algorithm that requires $O(T_3(n))$ time, and
- (4) some decision algorithm (its time bound is immaterial).

Then an algorithm requiring $O(\max\{T_1(n), T_2(n) \cdot T_3(n)\})$ time is *known* that solves the search (and hence the decision) problem for F .

As previously mentioned, some obstructions seem far more likely to be encountered in practice than others. To illustrate, we have reviewed a number of real GATE MATRIX LAYOUT “no” instances for $k = 3$ from the VLSI literature. Interestingly, every one has contained as a minor K_4 . Moreover, aside from K_4 , only four other obstructions were found. Thus, from a purely practical standpoint, the learning system looks promising.

To demonstrate how such a system might work, let $k = 4$, a value for which only a handful of obstructions have been recognized (and one whose obstruction set is so large that it is unlikely that all of its members can be isolated without massive computation). Consider the family of instances presented in [DKL], where it was shown that all previously-published greedy heuristics fail for values of k that are within *any* constant multiple of the optimum. (Each instance in the family has a “hard to find” $k = 4$ permutation.) Assuming the matrix has at least seven rows, it turns out that the corresponding graph contains only three vertices whose respective degrees each exceed three. Let us denote these three by A, B , and C . A is connected to every vertex in the graph except B and C . B and C are connected to every vertex in the graph except A . No other edges are present. From this it follows that the construction component cannot change a 0 to a 1 in any row corresponding to a vertex other than A, B , or C , because doing so would introduce the known obstruction K_5 . After the self-reduction is completed, exactly $3m/2$ entries have been changed from 0 to 1. At this time, the resultant matrix enjoys the “consecutive ones property.” It is now a simple task to construct a $k = 4$ permutation.

6. Summary

We have presented a computation-integrated approach to the problem of obstruction set isolation, using GATE MATRIX LAYOUT with $k = 3$ as our

archetypical example. We have also discussed a more general learning strategy for constructivization of RS poset applications that we think may be a promising topic for future study.

It should be clear that, for problems of this magnitude, neither traditional mathematics nor computational tools alone suffice: simply bounding the search space leaves too many elements for the individual obstructions to be isolated by hand; just boiling away nonminimal elements is fruitless in a crucible of unbounded size. For the sorts of problems that are now posed by the recent advances in well-partial-order theory, it is satisfying to see that mathematical analysis can work hand-in-glove with the efficient use of computational methods.

References

- [BFKL] R. L. Bryant, M. R. Fellows, N. G. Kinnersley and M. A. Langston, "On Finding Obstruction Sets and Polynomial-Time Algorithms for Gate Matrix Layout," *Proc. 25th Allerton Conf. on Communication, Control and Computing* (1987), 397-398.
- [DKL] N. Deo, M. S. Krishnamoorthy and M. A. Langston, "Exact and Approximate Solutions for the Gate Matrix Layout Problem," *IEEE Trans. on Computer-Aided Design* 6 (1987), 79-84.
- [FL1] M. R. Fellows and M. A. Langston, "Nonconstructive Advances in Polynomial Time Complexity," *Info. Proc. Letters* 26 (1987), 157-162.
- [FL2] ———, "Nonconstructive Tools for Proving Polynomial-Time Decidability," *J. of the ACM* 35 (1988), 727-739.
- [FL3] ———, "Layout Permutation Problems and Well-Partially-Ordered Sets," *Proc. 5th MIT Conf. on Advanced Research in VLSI* (1988), 315-327.
- [FL4] ———, "On Search, Decision and the Efficiency of Polynomial-Time Algorithms," *Proc. 21st ACM Symp. on Theory of Computing* (1989).
- [FRS] H. Friedman, N. Robertson and P. D. Seymour, "The Metamathematics of the Graph Minor Theorem," in Applications of Logic to Combinatorics, American Math. Soc., Providence, RI, to appear.
- [Ki] N. G. Kinnersley, "Obstruction Set Isolation for Layout Permutation Problems," Ph. D. Thesis, Washington State University, to appear.
- [RS1] N. Robertson and P. D. Seymour, "Graph Minors I. Excluding a Forest," *J. Comb. Th. Ser. B* 35 (1983), 39-61.
- [RS2] ———, "Graph Minors V. Excluding a Planar Graph," *J. Comb. Th. Ser. B* 41 (1986), 92-114.
- [RS3] ———, "Graph Minors XIII. The Disjoint Paths Problem," to appear.
- [RS4] ———, "Graph Minors XVI. Wagner's Conjecture," to appear.

PRACTICAL DETERMINATION OF THE DIMENSION OF AN ALGEBRAIC VARIETY

André Galligo — Carlo Traverso

Université de Nice et INRIA — Università di Pisa

INTRODUCTION

The determination of the dimension of an algebraic variety is a problem that, in principle, can be solved in an algorithmic way. If the variety is projective, this can be made through the computation of the Hilbert polynomial; if it is affine, consider a completion of the affine variety; the completion may have larger dimension than the affine variety, but in this case it may have irreducible components contained in the hyperplane at infinity. This can be checked, and if this happens, a decomposition into irreducible components reduces the problem to the other case. Moreover, if the completion of the affine variety is made with respect to the affine immersion given by a standard basis (also called Gröbner basis) with respect to an homogeneous term ordering, no irreducible component may be contained in the hyperplane at infinity. Hence, a standard basis computation is sufficient to decide the question.

Another connected problem is to determine whether a variety is pure dimensional. The problem is very hard, but in principle this too can be computed through an irreducible, or an equidimensional, decomposition. [GTZ], [BG], [BGS]

The complexity of these algorithms is in general very high. Many computations seemingly simple are practically impossible. In this case, one can use indirect alternative approaches. In this paper we show a few of them, that have succeeded, or failed, in four hard examples. These computations may provide a guide for the solution of other problems of the same kind. For some of the computations, that need to repeat similar computations many times (e.g. they may require to compute intersections with hyperplanes hundreds of times) we have not completed the computations, but were satisfied to compute a few cases just to show that the complete computation is possible and a different answer very unlikely: we are interested in methods, not in the specific answer to a particular problem. We point also that the trace methods described in [Tr1] can be used in these repeated computations to decrease the cost of the complete answer.

All the computations were made with the AIPi system, that is a system for standard basis computation written in MuLISP under MS-DOS [Tr2]. Hence "possible" and "impossible" computations are meant with respect to a quite small system, that can be used on any 640K IBM-PC compatible computer. Actually, the computations were made on a computer with an Intel 80836 processor, but the only difference with smaller MS-DOS computers would be execution times.

Of course, on a larger computer some of the "impossible" computations could become possible, but the suggestions for indirect computations remain useful.

Research with the contribution of Ministero della Pubblica Istruzione (Italy) and C. N. R. S. (France).

THE EXAMPLES

Example 1.

$$\begin{aligned}
 (1.1) \quad & x^3y^2 + 4x^2y^2z - x^2yz^2 + 288x^2y^2 + 207x^2yz + 1152xy^2z + 156xyz^2 \\
 & + xz^3 - 3456x^2y + 20736xy^2 + 19008xyz + 82944y^2z + 432xz^2 \\
 & - 497664xy + 62208xz + 2985984x \\
 (1.2) \quad & y^3t^3 + 4y^3t^2 - y^2zt^2 + 4y^2t^3 - 48y^2t^2 - 5yzt^2 + 108yzt + z^2t \\
 & + 144zt - 1728z \\
 (1.3) \quad & - x^2z^2t + 4xz^2t^2 + z^3t^2 + x^3z + 156x^2zt + 207xz^2t + 1152xzt^2 \\
 & + 288z^2t^2 + 432x^2z + 19008xzt - 3456z^2t + 82944xt^2 + 20736zt^2 \\
 & + 62208xz - 497664zt + 2985984z \\
 (1.4) \quad & y^3t^3 - xy^2t^2 + 4y^3t^2 + 4y^2t^3 - 5xy^2t - 48y^2t^2 + x^2y + 108xyt \\
 & + 144xy - 1728x
 \end{aligned}$$

Example 2.

$$\begin{aligned}
 (2.1) \quad & -x^3y^2 + 2x^2y^2z - x^2yz^2 - 144x^2y^2 - 207x^2yz + 288xy^2z + 78xyz^2 \\
 & + xz^3 - 3456x^2y - 5184xy^2 - 9504xyz - 432xz^2 + 10368y^2z \\
 & - 248832xy + 62208xz - 2985984x \\
 (2.2) \quad & -x^3zt^2 + x^2z^2t^2 - 6x^3zt + 4x^2z^2t + 32x^3t^2 - 72x^2zt^2 - 87xz^2t^2 \\
 & - z^3t^2 - 8x^3z - 432x^2zt - 414xz^2t + 2592xzt^2 + 864z^2t^2 \\
 & - 1728x^2z - 20736xzt + 3456z^2t - 186624zt^2 - 124416xz \\
 & - 1492992zt - 2985984z \\
 (2.3) \quad & x^2yt^3 - 2xy^2t^3 + y^3t^3 + 8x^2yt^2 - 12xy^2t^2 + 4y^3t^2 - 24xyt^3 \\
 & + 24y^2t^3 + 20x^2yt - 20xy^2t - 160xyt^2 + 96y^2t^2 + 128xt^3 \\
 & + 16x^2y + 96xyt + 2304xt^2 + 1152xy + 13824xt + 27648x \\
 (2.4) \quad & y^3t^3 - y^2zt^3 + 4y^3t^2 - 2y^2zt^2 + 72y^2t^3 + 71yzt^3 + z^2t^3 + 288y^2t^2 \\
 & + 360yzt^2 + 6z^2t^2 + 1728yt^3 - 464zt^3 + 432yzt + 8z^2t + 6912yt^2 \\
 & - 4320zt^2 + 13824t^3 - 13824zt + 55296t^2 - 13824z
 \end{aligned}$$

Example 3.

Is the same as Example 2, but in the equation 2.4 in place of the monomial z^2t^3 we have a monomial z^2

Example 4.

$$\begin{aligned}
 (4.1) \quad & x + y + z + t + u + v \\
 (4.2) \quad & xy + yz + zt + tu + ux \\
 (4.3) \quad & xyz + yzt + ztu + tuv + uvx + vxy \\
 (4.4) \quad & xyzt + yztu + ztuv + tuvx + uvxy + vxyz \\
 (4.5) \quad & xyztu + yztuv + ztuvx + tuvxy + uvxyz + vxyzt \\
 (4.6) \quad & xyztuv - 1
 \end{aligned}$$

Examples 1 and 2 were suggested by H. Cohn, (private communication), and arise as modular equations for special algebraic number fields, [Co], [CD]. Example 3 comes from a misprint in the input of example 2, but we have kept it since it illustrates a different and useful approach. Example 4 was popularized by J. H. Davenport, but we were unable to retrace its origin.

THE METHODS

Guessing the answer. The first point is: it is usually easier to prove that an answer is correct than computing it directly. Hence it is important to guess the answer.

We obtain a guess for the dimension of an algebraic variety V intersecting it with random linear varieties: if the dimension of V is n , then the intersection with n generic hyperplanes is non-empty, and the intersection with $n+1$ hyperplanes is empty. Taking random hyperplanes instead of generic ones we obtain a guess. The computations can be made modulo a prime number. Of course, we can only probabilistically guarantee that our guess is correct. We repeat the computations with different hyperplanes and prime numbers: when we are convinced of the correctness of the guess, we can choose a method to prove that the guess is correct.

In our four examples the guess is: Examples 1 and 2 have dimension 2, example 3 has dimension 1, example 4 has dimension 0.

The direct computation. The direct computation consists in finding a standard basis of the ideal I defining the variety. From this standard basis, we can compute the dimension of V , since the dimension of V is equal to the dimension of the variety defined by the ideal $In(I)$ generated by the initial forms of I .

This is known if the ideal I is homogeneous or quasi-homogeneous (see [MM], [Gi]). Indeed, the Hilbert function of I is equal to the Hilbert function of $In(I)$.

If I is not homogeneous, and the term-ordering is compatible with the degree (i.e. monomials of larger degree are larger), we can consider the ideal $\bar{I}$, obtained homogenizing a set of generators of I . We extend the term-ordering, considering the homogenization variable to be smaller than all other variables. If the set of generators of I is the standard basis of I , homogenizing them we obtain a standard basis of the homogeneous ideal $\bar{I}$, and $\bar{I}$ contains the homogenization of any element of I . Hence considering $In(I)$ we can compute the dimension of the projective variety $\bar{V}$ defined by $\bar{I}$.

Consider now the intersection V_0 of $\bar{V}$ with the hyperplane at infinity. Its homogeneous ideal I_0 is obtained from $\bar{I}$ sending the homogenization variable to 0, hence the initial ideals of I and I_0 are the same. This proves that the dimension of $\bar{V}$ is one more than the dimension of V_0 , (the ambient space has one dimension more) hence $\bar{V}$ has no top-dimensional component at infinity, hence the dimension of V and $\bar{V}$ are the same.

In the case that the term-ordering is not compatible with the degree, but is compatible with a positive weight, we have to consider the quasi-homogenization of the ideal, with the same proof; and in the case that the term-ordering is not compatible with a positive weight, (as the lexicographic term-ordering), one can find a suitable term-ordering compatible with a positive weight such that the standard bases with respect to the two term-orderings are the same.

To our knowledge, this theorem was independently proved many times, but was never explicitly written.

This "direct computation" is the key to all proofs of dimension, but often it is computationally impossible (or at least very difficult) to use it on a given ideal. Example 1 is a case in which this direct approach is possible with respect to the "homogeneous

reverse-lexicographic term-ordering." The initial ideal is generated by $xy^2t^2, x^3y^2, y^3t^3, y^3zt^2, x^2y^2zt, x^2y^3t, x^4yt, y^2zt^4, y^2z^2t^3, xyz^2t^3, x^2yzt^3, y^2z^3t^2, xyz^3t^2, x^2yz^2t^2, x^2z^4t, xy^2z^3t, x^2yz^3t, x^3z^3t, xy^3z^2t, x^3yz^2t, x^4yz^2, x^5yz, xyz^5t, yz^3t^4, yz^4t^3, yz^5t^2, xy^6t, x^3z^2t^4, y^6z^3t$. Inspecting this set, we easily remark that no monomial in x and z alone appears in the ideal, hence the variety defined by it contains the (x, z) plane (indeed, set-theoretically is composed of 6 planes). Moreover no coordinate hyperplane satisfies the equations, hence the dimension is 2.

An additional question that was raised by Cohn is the following: do any three equations generate the ideal? Do any two equations define a surface?. The computation of the standard basis of the ideal generated by three equations is not feasible, hence indirect methods have to be used.

The answer is no to the first question, and yes to the second. The first question is answered adding to both ideals (the original one and the one generated by three equations) a linear form (e.g. $t = 1$) and computing the standard bases of the two ideals: they are different. However, one can guess that the two ideals differ only by 1-dimensional components, since adding two random linear forms the ideals coincide. This guess could be proved to be correct checking sufficiently many pairs of hyperplanes (see the next subsection). The second question is answered remarking that the original equations have no common factor.

Too many lines in mutual generic position cannot meet a space curve. We want to prove that in example 1, the variety V' defined by the equations 2.1, ..., 2.3 coincides with the variety V defined by the full set up to a 1-dimensional subvariety. To prove this fact, assume that they differ in a 2-dimensional subset S ; then this subset would be composed of components of the intersection of the first and the third equation, hence by a surface of degree d being at most 25. Indeed, since the variety defined by the full set of equations has degree 6 (computed by the basis of the initial ideal), d is at most 19. Consider now sufficiently many sufficiently generic 2-planes P_i , pairwise meeting at one point at finite distance, and such that $P_i \cap V = P_i \cap V'$ (ideal-theoretic equality). This means that the P_i do not meet S at finite distance, hence in the projective space they meet the closure of S at infinity. If the configuration of the lines at infinity of the P_i is such that no curve of degree 19 in the 3-plane at infinity can meet all of them, then we have proved that S cannot exist, hence V and V' should coincide outside a 1-dimensional subset.

The precise number of such planes, and the meaning of genericity involved, can be found with Schubert calculus. Let C be a space curve of degree ≤ 19 . Consider the grassmanian of lines in three-space, and the hypersurface R of lines meeting C . This is an hypersurface of degree ≤ 19 in the grassmanian, and using the postulation formula on the grassmanian one sees that $\frac{20 \cdot 21^2 \cdot 22}{12} = 16170$ points in generic position cannot meet R .

This means that 16170 planes in generic position in the affine 4-space cannot simultaneously avoid the surface S . The genericity of the position means that the rank of a suitable matrix defined by the Plücker coordinates of the planes is maximal. This last computation can be either made modularly, or we can choose the planes in a systematic way guaranteeing the genericity. The details of the proof, in the general case, will be shown elsewhere.

The whole of the resulting computation is extremely long, but any single step is independent of the others (the parallel complexity is low).

Remark that this method can be used to prove that a variety has at most dimension 1: it is sufficient to take $V = \emptyset$, and V' as the variety under consideration.

In our case, the following method gives a far better answer.

Find a linear variety that does not meet the part at infinity. We are considering example 1, limited to equations 1, 2, 3, and we want to prove that the corresponding variety

V' coincides with the variety V defined by all four equations up to a subset of dimension 1. We consider the variety V'' defined by the first and the third equation. We compute its standard basis, and we verify that its degree is 25, i.e. the corresponding projective variety does not have a component at infinity. Now, a sufficiently generic 2-plane intersects V'' with multiplicity 25, hence does not intersect V'' at infinity. Hence it does not intersect V' at infinity, hence should meet every 2-dimensional component of V' at finite distance. If the intersection of the plane with V' coincides with the intersection with V , then no extra 2-dimensional component can exist.

The 2-plane of equations $z = x + y$, $t = x - y + 1$ is sufficient to prove the result.

Intersecting with sufficiently many hyperplanes, lower bound. The second example is more complicated, since the direct computation is impossible (with our software and hardware, and also with some more powerful one). It is however possible to compute the dimension indirectly. The dimension is at most 2, since the equations have no common divisor (they even do not have common variables), hence their locus cannot have common hypersurfaces. We want to show that the dimension cannot be 1 (nor 0: the y -axis is inside the locus).

Let V be the affine variety defined by the equation of example 2. Assuming that its dimension is 1, we can bound its degree: V is contained in a complete intersection of generic linear combinations of the equations, hence, assuming that it is a curve, it has degree at most $6^3 = 216$. It follows that it can have at most 216 components: we already have one, the y -axis. If we find 216 more hyperplanes not containing the y -axis and not intersecting at finite distance (e.g. hyperplanes $y = a$) that meet V in a curve, we have proved that V contains a surface. We have computed several of these intersection, with a in the range $-200 \dots 200$, and the intersection was a curve, and indeed a curve of degree 6 (V instead is of guessed degree 9). So 36 such intersections are sufficient to show that V contains a surface. We did not compute 36 intersection, since we had more interesting things to do, but the computation is surely feasible: computing a single intersection in that range took less than 4 minutes.

Looking at infinitesimal neighborhoods. Still in example 2, we can show that the y -axis is an irreducible component (indeed, with multiplicity 2). This can be made showing that the tangent cone at the origin of the intersection of V with $y = 1$ is a double point in the t -direction. The computation is almost immediate once you have a standard basis for the intersection, and we have made this computation in the former section. In more difficult cases one can use the tangent cone algorithm ([Mo]). The dimension of the tangent cone at a point is equal to the dimension of the variety at this same point. We can use this computation to give a lower bound for the dimension of a variety (we cannot exclude that we have higher dimensional components far from the point). We have no hard examples of successful applications of this criterion.

Intersecting with sufficiently many hyperplanes, upper bound: analyzing asymptotes of codimension 2. We consider example 3; it contains the y -axis, and we want to show that it cannot contain a surface (a pure 2-dimensional component). Let W be the closure in the projective 4-space of this surface. It would be contained in the intersection of the projective locus of the first two equations, hence its degree d would be at most 30 (or indeed 29, since it has a linear component at infinity; with a closer analysis one can even obtain better bounds).

We want to show that if we find at least $d + 1$ hyperplanes pairwise intersecting at finite distance, not all of them parallel to the same 2-plane, and meeting the affine variety V in a finite number of points, then W cannot exist.

For any such hyperplane H , $W \cap H$ is contained in the hyperplane at infinity H_0 , hence it should contain a component of $W \cap H_0$ (maybe with multiplicity, in such a way that the degree of the intersection counting multiplicities is d). These components are at most d . If H_1, H_2 are two hyperplanes meeting at finite distance, then $H_1 \cap H_2 \cap H_0$ is a line without multiplicity, hence they cannot both only contain the same component of W , that would be a line with multiplicity 1 (unless W itself is a linear variety, and this is ruled out by the condition that not all the planes missing W at finite distance are parallel to the same 2-plane).

It follows that any hyperplane in the collection should contain a different component of $W \cap H_0$, hence we cannot have more than d of them. In our example, it is not difficult to find 36 such hyperplanes (we didn't, for the same reason expressed in the former example; each computation is several minutes long) hence to prove that example 3 has dimension 1.

Counting points. Still in example 3, if we want to find that V contains other curves besides the y -axis, we consider that it cannot contain more than $5 \cdot 6^3$ points (or even $(5 \cdot 6^2 - 1) \cdot 6$ considering that it contains the y -axis); it is sufficient to count the points on the hyperplanes (and on their intersections) to find more than that many points on V .

We now consider the third example; let V be the affine variety defined by the equations 4.1, ..., 4.6 (it is non empty, since we can easily show points on it).

To prove that V has dimension 0, we consider the projective variety $\bar{V}$ obtained homogenizing the equations 4.i. We have different possibilities to prove that V has dimension 0.

Analyzing the part at infinity. Let H be the hyperplane at infinity; if $H \cap \bar{V} = \emptyset$, then V has dimension 0. This is unfortunately false in our example.

Avoiding the part at infinity. If the $H \cap \bar{V} \cap H' = \emptyset$, where H' is an hyperplane not intersecting $\bar{V}$ outside H , then V has dimension 0. Unfortunately, this is false in our example, since we can easily prove that $H \cap \bar{V}$ is composed of 5 curves (2 straight lines and 3 conics, intersecting variously).

Avoiding singularities at infinity. If we can find an hyperplane, meeting $\bar{V} \cap H$ in a non-singular point, and not meeting $\bar{V}$ outside H , then V has dimension 0: dimension ≥ 1 is ruled out since otherwise the intersection point should be common to two components, the one at infinity, and the other at finite distance, having the point as asymptote. Unfortunately this too is impossible, since we can easily see that every point of $\bar{V}$ lying in H has multiplicity at least 6.

These methods being fruitless, we have tried another idea: we want to prove that the irreducible components of $\bar{V}$ that meet H are contained in H (this leaves only points). The three following methods follow successfully this idea in different ways.

Analyzing 1-dimensional asymptotes. Bound the degree of the one-dimensional components. This allows to bound the number of hyperplanes in a pencil that do not meet these components, (144 in our case: one verifies—with indirect methods!—that intersecting all the equations of example 4 but the one of degree five one has a curve of degree $\leq 144 = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 6$), unless an hyperplane of the pencil contains it. Hence computing a finite number of hyperplanes we can settle the question. We have hence to compute the intersection of $\bar{V}$ with hyperplanes of equation $x = a$, and if the intersection is empty for 144 values of a this means that the curves at finite distance are all contained in some $x = b$. Since the variety is symmetric under permutation of variables, this settles the problem. The computation was only partly done, but is completely feasible.

Analyzing multiplicities. Look at the Hilbert function of the affine cone corresponding to the projective variety at its vertex. If the multiplicity is equal to the multiplicity of the part at infinity, then no other component of the same dimension can be present.

Study a neighborhood of the component at infinity. Prove that exists a neighborhood A of H such that $A \cap \bar{V} \subseteq H$. The complementary of any such neighborhood contains only points, hence the question is settled.

The two last methods lead approximately to the same computations, and these were completely performed. Hence we achieved the proof that V has dimension 0. We discuss with some details the computation with the last method.

The projective variety $\bar{V}$ is defined homogenizing the equations 4.1; this means that the equation 4.6 has to be substituted by

$$(4.6^*) \quad xyztuv - w^6,$$

where w is the homogenization variable (the other equations are already homogeneous). The question being local, we can de-homogenize, i.e. we can take subsequently the open sets $x \neq 0$, $y = 0$, etc.; by symmetry, it is sufficient to consider one of them, say $x \neq 0$, i.e. we can add the equation $x = 1$.

Consider now the ideal I obtained adding to the equations 4.1, ..., 4.5, 4.6* the equations $x - 1$ and w^k . If we prove $w^h \in I$ for some $h < k$, we have proved $w^h(1 + \psi w^{k-h}) \in (*, x - 1)$; and this is precisely what we need, the open set A containing H being defined by $1 + \psi w^{k-h} \neq 0$, and w^h defining the hyperplane H .

The computations succeeds with $k = 12$ (indeed, it is clear that one can better try only multiples of 6, hence this is the first case to try). Hence the proof is complete.

REFERENCES

- [BG] D. Bayer, A. Galligo, *Inverse systems and equidimensional decomposition of polynomial ideals*, (in preparation).
- [BGS] D. Bayer, A. Galligo, M. Stillman, *Computation of the primary decomposition of polynomial ideals*, (in preparation).
- [Co] H. Cohn, *An explicit modular equation in two variables and Hilbert's Twelfth Problem*, Math. of Comp. 38 (1982), 227-236.
- [CD] H. Cohn, J. Deutsch, *An explicit modular equation in two variables for $\mathbb{Q}[\sqrt{3}]$* , Math. of Comp. 50 (1988), 557-568.
- [GPT] A. Galligo, L. Pottier, C. Traverso, *Greater Easy Common Divisor and standard basis completion algorithms*, in "ISSAC 88," Lecture Notes in Computer Science (to appear), Springer Verlag, Berlin-Heidelberg-New York, 1988.
- [Gi] M. Giusti, *Combinatorial dimension theory of algebraic varieties*, Journal of Symbolic Computation (to appear).
- [GTZ] P. Gianni, B. Trager, G. Zacharias, *Gröbner Basis and Primary Decomposition of Polynomial Ideals*, Journal of Symbolic Computation.
- [MM] M. Möller, T. Mora, *The computation of Hilbert function*, in "EUROCAL 83," Lecture Notes in Computer Science 162, Springer Verlag, Berlin-Heidelberg-New York, 1983, pp. 157-167.
- [Mo] T. Mora, *An algorithm to compute the equations of tangent cones*, in "EUROCAM 82," Lecture Notes in Computer Science 144, Springer Verlag, Berlin-Heidelberg-New York, 1982, pp. 158-166.
- [Tr1] C. Traverso, *Gröbner trace algorithms*, in "ISSAC 88," Lecture Notes in Computer Science (to appear), Springer Verlag, Berlin-Heidelberg-New York, 1988.
- [Tr2] C. Traverso, *Experimenting the Gröbner basis algorithm with the AlPi system*, (submitted to ISSAC 89).

A Computer Generated Census of Cusped Hyperbolic 3-Manifolds

Martin Hildebrand
Department of Mathematics
Harvard University
Cambridge, MA 02138

Jeffrey Weeks
137 Fayette St.
Ithaca, NY 14850

Abstract. *In this paper, we describe how we used a computer to produce a census of cusped hyperbolic 3-manifolds obtained from 5 or fewer ideal tetrahedra. We note some of the techniques involved in writing and debugging the programs and give a brief summary of the results.*

Introduction

We have used a computer program to produce a census of cusped hyperbolic 3-manifolds obtained from 5 or fewer ideal tetrahedra. By census, we mean that we have found all such hyperbolic 3-manifolds and that we have been able to identify them uniquely. We are also able to identify certain characteristics of the manifolds.

A hyperbolic 3-manifold is a 3-manifold with a Riemannian metric such that each point has a neighborhood isometric to a neighborhood in hyperbolic 3-space. We normalize the metric to have constant sectional curvature -1 . We also require that our manifolds be complete; i.e. all Cauchy sequences converge to a point in the manifold.

Thurston has conjectured that every compact 3-manifold can be explained in terms of eight locally homogeneous geometries. Hyperbolic geometry is one of these geometries, and informally "most" 3-manifolds are hyperbolic 3-manifolds. Manifolds in the other seven geometries are generally well understood. Thus to understand 3-manifolds in general, one needs to understand hyperbolic 3-manifolds.

Hyperbolic 3-manifolds may contain some things known as cusps. Topologically, a cusp looks like a torus cross a half-open interval (or, in a non-orientable cusp, a Klein bottle cross a half-open interval). One way to look at an example of a cusp is to consider the complement of a knot in the 3-dimensional sphere. By complement, we are referring to the portion of the 3-dimensional sphere outside of the knot. In most cases, this complement is a hyperbolic 3-manifold. View a tubular neighborhood of the knot. A cusp is contained there; it is the portion of the tubular neighborhood which is in the complement of the knot. Figure 1 shows what a cusp looks like in the hyperbolic metric. One identifies opposite faces of the solid in Figure 1 to create the cusp; note that this identification turns each horizontal cross-section of the solid into a torus which corresponds to the boundary of some tubular neighborhood of the knot. Figure 1 also shows a knot and where the cusp comes from in the complement. The cusp is complete and has finite volume.

From a cusped hyperbolic 3-manifold, we can construct many other manifolds by a process known as Dehn filling. Topologically, we can view the manifold as having a torus boundary at the cusp, and we can attach the boundary of a solid torus to it. There are infinitely many ways to attach the solid torus, and these typically produce different manifolds. This process

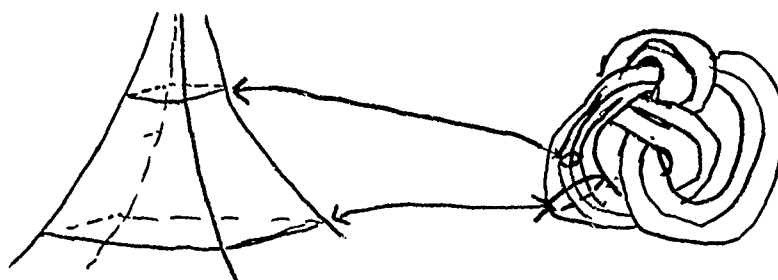


Figure 1

of Dehn filling, when applied to knot or link complements, can produce all oriented closed 3-manifolds. Most knot and link complements are hyperbolic.

We shall construct cusped hyperbolic 3-manifolds by identifying faces on objects known as ideal tetrahedra. We pick 4 points on the boundary of hyperbolic space (i.e. on the sphere at infinity). The edges of an ideal tetrahedron are formed by taking geodesics between all pairs of these 4 points. The faces are taken to be the portion of a geodesic plane inside any ideal triangle formed by 3 edges. It is known that the hyperbolic volume of an ideal tetrahedron is finite and depends only on its dihedral angles, and this volume can be computed from these angles. (This is quite different from our Euclidean sense of volume!) The volume of the manifold can be computed as the sum of the volumes of the ideal tetrahedra from which it is constructed.

For the census, we will want to consider all possible ways to glue faces on a specified number of ideal tetrahedra, although we will want to take measures so that the computer can rule out sets of gluings as being unable to produce cusped hyperbolic 3-manifolds without checking each one individually. We use some properties of cusped hyperbolic 3-manifolds to take these measures.

The definitive reference on hyperbolic 3-manifolds is [Th].

Some Things Already Known About Hyperbolic 3- Manifolds

For hyperbolic 3-manifolds of finite volume, there is a nice fact: for a given topological manifold, there is a unique hyperbolic structure. This fact, known as Mostow's Rigidity Theorem, enables one to obtain topological invariants of the manifold from the hyperbolic structure. For example, the volume is such an invariant.

Jeff Weeks has written a computer program to compute the hyperbolic structures on 3-manifolds. This program starts with a triangulation of the manifold into ideal tetrahedra. There are some equations which describe the gluings around an edge and which ensure that the manifold is complete. The computer solves these equations via a Newton's method algorithm to obtain the hyperbolic structure. The results are stored as dihedral angles of the ideal tetrahedra. [We]

A hyperbolic 3-manifold may have different triangulations although all triangulations will give the same hyperbolic structure, by Mostow's theorem. Thus the triangulation produced by this program will not uniquely describe the manifold. One may obtain from this description several invariants, such as the volume, which depend only on the hyperbolic structure, and computer programs exist to compute them.

Identifying Manifolds

One would like to have a computer program to produce a "canonical triangulation" of the manifold. Because one can triangulate a hyperbolic 3-manifold in more than one way, we need

to distinguish a canonical triangulation. We use the triangulation dual to the Ford domain. (If you don't know what a Ford domain is, don't worry: The key point is the existence of a canonical triangulation.) There is a computer program, known as "canonize," which finds the canonical triangulation. Occasionally special symmetries in a manifold will cause the hoped-for canonical triangulation to be a canonical cell-decomposition which is not a triangulation. In this case, the program arbitrarily triangulates the cells which are not tetrahedra. We then use ad hoc methods to decide when two such ambiguous triangulations represent the same manifold.

We need to consider how to distinguish between triangulations. We shall first describe a method used to store the triangulation. This method is called the terse description. We pick a tetrahedron as the base tetrahedron (and number it 0), and we number the faces from 0 to 3. We number the vertices according to which face they are opposite. We also have a method for numbering the edges from the numbering of the faces. We then consider face 0 of tetrahedron 0. Either it can be glued to a face on a tetrahedron already included in the description (in this case, only tetrahedron 0) or it is glued to a face on a tetrahedron not yet included in the description. In the latter case, we add the tetrahedron to the description (and give it the smallest whole number not already used to number a tetrahedron). We label the faces by taking the numbering on the old tetrahedron and reflecting across the common face. For instance, if we are gluing a new tetrahedron onto face 0 of the old tetrahedron, then face 0 of the old tetrahedron is glued to face 0 of the new tetrahedron while vertices 1, 2, and 3 of the old tetrahedron are glued to vertices 1, 2, and 3 of the new tetrahedron. On the other hand, if we are gluing to a face on a tetrahedron already included in the description, we already have a numbering of the tetrahedron involved. We need to consider how the faces are glued. This can be described by reflecting as above to get the numbering of faces one would get if the tetrahedron we are gluing onto had been new and then finding the permutation of face numberings to get the actual face numberings which were defined when the tetrahedron was added on. For example, if we glue face 0 of tetrahedron 0 to face 1 of tetrahedron 0 and we identify vertices 1, 2, and 3 to vertices 3, 0, and 2, respectively, the permutation desired sends 0 to 1, 1 to 3, 2 to 0, and 3 to 2. We consider faces which have not already been glued until all faces have been glued. The ideas behind the terse description are due to Bill Thurston.

If the triangulation has n tetrahedra, we need $2n$ bits to describe whether we glue onto a new tetrahedron and $n + 1$ entries of the structure consisting of a tetrahedron number and a permutation to describe the gluing for when we glue onto an old tetrahedron. For example, consider this triangulation into 3 tetrahedra:

```
000110
0 2031
2 0321
2 2103
2 2031
```

Note that for computational convenience, each word is read from right to left. We consider first face 0 of tetrahedron 0. The right-hand 0 in the first line tells us that we glue onto an old tetrahedron. So we look at the first line of the information on old tetrahedra. It tells us to glue to face 1 of tetrahedron 0 as described in the example above. Face 1 of tetrahedron 0 has been glued by the time we reach it and thus is not considered. Face 2 of tetrahedron 0 is glued to a new tetrahedron. The new tetrahedron is numbered 1, and face 2 of tetrahedron 1 is glued to face 2 of tetrahedron 0. Face 3 of tetrahedron 0 is glued to a new tetrahedron, tetrahedron number 2. Face 3 of tetrahedron 2 is glued to face 3 of tetrahedron 0. The next face we consider is face 0 of tetrahedron 1, and it is glued to face 1 of tetrahedron 2. We consider faces 1 and 3 of tetrahedron 1 and find that they are glued to faces 0 and 2 of tetrahedron 2, respectively. Then all faces are glued and we have a triangulation.

One thing which still needs to be considered is that we must choose a base tetrahedron and

a numbering of the faces on the tetrahedron in some well-defined manner. Different choices of base tetrahedra and face numberings generally produce different terse descriptions so we would like to pick the base tetrahedron and face numberings in a canonical way. We use edge classes. An edge class of an edge of a tetrahedron in the triangulation is the set of edges glued to that edge (and includes the edge itself.) The size of an edge class is the number of edges it contains. To pick a tetrahedron, we use the sum of the sizes of the edge classes of the edges around a tetrahedron and we find the smallest sum to choose the tetrahedron. We use the sum of squares of the sizes of edge classes to break some ties. We may have a tie at the end, and then we consider each tetrahedron in the tie separately. For each (possibly tied) tetrahedron, we also use the size of the edge classes to find a face numbering of that tetrahedron. For each tetrahedron, we may have a tie of the face numberings. We compute the terse descriptions for each (possibly tied) face numbering of each (possibly tied) tetrahedron considered as the base tetrahedron. We then compare the terse descriptions and pick the one which is lexicographically least. It's potentially a little bit cumbersome (we have to deal with up to $24n$ choices), but it provides a way to pick a well-defined base tetrahedron and face ordering. We often refer to the final terse description as the terse description of the triangulation.

Organizing the Census

We would like to have the computer generate all ways to glue n ideal tetrahedra to form cusped hyperbolic 3-manifolds. There are several steps to organizing the census for n tetrahedra.

The first step is to figure out all possible pairings of faces that keep the triangulation connected. For small values of n , this is easily done by hand, but for $n \geq 5$, this can be a messy process and is best left to a computer. One way to view the face pairings is as a graph on n vertices. Each vertex represents a tetrahedron, and an edge represents a face gluing. We want our graphs to be connected and to have 4 edges leaving each vertex in the graph. One can represent the graph as an n by n symmetric matrix with row and column sums equal to 4. The diagonal elements must be either 0 or 2. We fix the diagonal elements and then generate, via a recursive routine, all possible matrices satisfying that description. We discard matrices representing graphs which are not connected. The problem is that some of the matrices may represent the same graph with the vertices renumbered. We get the computer to consider possible renumberings in a recursive way. Because there are many possible renumberings, we search for ways to eliminate classes of renumberings. If we have a different number of 0's, 1's, 2's, or 3's, we know the matrices can not be renumbered. We also quickly detect partial renumberings which fail and do not continue the recursion. In the end, we have a list of all possible face pairings.

The number of such graphs grows rapidly. For n between 1 and 7, there are 1, 2, 4, 10, 28, 97, and 359 graphs, respectively.

The next stage involves systematically working through all the possibilities of gluing the 2 faces in each pairing. Including orientation reversing gluings, there are 6 such gluings. So for a 5-tetrahedral graph, there are 6^{10} possible gluings to consider. This number is over 60 million, and we would like not to consider all of them directly.

We consider the possibilities for each pairing and continue on to the next pairing provided that some criteria are met. Manifolds which fail to meet the criteria either can not possibly be hyperbolic 3-manifolds of finite volume or can be triangulated with fewer tetrahedra and have already appeared. We use some known properties to speed up the algorithm to produce the census. There are some arguments which enable us to ignore triangulations containing edge classes of low size, namely those with size 1 or 2 and some with size 3. Also, we require that the Euler characteristic of a cross section of each cusp be 0; otherwise, we don't get a torus or Klein bottle cross section as required. This fixes the number of edge classes as equal to the

number of tetrahedra. At a given point partway through the gluing, we can bound the number of edge classes from above and tell if we will have too few edge classes, and we may be able to detect edge classes of small order. So we have some methods of telling partway through the face gluings when some are not going to produce any new cusped hyperbolic 3-manifolds. This saves time, especially when the data structures are chosen carefully. We also keep track of the edge classes as they are built; to save time, we are careful not to reconstruct the entire edge classes when we add one face gluing.

We compute the terse descriptions of triangulations which satisfy the Euler characteristic and edge class criteria and eliminate redundancies.

For $n = 3, 4$, and 5 , the census revealed 31, 224, and 1075 triangulations which may be hyperbolic (i.e. satisfy the Euler characteristic and edge class conditions.)

Searching for Bugs in the Programs

With programs this large and full of various permutations and indices, it's natural that bugs creep in, and we wish to catch them before they do harm. There are various consistency checks which were useful in searching for bugs.

For instance, we could expand a terse description of a triangulation into a full description and then run the program that produces the terse description of a triangulation. We should get back the original terse description. Likewise, we can feed a canonical triangulation of a manifold into "canonize." We should get back the same canonical description. We can also put in different triangulations of the same manifold and check that we get the same canonical triangulation. We can check by hand some of the results of the program to generate graphs. (We did so for $n = 5$.) We had several versions of the program to run the census, and thus we were able to compare results between versions. We had straightforward programs for $n = 3$ and $n = 4$, and this provided a basis for comparison with faster, but more complicated versions.

In the census, the same triangulation (before being fed into "canonize") could not come from different graphs. We checked to make sure this was the case.

We picked an example of two closely related manifolds obtained by the census, and we confirmed that they were different by using methods not involving the census even though they shared many invariants.

There are some known results which we can compare with the census. For instance, Colin Adams and Bill Sherman showed that the smallest number of tetrahedra needed to produce a 3-cusped manifold is 4 and that only 1 such manifold can be obtained from 4 tetrahedra. [AS] Our results agree with that.

Computing Hyperbolic Structures on the Triangulations

The triangulations we computed may be hyperbolic, but we need to compute the hyperbolic structures on them. For each of the manifolds for which we compute a hyperbolic structure we get a terse description of the canonical triangulation of the manifold, and we can again search for duplications. The tricky part is figuring out what to do with the triangulations for which the program did not find hyperbolic structures. Did the Newton's method algorithm fail to converge when it should have? Sometimes, we may get manifolds with volume 0, and we have all flat tetrahedra (of volume 0, but the vertices are at distinct locations on the boundary of hyperbolic space). These we assume are non-hyperbolic. We ignore them. Sometimes, we have degenerate tetrahedra (where at least 2 vertices are at the same place on the boundary of hyperbolic space). These are assumed to represent an incompressible surface which breaks the manifold into 2 distinct manifolds (in effect, we have a composite manifold); we ignore these results. The degenerate vertices often cause infinity to be a solution of the equations used in the Newton's method algorithm, and this wreaks havoc. We observed by hand when this occurred to see that we appeared to get a solution of infinity, and in the end this explained any remaining manifolds where the Newton's method algorithm failed.

We have a complete, non-redundant list of all cusped hyperbolic 3-manifolds obtained from 5 or fewer ideal tetrahedra.

We got 415 manifolds for $n \leq 5$. It was already known that only 1 manifold (the Gieseking) is obtained from the $n = 1$ case and that 4 manifolds are obtained from the $n = 2$ case. Of the 415, 16 first appeared for $n = 3$, 82 first appeared for $n = 4$, and 312 first appeared for $n = 5$. We sorted the list by hyperbolic volume. There are some volumes for which there are multiple manifolds. Also, we did not find an orientable manifold which has volume smaller than the volume of the complement of the figure-8 knot, which is conjectured to have the smallest volume of an orientable cusped hyperbolic 3-manifold.

Of these manifolds, 288 had 1 orientable cusp and no non-orientable cusps, 15 had 2 orientable cusps and no non-orientable cusps, 73 had no orientable cusps and 1 non-orientable cusp, 3 had 1 orientable cusp and 1 non-orientable cusp, 35 had no orientable cusps and 2 non-orientable cusps, and 1 had 1 orientable cusp and 2 non-orientable cusps. Of the manifolds with no non-orientable cusps, 2 of the manifolds with 1 cusp were non-orientable, and the remaining manifolds were orientable.

A preliminary examination of the 415 manifolds obtained in the census suggests a high degree of order. Specifically, it appears that most if not all of the simplest one-cusp manifolds can be obtained as Dehn fillings on four multi-cusp manifolds of volume 3.66386... . Unfortunately, the evidence for such relationships is at present purely circumstantial. We hypothesize specific relationships by comparing the volumes and first homology groups of the one-cusp manifolds with the same invariants for the manifolds obtained by Dehn filling one cusp of a multi-cusp manifold, and also by comparing horoball packings. We will eventually write a program to produce ideal triangulations for the manifolds obtained by Dehn filling a cusp of a multi-cusp manifold, which will allow us to use an existing program which determines whether two cusped hyperbolic 3-manifolds are homeomorphic. At that point we hope to explain most of the manifolds in our census as Dehn fillings on a small number of multi-cusp manifolds.

Summary

We used the computer to produce a census of cusped hyperbolic 3-manifolds obtained from gluing faces on five or fewer ideal tetrahedra. The computer generated all possible gluings of tetrahedra which could produce cusped hyperbolic 3-manifolds. Theoretical considerations reduced the number of possibilities to consider and enabled the computer to find unique identifications of the hyperbolic 3-manifolds. The manifolds obtained in the census are available for further study.

Acknowledgements

Both authors would like to acknowledge the support of the National Science Foundation's Geometry Supercomputer Project, which is located at the University of Minnesota's Supercomputer Institute. We would also like to acknowledge conversations with Bill Thurston, Colin Adams, and others, and we would like to acknowledge the energetic organization of Al Marden in running the Geometry Supercomputer Project.

References

- [Bo], [Mu], and [Sp] provide general background to topology and geometry while the other references are specialized to hyperbolic 3-manifolds.
- [AHW] C. Adams, M. Hildebrand, and J. Weeks. "Hyperbolic Invariants of Knots and Links," to appear in *Transactions of the A.M.S.*
- [AS] C. Adams and W. Sherman. "Minimal Ideal Triangulations of Hyperbolic 3-Manifolds," preprint.

- [Bo] W. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Academic Press, 1986.
- [Mu] J. Munkres. *Topology: A First Course*. Prentice-Hall, 1975.
- [Sp] M. Spivak. *A Comprehensive Introduction to Differential Geometry*. 5 volumes. Publish or Perish, 1979.
- [Th] W. Thurston. "The Geometry and Topology of 3-manifolds", lecture notes, Princeton University, 1978-79.
- [We] J. Weeks. "Hyperbolic Structures on 3-manifolds." Ph.D. dissertation, Princeton University, 1985.

Classicality of Trigonal Curves of Genus Five

Paulo Viana¹
M.I.T. Massachusetts Institute of Technology
and
P.U.C. Pontificia Universidade Católica
do Rio de Janeiro

1 Introduction.

This paper is the description of a computer-assisted search leading to the

Theorem 1 *All trigonal curves of genus five are classical.*

Classical here is to be understood in the sense of F.K.Schmidt [9], that is, the vanishing sequence of the canonical line bundle is the classical sequence 0,1,2,3,4. Non-classical curves are only possible if the characteristic of the field of definition does not exceed $2g - 2$, where g is the genus of the curve, and non-classical curves up to genus 4 were classified by Komiya [7].

For curves of genus 5 it is known that the least integer d for which the curve has a g_d^1 can be only 2, 3 or 4. If $d = 2$ then the curve is hyperelliptic, and F.K.Schmidt showed that these curves are classical (Satz 8 [9]). We deal here with the case $d = 3$.

The phenomenon of non-classicality is interesting for a number of reasons. There are several instances where non-classical curves are the examples to be avoided to be able to carry over a characteristic 0 reasoning to the prime characteristic case. Moreover, the rare non-classical curves are usually interesting from the point of view of other prime characteristic phenomena such as: non-reflexivity (see [4]), large automorphism groups (see [10]), supersingularity (see [11]), etc.

There are non-classical trigonal curves in other genera; besides all the examples in low genus studied in [7] and some of the examples in [9] we mention the following family of non-classical curves of genus 9, in characteristic 3:

$$y^3 + y = x^{10} + \lambda_7 x^7 + \lambda_6 x^6 + \lambda_4 x^4 + \lambda_3 x^3 + (\lambda_4 \lambda_7^2 + \lambda_6^3)^{1/9} x^2 + \lambda_1 x.$$

¹Supported by CNPq - Brazilian National Research Council, grant 200625/87.

2 Geometrical Considerations.

Let C be a (smooth, irreducible) trigonal curve of genus 5 defined over an algebraically closed field k . It is classically known that there is a unique linear system of type g_3^1 , and that if D is an effective divisor in this g_3^1 then $2D$ is special while $3D$ is not, so that we can fix a canonical divisor K such that $K - D$ is effective. We have $\deg(K - D) = 5$, and by the Theorem of Riemann-Roch $h^0(\mathcal{O}(K - D)) = 3$, and thus $K - D$ is in a g_5^2 . Using the embedding given by $|K - D|$ we view C as a plane quintic, the genus formula for plane curves showing that we must have an ordinary node or an ordinary cusp P , and the g_3^1 is cut out by lines passing through P . Using projective transformations we can suppose $P = (0 : 0 : 1)$, and thus we have described C as $F = 0$, where

$$F(X_0, X_1, X_2) = \sum_{i+j \leq 5} a_{ij} X_0^{5-(i+j)} X_1^i X_2^j \quad (1)$$

and $a_{05} = a_{04} = a_{14} = 0$. We can also choose another point of C to be $(1 : 0 : 0)$, so as to have $a_{00} = 0$.

If L is a tangent of C at P then the intersection number $I(P, C \cap L)$ of C and L at P is at least 3. We distinguish two cases:

1. There is a tangent L such that $I(P, C \cap L) < 5$. In this case C meets L in another point Q , and using projective transformations we can assume $Q = (0 : 1 : 0)$, and so L is the line $X_0 = 0$. These conditions imply that $a_{23} = a_{50} = 0$.
2. For every tangent L we have $I(P, C \cap L) = 5$. We can still choose $X_0 = 0$ as a tangent, so as to have $a_{23} = 0$, and the condition for the intersection number implies $a_{32} = a_{41} = 0$.

A system of adjoint curves is given by conics passing through P , and so there is a canonical morphism of the form $(X_0^2 : X_0 X_1 : X_1^2 : X_0 X_2 : X_1 X_2)$, or, if $x := X_1/X_0$ and $y := X_2/X_0$, $(1 : x : x^2 : y : xy)$. Clearly the function field $k(C)$ is given by $k(x, y)$.

3 Arithmetical Considerations.

The ramification index e_Q of a point Q over $k(x)$ can be 1, 2 or 3, and if the characteristic p of k is not 3 then we have, by the Riemann-Hurwitz formula,

$$\sum_Q e_Q - 1 = \deg \mathcal{D}_{k(C)|k(x)} = 14, \quad (2)$$

where $\mathcal{D}_{k(C)|k(x)}$ denotes the different of $k(C)$ over $k(x)$. The point Q is unramified (resp. of ordinary ramification, resp. of total ramification) if $e_Q = 1$ (resp. $e_Q = 2$, resp. $e_Q = 3$). The possible order sequences at a ramification point over $k(x)$ were

determined by Coppens [1, 2] and Kato [5] for the case of characteristic 0, but it is easily checked that their proofs remain valid in prime characteristics to yield the following: the order sequence at a total ramification point can be only 0,1,3,4,6, in which case the point is said to be of type I, or 0,1,3,4,7, and then the point is said to be of type II. The order sequence at an ordinary ramification point can be only the classical sequence 0,1,2,3,4, in which case the point is said to be of type I, or 0,1,2,3,5, and then the point is said to be of type II. We conclude that the existence of an ordinary ramification point of type I implies the classicality of the curve; moreover, we use the Corollary 1.7 in [12] to conclude that the existence of a total ramification point of type I implies the classicality of the curve in characteristics other than 3 and 5, that the existence of a total ramification point of type II implies the classicality of the curve in characteristics other than 2, 3 and 7, and that the existence of an ordinary ramification point of type II implies the classicality of the curve in characteristics other than 5.

On the other hand, if $p \neq 3$ then we can use Corollary 6.2 in [6], checking that its proof remains valid in characteristics other than 3, to conclude that if the extension field $k(C)|k(x)$ is non-cyclic then there is an ordinary ramification point of type I, and therefore the curve is classic. So we can have nonclassicality only if $k(C)|k(x)$ is cyclic, and in this case all the ramification points are total; by the Riemann-Hurwitz formula we have 7 ramification points altogether. We can now use Theorem 2.1 and Proposition 3.3 in [1], again checking that the arguments there are characteristic free, to conclude that we have exactly 5 total ramification points of type I and 2 of type II. In the presence of these points the only characteristic left for nonclassicality is 3.

4 Computational Considerations.

As the field extension $k(C)|k(x)$ must be separable we may take x as a separating variable, and the curve C will be classical if and only if the Wronskian determinant with respect to the classical order sequence (see [12]) is not identically zero, ie, if and only if

$$W := \det(D_x^{(i)}(f_j))_{i,j=0,\dots,4} \neq 0, \quad (3)$$

where f_j are the functions $1, x, x^2, y, xy$ given by the embedding and $D_x^{(i)}(f_j)$ denotes the i^{th} Hasse-Schmidt derivative of f_j (see [3]). These derivatives are used to avoid the vanishing of higher order terms in prime characteristics; they are defined in $k[x]$ by

$$D_x^{(i)}(\sum c_l x^l) = \sum \binom{l}{i} c_l x^{l-i},$$

and extend uniquely to $k(x)$ and to $k(x, y)$. They satisfy the product rule

$$D_x^{(i)}(gh) = \sum_{j=0}^i D_x^{(j)}(g) D_x^{(i-j)}(h),$$

and the iteration rule

$$\binom{j}{i} D_x^{(j)}(g) = D_x^{(j-i)}(D_x^{(i)}(g)),$$

and these rules can be used to compute the derivatives implicitly from equation (1). This process, however, is slow.

As an alternative way to compute $D_x^{(l)}(y)$, for which we are indebted to Karl-Otto Stöhr, we use the the *generalized Taylor expansions* (see [8]) of x and y :

$$\begin{aligned} Tx &= x + t \\ Ty &= \sum_{l=0}^{\infty} D_x^{(l)}(y) t^l. \end{aligned} \quad (4)$$

Taking these expansions in equation (1) we have $F(1, Tx, Ty) = 0$, and the condition for the vanishing of the coefficient of t^l gives $D_x^{(l)}(y)$ as a rational function in x, y and $D_x^{(m)}(y)$ for $m < l$, and so we can obtain these derivatives recursively. This process is made easier by the following general lemma:

Lemma 1 *Let K be an algebraic function field in one variable, and suppose that $K = k(x, y)$, where x and y satisfy a polynomial relation*

$$f(x, y) = \sum_{i,j} a_{ij} x^i y^j, \quad (1)$$

and that $K|k(x)$ is a separable field extension. The Hasse-Schmidt derivatives can be recursively expressed as rational functions

$$D_x^{(l)}(y) = n_l / d_l,$$

where $n_l \in k[x, D_x^{(m)}(y), m < l]$ and $d_l = d \in k[x, y]^*$ for all l (that is, d_l is in fact independent of l).

Proof We take the expansions (4) in the equation (1) to get

$$\sum_{i,j} a_{ij} \sum_{n=0}^i \binom{i}{n} x^n t^{i-n} \sum_l \sum_{i_1+\dots+i_j=l} \prod_{r=1}^j D_x^{(i_r)}(y) t^{i_r} = 0.$$

We view the left-hand side of this equation in $k[x, D_x^{(m)}(y)][[t]]$, and observe that the first occurrence of the derivative $D_x^{(l)}(y)$ is in the coefficient of t^l , corresponding to the term with $n = i$, $i_r = l$ for some $r = 1, \dots, j$, and $i_s = 0$ for $s \neq r$. Thus the vanishing of the coefficient of t^l can be expressed as

$$D_x^{(l)}(y) \sum_{i,j} j a_{ij} x^i y^{j-1} - n_l = 0,$$

where $n_l \in k[x, D_x^{(m)}(y), m < l]$; set

$$d := \sum_{i,j} j a_{ij} x^i y^{j-1}.$$

If d vanishes then we have $a_{ij} \neq 0$ only if $p|j$, which contradicts the separability condition, and so we have proved the lemma.

Using the lemma we compute the Hasse-Schmidt derivatives $D_x^{(l)}(y)$ and substitute them into (3). Observe that using properties of determinants we have

$$W = [D_x^{(3)}(y)]^2 - D_x^{(2)}(y) \cdot D_x^{(4)}(y).$$

Because of the lemma we know that $d \neq 0$, and that for some integer n we have that $d^n W$ is a polynomial in x and y ; by inspection we see that in our case $n = 8$, and so we are reduced to check the vanishing of $d^8 W \in k[x, y]$, and for that we use all the simplifications in section 1. From what we have seen in section 2 we can suppose that $p = 3$.

As the polynomial expression $d^8 W$ is too large to be conveniently dealt with, we have done separately the computation of several of its subexpressions, eg, terms in $d^8 W$ of the form x^n , etc. Theorem 1 is obtained by checking that in all cases the condition of vanishing of these subexpressions implies some decreasing of the genus. To illustrate this procedure we describe it in detail for the second case in section 1, that is, when for any tangent L at P the intersection number of C and L at P is 5, the treatment of the other case being similar.

We start by observing that in this case P can not be a cusp, because if it were then the condition on the intersection number would imply that P is in fact a higher-order cusp, and the genus would drop. We then have an ordinary node, and as we can suppose that $X_0 = 0$ is a tangent at P we must necessarily have $a_{13} \neq 0$. Notice that as $\deg(f) = 5$ we must have $a_{50} \neq 0$. We then have

$$d^8 W = a_{02}^2 a_{03}^2 a_{13}^6 y^{20} + \dots + (a_{01}^2 a_{03}^2 a_{13}^6 + a_{02}^4 a_{13}^6 + a_{02} a_{03}(\dots)) y^{18} + \dots.$$

If $d^8 W$ is to vanish identically we must have $a_{02} a_{03} = 0$, but if $a_{02} \neq 0$ then $a_{03} = 0$ and $d^8 W = a_{02}^4 a_{13}^6 y^{18} + \dots$, and the curve would be classical, so we have $a_{02} = 0$, and

$$d^8 W = a_{01}^2 a_{03}^2 a_{13}^6 y^{18} + \dots,$$

and so $a_{01} a_{03} = 0$. We cannot have $a_{01} = a_{03} = 0$ for otherwise $x|f(x, y)$ and the curve would be reducible. If $a_{03} = 0$ then $a_{01} \neq 0$ and

$$d^8 W = a_{01}^4 a_{12} a_{13}^5 y^{13} - (a_{01}^4 a_{11} a_{13}^5 + a_{01}^4 a_{12}^2 a_{13}^4) y^{12} + a_{01}^5 a_{13}^4 a_{22} y^{11} + \dots,$$

and so $a_{12} = a_{11} = a_{22} = 0$, and proceeding in this way we see that the vanishing of $d^8 W$ implies the vanishing of a_{21}, a_{20}, a_{31} and a_{30} . We are left with

$$d^8 W = a_{01}^2 a_{13}^2 a_{50}^6 x^{26} + \dots$$

and the curve is classical. The other case, when $a_{03} \neq 0 = a_{01}$, is treated similarly.

The strategy actually used for dealing with $d^8 W$ follows very close the above description: critical terms were selected in the expression of $d^8 W$ and these terms

were computed one at a time, using at each point the simplifications obtained by the conditions of vanishing of previously computed terms. The computation of these terms, some of which were quite long, were done using the MACSYMA program at the Sun Workstations of the Department of Mathematics of M.I.T.

The author did this work while enjoying a period as a visitor at M.I.T., and he is happy to have this opportunity to thank professor Steven Kleiman for the invitation.

References

- [1] M.Coppens *The Weierstrass gap sequences of the total ramification points of trigonal coverings of P^1* , Indag. Mat. 47 (1985) 245-270.
- [2] M.Coppens *The Weierstrass gap sequences of the ordinary ramification points of trigonal coverings of P^1 . The existence of a kind of Weierstrass gap sequence*, J.Pure Appl. Algebra, 43 (1986) 11-25.
- [3] H.Hasse-F.K.Schmidt *Noch eine Begründung der höheren Differentialquotienten in einen algebraischen Funktionenkörper einer Unbestimmten*, J.Reine Angew. Math., 177(1937) 215-237.
- [4] M.Homma *Reflexivity of tangent varieties associated with a curve*, preprint.
- [5] T.Kato *On Weierstrass points whose first non-gaps are three*, J.Reine Angew. Math., 316 (1980) 99-109.
- [6] T.Kato-Horiuchi *Weierstrass gap sequences at the ramification points of trigonal Riemann surfaces*, J.Pure Appl. Algebra, 50 (1988) 271-285.
- [7] K.Komiya *Algebraic curves with non-classical types of gap sequences for genus three and four*, Hiroshima Math. J., 8 (1978) 371-400.
- [8] F.K.Schmidt *Die Wronskische Determinante in beliebigen differenzierbaren Funktionen Körpern*, Math. Z., 45(1939) 62-74.
- [9] F.K.Schmidt *Zur arithmetischen Theorie der algebraischen Funktionen II. Allgemeine Theorie der Weierstrasspunkte*, Math. Z., 45 (1939) 75-96.
- [10] H.Stichtenoth *Über die Automorphismengruppe eines algebraischen Funktionenkörpers von Primzahlcharakteristik. Teil II Ein spezieller Typ von Funktionenkörpern*, Arch. Math., 24 (1973) 615-631.
- [11] K.O.Stöhr-P.Viana *A Study of Hasse-Witt matrices by local methods*, to appear in Math.Z.
- [12] K.O.Stöhr-J.F.Voloch *Weierstrass points and curves over finite fields*, Proc. London Math. Soc., 52 (1986) 1-19.

Symmetric Matrices with Alternating Blocks

ABRAMO HEFEZ

Univ. Fed. do Esp. Santo,
Depart. de Matematica,
Vitoria - ES, Brazil.

and

ANDERS THORUP

Matematisk Institut,
Københavns Universitet,
Copenhagen, Denmark.

A statement in algebraic geometry over fields of arbitrary characteristic follows from the existence of matrices with integer entries of the type mentioned in the title. It is shown how these matrices can be built from a finite number of small matrices. It is reported how these small matrices, of which the largest is a 25 by 25 matrix, were found using computer algebra systems.

INTRODUCTION

The matrices of the title are block matrices of the following form:

$$M = \begin{pmatrix} 0 & M_{12} & \cdots & M_{1r} \\ -M_{12} & 0 & \cdots & M_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ -M_{1r} & -M_{2r} & \cdots & 0 \end{pmatrix},$$

where each block $M_{i,j}$ is an alternating $s \times s$ matrix. Matrices of the above form are symmetric matrices of size equal to rs . They occur naturally as local Hessians of the Plücker embedding of the Grassmannian into projective space.

The Grassmannian and its dual variety were studied by Hefez and Thorup in [H-T-87]. For their duality result they needed the rank of the general matrix of the above form with values in an arbitrary field. To find the general rank, they construct certain matrices with integer entries. More precisely, when both r and s are greater than 2, they construct a matrix of the above form whose determinant is equal to ± 1 if rs is even and equal to ± 2 if rs is odd.

The object of the present note is to report on their construction with the emphasis on its computational aspects. Section 1 describes the geometric background of the problem. Section 2 treats matrices of the above form with entries in an arbitrary field and constructs matrices with integer entries out of simpler matrices of small sizes. The necessary small matrices, of which the largest is a 25×25 matrix with $r = s = 5$, are listed in Section 3. These matrices were found on a computer.

1. GEOMETRIC BACKGROUND

Let X be a subvariety of projective N -space $\mathbb{P}^N$. The dual variety X' is the set of tangent hyperplanes to X , considered as a subvariety of the dual projective space $\mathbb{P}^N$. The concept of duality is treated in the extensive survey of Kleiman [K-86]. The variety is called *reflexive* when (omitting the details) the double dual, X'' , equals X . It is called *ordinary* if, moreover, the dual variety, X' , is a hypersurface in $\mathbb{P}^N$. In characteristic zero every variety is reflexive.

The above notions were studied by Hefez and Kleiman [H-K-85] using local Hessians. With every point x of X there is associated a local Hessian which is a certain symmetric matrix of size equal to the dimension of X . They show that X is ordinary iff the local Hessian generically is of corank equal to 0, i.e., is regular. Following their terminology, the variety X is called *semi-ordinary* if the local Hessian generically is of corank equal to 1.

The case where X is the Grassmannian of r -dimensional subspaces ($r \geq 2$) in a vectorspace of dimension $r + s$ ($s \geq 2$), embedded as usual in $\mathbb{P}^N$ with $N = \binom{r+s}{s} - 1$, is studied by Hefez and Thorup [H-T-87]. They prove the following result:

(a) The dual of the Grassmannian is a hypersurface in $\mathbb{P}^N$ except in the following cases: One of r, s is equal to 2 and the other is odd. In the exceptional cases, the dual has codimension 3 in $\mathbb{P}^N$.

(b) The Grassmannian is reflexive except in the following cases: The characteristic of the field is 2, and both r and s are odd. In the exceptional cases, the Grassmannian is semi-ordinary.

(c) The Grassmannian is ordinary iff none of the exceptional cases in (a) or (b) occur.

Part (a) of the result for $n = 1$ and in characteristic zero is partly contained in Griffiths and Harris [G-H-79, (3.10) p.396] and partly in Mumford [M-78]. The full result in characteristic zero is proved independently by Knop and Menzel in [K-M-87].

The proof of the result involves an analysis of how the dimension of the dual variety varies with the characteristic of the groundfield, and a consideration of the local Hessians of the Grassmannian. The latter Hessians are block matrices of the form mentioned in the introduction. The consideration of the Hessians is based on the following fact: The general matrix of the form mentioned in the introduction with entries in a given field has corank equal to 0 except for the exceptional cases listed in the result above. In the exceptional cases of (a), the general matrix is of corank equal to 2; in the exceptional cases of (b), it is of corank equal to 1.

2. MATRICES OF TYPE (r, s)

A matrix M with entries in a given field will be called a *matrix of type (r, s)* if it is of the following form:

$$M = \begin{pmatrix} 0 & M_{12} & \cdots & M_{1r} \\ -M_{12} & 0 & \cdots & M_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ -M_{1r} & -M_{2r} & \cdots & 0 \end{pmatrix},$$

where each block $M_{i,j}$ is an alternating $s \times s$ matrix.

The following is a series of elementary observations about matrices of type (r, s) .

(2.1). A matrix of type (r, s) is a symmetric matrix of dimension rs with zeroes in the diagonal.

(2.2). There is an obvious permutation of the entries of a matrix M of type (r, s) which transforms it into a matrix of type (s, r) without changing its determinant.

(2.3). The only matrix of type (r, s) with r or s equal to 1 is the zero matrix.

(2.4). A matrix M of type $(2, s)$ has the form,

$$M = \begin{pmatrix} 0 & A \\ -A & 0 \end{pmatrix},$$

where A is an alternating $s \times s$ matrix. Hence $\det M = (\det A)^2$. If s is even, then an obvious choice of A gives a matrix M with $\det M = 1$ and, therefore, the generic corank is equal to 0. Assume that s is odd. Then, as is wellknown, an alternating $s \times s$ matrix is singular. Therefore, the corank of M is at least equal to 2. Again, an obvious choice of A gives a matrix M of corank 2. Hence, the generic corank of M is equal to 2 if s is odd.

(2.5). Assume that the characteristic of the field is 2. Then a matrix M of type (r, s) is alternating. In particular, $\det M = 0$ if rs is odd.

LEMMA (2.6). *Let r and s be integers greater than 2. Then there exists a matrix N with integer entries and of type (r, s) such that*

$$\det N = \begin{cases} \pm 1 & \text{if } rs \text{ is even,} \\ \pm 2 & \text{if } rs \text{ is odd.} \end{cases}$$

PROOF: For positive integers r and s , denote by $\nu(r, s)$ the minimum value of exponents v such that for some matrix M with integer entries and of type (r, s) , the determinant of M is equal to $\pm 2^v$. (Set $\nu(r, s) := \infty$ if no such exponent exists.) Clearly, the Lemma asserts that if r and s are greater than 2, then $\nu(r, s) = 0$ if rs is even and $\nu(r, s) = 1$ if rs is odd.

First observe that the following relations hold:

- (1) $\nu(r, s) = \nu(s, r)$.
- (2) $\nu(r_1 + r_2, s) \leq \nu(r_1, s) + \nu(r_2, s)$.
- (3) $\nu(r, s) = 0$, if r and s are even.
- (4) $\nu(r, s) > 0$, if r and s are odd.

Indeed, (1) follows from (2.2), (2) follows by observing that a direct sum of matrices of types (r_1, s) and (r_2, s) is a matrix of type $(r_1 + r_2, s)$, (3) follows from (1) and (2) using the obvious value $\nu(2, 2) = 0$, and (4) follows from (2.5) reducing the matrix modulo 2.

Next, the matrices listed in Section 3 give the values

$$\nu(3, 4) = \nu(6, 3) = 0,$$

and, using (4),

$$\nu(3, 3) = \nu(5, 3) = \nu(5, 5) = 1.$$

Moreover, using the above values and (2) and (3), it follows that

$$\nu(5, 4) \leq \nu(2, 4) + \nu(3, 4) = 0$$

and

$$\nu(5, 6) \leq \nu(2, 6) + \nu(3, 6) = 0.$$

In particular, from (1), (2), (3) and the above values it follows that $\nu(r, 4n) = 0$ if r is even or equal to 3 or 5.

Finally, write each of r and s in the form $t + 4n$, where t is equal to 3, 4, 5 or 6 and n is non-negative. The conclusion follows immediately from the above results.

PROPOSITION (2.7). Let r and s be integers greater than 1. Then the general matrix M of type (r, s) with entries in a field is of corank equal to 0 except in the following cases:

- (a) One of r, s is equal to 2 and the other is odd. In this case M is of corank equal to 2.
- (b) The characteristic of the field is 2, and both r and s are odd. In this case M is of corank equal to 1.

PROOF: The assertion holds if r or s is equal to 2 by (2.2) and (2.4). Hence for the rest of the proof we may assume that r and s are greater than 2.

Let N be a matrix of type (r, s) with the property asserted in Lemma (2.6), and consider its reduction, $\bar{N}$, modulo the characteristic of the field. Then $\bar{N}$ is regular, that is, of corank equal to 0, except when the characteristic is 2 and rs is odd. Therefore, M is of corank equal to 0 except in case (b).

Thus it remains to prove in case (b) that the general matrix M of type (r, s) is of corank equal to 1. By (2.5), the corank is at least equal to 1. Therefore, it suffices to prove that the matrix $\bar{N}$ is of corank equal to 1. Let $d := rs$ be the dimension of the matrix N , and consider the matrix adjugate to N . It is a $d \times d$ matrix whose entries up to sign are the $(d-1) \times (d-1)$ minors of N . Since N has determinant equal to ± 2 , the adjugate has determinant equal to $\pm 2^{d-1}$. It follows that at least one entry in the adjugate has to be odd. Thus some $(d-1) \times (d-1)$ minor of N is odd, and therefore, the reduction $\bar{N}$ has rank $d-1$, that is, $\bar{N}$ is of corank equal to 1. Hence the Proposition is proved.

3. SMALL MATRICES OF TYPE (r, s)

This section lists matrices of types (2,2), (3,3), (3,4), (5,3), (6,3) and (5,5) with determinants as required in the proof of Lemma (2.6). Except for the first (trivial) matrix in the list, they were computed on the IBM 4341 at the University of Copenhagen as follows:

A matrix M of type (3,3) is given by 3 blocks A, B, C . As each block is an alternating 3×3 matrix, it may be identified with a triple of coordinates. Therefore, the determinant $\det M$ is a polynomial in 9 variables. It is homogeneous of degree 9. This polynomial was computed and factored by REDUCE (Version 3.0). REDUCE found the following identity:

$$\det \begin{pmatrix} 0 & A & B \\ -A & 0 & C \\ -B & -C & 0 \end{pmatrix} = -2 \det(A, B, C)^3.$$

In the latter identity A, B, C are alternating 3×3 matrices. On the left hand side they are blocks in a 9×9 matrix. On the right hand side each of A, B and C is considered as a column (in a 3×3 matrix) by taking its 3 entries above the diagonal.

An obvious choice of the 3×3 matrix on the right hand side of the above identity produced the matrix of type (3,3) in the list below.

The higher values of r, s were handled differently. A small interactive program was written using MATLAB to compute determinants. The entries of the matrices

were varied in small loops until the desired determinant was obtained. In the case of matrices of type (5,5) the number of variable entries is equal to 100.

The list of matrices found is the following:

Type of matrix M is (r,s)=(2,2).
Size is 4x4. det(M)=1.

```
0 0 0 1
0 0 -1 0
0 -1 0 0
1 0 0 0
```

Type of matrix M is (r,s)=(3,3).
Size is 9x9. det(M)=2.

```
0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0
0 0 0 0 -1 0 -1 0 0
0 0 0 0 0 0 0 1 0
0 0 -1 0 0 0 -1 0 0
0 1 0 0 0 0 0 0 0
0 0 -1 0 -1 0 0 0 0
0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 0
```

Type of matrix M is (r,s)=(3,4). Size is 12x12. det(M)=1.

```
0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 -1 0 0 0 0 0 -1 0
0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 -1 0 0 0
0 -1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 -1 0
0 0 0 -1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0
0 -1 0 0 0 0 0 -1 0 0 0 0
1 0 0 0 0 0 1 0 0 0 0 0
```

Type of matrix M is (r,s)=(5,3). Size is 15x15. det(M)= 2.

```
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
0 0 0 0 0 1 0 0 0 0 0 0 -1 0 0
0 0 0 0 -1 0 0 0 0 -1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0 -1 0 0 0 -1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 -1 0 0
0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0
0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0
1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 -1 0 0 0 -1 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
```

Type of matrix M is (r,s)=(6,3). Size is 18x18. det(M)= -1.

```

0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 -1 0 0 0 0 -1
0 0 0 0 -1 0 0 0 0 0 -1 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0
0 0 -1 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 -1 0 -1 0 0 0 0 0
0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 -1 0 0
0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 -1 0 0 -1 0 0 0 0 0 0 0 -1 0 0
1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 -1 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 -1 0 -1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 -1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

```

Type of matrix M is (r,s)=(5,5). Size is 25x25. det(M)= 2.

```

0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1
0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0
0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0
0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 1 0 0
0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0
1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 -1 0 0
0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 -1 0 0 0 0 0 0
1 0 0 0 0 0 0 1 0 0 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 0 0

```

REFERENCES

- [G-H-79] Ph. Griffiths and J. Harris, *Algebraic geometry and local differential geometry*, Ann. Sci. École Norm. Sup. 12 (1979), 355-432.
- [H-K-85] A. Hefez and S. Kleiman, *Notes on the duality of projective varieties*, in "Geometry Today (Giornate di Geometria, Rome 1984)," (eds.) E. Arbarello, C. Procesi, E. Strickland, Birkhäuser, Boston, 1985, pp. 143-183.
- [H-T-87] A. Hefez and A. Thorup, *Reflexivity of Grassmannians and Segre varieties*, Communications in Algebra 15,6 (1987), 1095-1108.
- [K-86] S. Kleiman, *Tangency and Duality*, in "Proc. 1984 Vancouver Conf. in Algebraic Geometry," (eds.) J. Carrell, A. V. Geramita, P. Russell, CMS Conf. Proc. 6, Amer. Math. Soc., 1986, pp. 163-226.
- [K-M-87] F. Knop and G. Menzel, *Duale Varietäten von Fahnenvarietäten*, Comment. Math. Helv. 62,1 (1987), 38-61.
- [M-78] D. Mumford, *Some Footnotes to the Work of C.P. Ramanujan*, in "C. P. Ramanujan - A Tribute," Tata Studies in Math., Springer, 1978, pp. 247-262.

COHOMOLOGY TO COMPUTE

D. Leites* (Stockholm University, Department of Mathematics,
Box 6701, 11385 Stockholm, Sweden)

G. Post (University of Twente, Department of Applied Mathematics,
Box 217, 7553 EA Enschede, The Netherlands)

Introduction

Our purpose is to interest people to calculate (co)homology with help of a computer, in particular, (co)homology of Lie algebras and Lie superalgebras.

Homology is easy to explain. Given a linear space V and $d \in \text{End}(V)$ such that $d^2 = 0$, we call $H(d) = \ker(d)/\text{im}(d)$ the homology of d . Here $\ker(d) = \{v \in V \mid d(v) = 0\}$ and $\text{im}(d) = \{v \in V \mid v = d(w) \text{ for some } w \in V\}$. Cohomology is a special case of homology.

Here, we give an introduction in Lie algebras and the basics of cohomology of Lie algebras. A method to implement d is described. Finally we give a list of open problems. Probably not all of them are suitable to tackle by computer. However, having computed partial results, this might give ground for a reasonable hypothesis.

Throughout we refer to Fuks [1986].

*D.L. acknowledges the financial support of a Bendixson grant, Swedish N.R.F. and, partially, M.P.I. Bonn.

1. Lie algebras

For sake of completeness, we start with a definition of a Lie algebra.

A Lie algebra $\mathfrak{g}$ over a field k is a vector space over k , equipped with a bilinear map $[\cdot, \cdot]: \mathfrak{g} \rightarrow \mathfrak{g}$ (called Lie product or bracket) such that

- 1) $[x, x] = 0 \quad (\forall x \in \mathfrak{g})$
- 2) $[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0.$
 $(\forall x, y, z \in \mathfrak{g}).$

From property 1) one easily derives $[x, y] = -[y, x]$, by considering $[x+y, x+y] = 0$.

If $\mathfrak{g}$ and $\mathfrak{h}$ are Lie algebras over k , and $\rho: \mathfrak{g} \rightarrow \mathfrak{h}$ is a linear map, then ρ is called a morphism of Lie algebras if for all $x, y \in \mathfrak{g}$:

$$\rho([x, y]) = [\rho(x), \rho(y)]$$

The standard example of a Lie algebra is the following.

Let $\text{End}(V)$ be the space of linear maps from V to V , where V is a vector space over k . $\text{End}(V)$ has a natural product, i.e. composition of two endo-morphisms. For $[\cdot, \cdot]$ we take the commutator, so

$$[x, y] = x \circ y - y \circ x$$

$\text{End}(V)$ with this bracket becomes a Lie algebra, denoted by $\mathfrak{gl}(V)$.

A morphism $\rho: \mathfrak{g} \rightarrow \mathfrak{gl}(V)$ (for some V) is called a representation of $\mathfrak{g}$.

It gives a realization of $\mathfrak{g}$ by linear operators on V , or even as matrices if $\dim(V) < \infty$.

A different terminology is simultaneously in use.

If $\rho: \mathfrak{g} \rightarrow \mathfrak{gl}(V)$ is a representation, then V is called a $\mathfrak{g}$ -module. The action $x \cdot v$ ($x \in \mathfrak{g}$, $v \in V$) is defined by:

$$x \cdot v = \rho(x)v,$$

and is linear in x and v . Moreover it satisfies

$$[x, y] \cdot v = (x \circ y)(v) - (y \circ x)(v)$$

The Lie algebras and its modules that we encounter are often graded. Suppose Λ is an additive subgroup of k . Then g (or V) is Λ -graded means that we have direct sum decompositions in linear subspaces

$$g = \bigoplus_{k \in \Lambda} g_{(k)} ; \quad V = \bigoplus_{k \in \Lambda} V_{(k)}$$

such that

$$[g_k, g_l] \in g_{(k+l)} \quad (g_k \in g_k, g_l \in g_l)$$

and

$$g_k \cdot v_l \in V_{(k+l)} \quad (g_k \in g_k, v_l \in V_{(l)})$$

Gradings play an important role in (co)homology.

2. Definition of cohomology of Lie algebras

Let g be a Lie algebra over the field k , and A a g -module. We denote the linear space of q -linear, skew-symmetric maps $c: g \times \dots \times g \rightarrow A$ by $C^q(g; A)$, and put $C^*(g; A) = \bigoplus_{q \in \mathbb{Z}} C^q(g; A)$; $C^q(g; A) = 0$ for $q < 0$.

Define $d_q: C^q(g; A) \rightarrow C^{q+1}(g; A)$ by

$$\begin{aligned} (q \geq 0) \quad d_q c(g_1, \dots, g_{q+1}) &= \sum_{1 \leq s < t \leq q+1} (-1)^{s+t-1} c([g_s, g_t], g_1, \dots, \hat{g}_s, \dots, \hat{g}_t, \dots, g_{q+1}) \\ &\quad + \sum_{1 \leq s < t \leq q+1} (-1)^s g_s \cdot (c(g_1, \dots, \hat{g}_s, \dots, g_{q+1})) \end{aligned}$$

$$(q < 0) \quad d_q = 0$$

Here $\hat{}$ means, that the element under it is deleted.

From these d_q , one builds $d: C^*(g; A) \rightarrow C^*(g; A)$ by setting $d(c) = d_q(c)$, if $c \in C^q$ and extending it by linearity.

One can check that $\text{dod} = d^2 = 0$. Hence $\text{im}(d) \subset \ker(d)$. This leads to the following definition:

Definition $H^*(g; A) = \ker(d)/\text{im}(d)$;

$$H^q(g; A) = \ker(d_q)/\text{im}(d_{q-1}).$$

One easily checks that $H^*(g; A) = \bigoplus_{q \geq 0} H^q(g; A)$.

If $\psi \in C^*(g; A) \cap \ker(d)$, then its cohomology class is denoted by $[\psi]$. A special, important case is $A = k$, and $g.k = 0$ ($\forall g \in g, k \in k$), i.e. the trivial representation. In this case, the last term in the definition of d_q disappears. We write $H^q(g)$ instead of $H^q(g; k)$. Now suppose that g and A are graded. We denote by $C_{(\lambda)}^q(g; A)$ those elements of $C^q(g; A)$ that satisfy

$$c(g_1, \dots, g_q) \in A_{(\lambda - \sum \lambda_i)} \quad \text{if } g_i \in g_{(\lambda_i)}.$$

One can easily check that $C^q(g; A) = \prod_{(\lambda)} C_{(\lambda)}^q(g; A)$ and, putting $C_{(\lambda)}^*(g; A) = \bigoplus_{q \in \mathbb{Z}} C_{(\lambda)}^q(g; A)$, that $d C_{(\lambda)}^*(g; A) \subset C_{(\lambda)}^*(g; A)$. Thus $C_{(\lambda)}^*(g; A)$ constitutes a subcomplex, and we can define $H_{(\lambda)}^*(g; A)$ in the obvious way.

This describes the cohomology $H^*(g; A)$ completely, since we clearly have

$$H^*(g; A) = \prod_{\lambda \in \Lambda} H_{(\lambda)}^*(g; A).$$

In case of the trivial module $A = k$, we set $A = A_{(0)}$.

If g and A are finite-dimensional, we find $H^* = \bigoplus_{(\lambda)} H_{(\lambda)}^*$.

In case g and A are infinite-dimensional, there is often some topology given, and $C^q(g, A)$ is required to consist of continuous mappings.

In this case we find $H^* \subset \prod_{(\lambda)} H_{(\lambda)}^*$.

Theorem (Fuks [1986], Th. 1.5.2b) Suppose there exists a $g_0 \in g$ such that $[g_0, g_\lambda] = \lambda g_\lambda$ and $g_0 \cdot a_\mu = \mu \cdot a_\mu$.
 $(g_\lambda \in g_{(\lambda)}, a_\mu \in A_{(\mu)})$.
 Then $H_{(\lambda)}^* = 0$ for $\lambda \neq 0$.

This, of course, reduces the work to be done greatly.

3. Applications of cohomology

We list some applications of cohomology.

Example 3.1 $H^2(\mathfrak{g})$ describes the non-trivial central extensions of $\mathfrak{g}$.
Namely, for $\{\psi\} \in H^2(\mathfrak{g})$, define on $\mathfrak{g} \oplus \mathbb{k}$ the bracket by:

$$[(g_1, c_1), (g_2, c_2)] = ([g_1, g_2], \psi(g_1, g_2))$$

Example 3.2 (Fuks [1986], §1.4) Consider the vector space $\mathfrak{g}$ as a $\mathfrak{g}$ -module with $g \cdot x = [g, x]$ ($g, x \in \mathfrak{g}$), and let $H^*(\mathfrak{g}, \mathfrak{g})$ denote the corresponding cohomology.

- Then: (a) $H^0(\mathfrak{g}; \mathfrak{g})$ is the center of $\mathfrak{g}$, i.e. all $x \in \mathfrak{g}$,
such that $[g, x] = 0 \quad \forall g \in \mathfrak{g}$
(b) $H^1(\mathfrak{g}; \mathfrak{g})$ is the space of exterior derivations.
(c) $H^2(\mathfrak{g}; \mathfrak{g})$ is the space of infinitesimal deformations of $\mathfrak{g}$, i.e. the tangent space to the variety of deformations.

Furthermore we mention that cohomology is useful in

- deriving combinatorial identities, e.g.

$$\prod_{n \geq 1} (1 - t^n) = 1 + \sum_{n \geq 1} (-1)^n (t^{(3n^2+n)/2} + t^{(3n^2-n)/2})$$

- listing invariant differential operators in spaces of tensor fields.

For details we refer to Fuks [1986], Ch. 2.

4. Implementation of d.

We saw one important application of gradings to cohomology (Theorem 2.3). However, not always there exists an element g_0 . Nevertheless, for computations the grading can be very important. Namely it can happen that $\mathfrak{g}$ is infinite-dimensional, but $C_{(\lambda)}^q(\mathfrak{g}; A)$ is finite-dimensional (for all q and λ). This makes computation possible.

For explicit implementing the d-operator, it is even necessary in our approach.

For convenience we assume that $\dim(g) < \infty$ and $\dim(A) < \infty$. All results below are easily extended to the infinite-dimensional case with $\dim(C_{(\lambda)}^q(g; A)) < \infty$. First we consider $A = k$ and $C^q(g)$. Let $\dim(g) = n$ and $\{x_1, \dots, x_n\}$ its basis. We order these elements with their indices, i.e. $x_i < x_j$ if $i < j$. Let $I = (i_1, \dots, i_q)$ be a multi-index, $1 \leq i_1 < \dots < i_q \leq n$. Then $e_I \in C^q(g)$ is defined by

$$e_I(x_{j_1}, \dots, x_{j_q}) = \delta_{I,J} ; J = (j_1, \dots, j_q), j_1 < \dots < j_q.$$

One can check that $\{e_I\}$ is a basis for $C^q(g)$. We also write

$$e_I = e_{i_1} \wedge e_{i_2} \wedge \dots \wedge e_{i_q},$$

and treat $e_{i_1} \wedge \dots \wedge e_{i_q}$ as exterior product, i

$$e_{\sigma(i_1)} \wedge e_{\sigma(i_2)} \wedge \dots \wedge e_{\sigma(i_q)} = \text{sgn}(\sigma) \cdot e_{i_1} \wedge \dots \wedge e_{i_q}$$

where σ is a permutation of $\{i_1, i_2, \dots, i_q\}$.

Proposition Set $[x_i, x_j] = \sum_{k=1}^n c_{ij}^k x_k$.

Then $d(e_k) = \sum_{i < j} c_{ij}^k e_i \wedge e_j$, and

$$d(e_{i_1} \wedge \dots \wedge e_{i_q}) = \sum_{s=1}^q (-1)^{s-1} d(e_{i_s}) \wedge e_{i_1} \wedge \dots \wedge \hat{e}_{i_s} \wedge \dots \wedge e_{i_q}.$$

This proposition yields an effective procedure for calculating $d(e_I)$. The last author programmed it in REDUCE 3.

For e_I , we take $e_I = \text{EXT}(i_1, i_2, \dots, i_q)$, an operator.

Moreover we need the structure constants c_{ij}^k . Hence we defined $\text{LIELIST}(\langle \rangle) = ((c_{i_1 j_1}^{k_1} i_1 j_1) \dots (c_{i_m j_m}^{k_m} i_m j_m))$.

Since terms with $c_{ij}^k = 0$ give no contribution, we omit these in $\text{LIELIST}(K)$. Moreover we have a procedure $\text{MULFORM}(I, J)$, which yields as result $\text{EXT}(I \cup J)$ with $I \cup J$ arranged increasingly, with the proper sign.

In essence $d(\text{EXT}(i_1, \dots, i_q))$ is now given by:

$$d(\text{EXT}(i_1, \dots, i_q)) = \text{SUM OF: FOR EACH } i_s \text{ IN } (i_1 \dots i_q) \\ \text{AND FOR EACH element IN LIELIST}(i_s) \\ (-1)^s * \text{CAR}(\text{element}) * \text{MULFORM}(\text{CDR}(\text{element}), \\ (i_1, \dots, \hat{i}_s, \dots, i_q))$$

Now we consider $C^q(g; A)$, with a non-trivial module A .

Suppose $v^1, \dots, v^m$ is a basis of A . We write $e_i * v^k$ for the element of $C^q(g; A)$ that satisfies

$$e_i * v^k(x_{j_1}, \dots, x_{j_q}) = \delta_{i,j} \cdot v^k.$$

Now looking at the definition of d , we see that d consists of two parts, of which the first part is similar to the d -operator in the case of the trivial module, i.e. as described above. If we denote this part by d_t , and the other part by d_m , we find

$$d(\text{EXT}(i_1, \dots, i_q) * v^k) = d_t(\text{EXT}(i_1, \dots, i_q) * v^k) + d_m(\text{EXT}(i_1, \dots, i_q) * v^k).$$

Let us describe the second part. It is an element of $C^{q+1}(g; A)$, i.e. of the form $\sum \text{EXT}(j_1, \dots, j_{q+1}) * v^j$.

From the definition one sees that this sum need only to range over elements $(j_1, \dots, j_{q+1})$ such that $(i_1, \dots, i_q)$ is contained in it, i.e. $(j_1, \dots, j_{q+1}) = (i_1, \dots, i_{s-1}, i, i_s, \dots, i_q)$.

Let us denote $x_i \cdot v^j = \text{ACT}(i, j)$. Then for $d_m(e_i * v^k)$ we find the following

$$d_m(\text{EXT}(i_1, \dots, i_q) * v^k) = \text{SUM OF: } i = 0 : n \\ (-1)^s * \text{ACT}(i, k) * \text{EXT}(i_1, \dots, i_{s-1}, i, i_s, \dots, i_q)$$

where s is such that $i_{s-1} < i < i_s$.

These two procedures describe the d -operator. Using them one calculates $H^q(g; A)$.

We mention one test for correctness. Because of $d^2 = 0$, one can apply d to A , with $A = dc$. The result must be 0.

5. Some open problems

We mention the problems without further explanation.

Problem 1 Defining relations for maximal nilpotent subalgebras N_+ of vector (or Cartan-type) Lie algebras. This means calculation of the homology $H_2(N_+)$.

Problem 2 Cohomology of $\hat{H}_n$ with trivial module. It is known that $H^*(\hat{H}_n)$ is finite-dimensional (see Fuks [1986], p. 114).

Problem 3 Cohomology of W_n with coefficients $I(\chi_1) \otimes \dots \otimes I(\chi_r)$, where $I(\chi) = \mathbb{C}[\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n}] \otimes V_\chi$. V_χ is the $\mathfrak{gl}(K^n)$ -module with highest weight χ . If ρ_χ is the representation corresponding to V_χ , the W_n -action in $I(\chi)$ is given by setting for $v \in V_\chi$, $D \in W_n$:

$$D(v) = \begin{cases} D \otimes v & \text{if } \deg D = -1 \\ \rho_\chi(D)(v) & \text{if } \deg D = 0 \\ 0 & \text{if } \deg D > 0 \end{cases}$$

and further recursively

$$D(\partial_{i_1} \dots \partial_{i_s} v) = \sum_k \partial_{i_1} \dots \partial_{i_{k-1}} [D, \partial_{i_k}] \partial_{i_{k+1}} \dots \partial_{i_s} v + \partial_{i_1} \dots \partial_{i_s} D(v)$$

The 0-th cohomology corresponds to invariant differential operators (like the exterior, say). Some answers are given in Fuks [1986], Leites [1985], in particular in this way P. Grozman discovered a new operator

$$f(x) \left(\frac{d}{dx} \right)^{-\frac{2}{3}}, g(x) \left(\frac{d}{dx} \right)^{-\frac{2}{3}} \mapsto \left[3 \begin{vmatrix} f & g \\ f'' & g'' \end{vmatrix} + 2 \begin{vmatrix} f' & g' \\ f'' & g'' \end{vmatrix} \right] dx^{\frac{5}{3}},$$

invariant with respect to the group of diffeomorphisms of the 1-dimensional manifold with local coordinate x .

Problem 4 Cohomology of finite-dimensional Lie algebras in prime characteristic p , in particular for $p = 2$, with trivial, adjoint, etc. coefficients. No answers are known for $p = 2$.

Finally, we remark that the notion of (co)homology extends to Lie superalgebras. No (co)homology of infinite-dimensional Lie superalgebras is calculated yet. A list of examples of Lie superalgebras and their modules can be found in Fuks [1986] and Leites [1985]. The most interesting and, fortunately, the easiest problems are the cases of trivial and adjoint representation. All the above problems 1-4 apply to Lie superalgebras as well.

References

- Fuks [1986]: Cohomology of infinite-dimensional Lie algebras, Consultants bureau, New York (1986).
Leites [1985] Lie superalgebras, JOSMAR, vol. 30 (1985).

Also helpful might be:

- D. Leites (ed): Seminar on Supermanifolds (numbers 1-32), Reports Department of Mathematics, Stockholm University.

Use of symbolic methods in analyzing an integral operator

H. F. Trotter, Department of Mathematics

Princeton University, Princeton NJ 08540

Introduction

The "stability of matter" problem in theoretical quantum mechanics is to derive from basic theory a mathematically rigorous lower bound on the energy per nucleus of an arbitrary configuration of nuclei and electrons. Such a lower bound provides a theoretical explanation of why the electrons do not simply collapse into the nuclei. The existence of a lower bound for the energy was originally proved by Dyson and Lenard in [1]. Lieb and Thirring [4,3] later established a much better bound, coming within a factor of about 5.4 of the value suggested by experimental data. Recently, C. Fefferman [2] has presented a method that promises to yield a further improvement. Roughly, the idea is to express the total energy as an integral over all balls (of all sizes) in R^3 , and then for each ball that contains nuclei to assign its energy in equal shares to all nuclei in it, and for each ball that contains no nuclei to assign its energy to the nearest nucleus. The lowest energy assigned to any nucleus is obviously a lower bound for the average energy per nucleus. Arguments given in [2] provide bounds for the energy contributed to a nucleus by all balls except those that are contained within a sphere of radius 2δ about the nucleus and have their center within a distance δ of the nucleus, where 2δ is the distance to the nearest other nucleus. It is also shown in [2] that a lower bound for the energy contributed by the remaining balls can be obtained in terms of the sum of the negative eigenvalues of the quadratic form $Q = K - V$ described below, which is essentially the part contributed by the same family of balls to the energy for a single electron in the field of a stationary nucleus with charge Z . (In the limit as δ goes to infinity, K becomes the kinetic energy given by the Laplacian, and V the Coulomb energy (given by a " $1/r$ " potential) for such an electron.) The present paper discusses the computational problem of getting a rigorous lower bound on the negative eigenvalues of Q . As stated above, Fefferman is entirely responsible for formulating the problem; in addition, he has been closely involved in the computational analysis and some of what is reported here is joint work with him.

The actual problem is three-dimensional, but because it has spherical symmetry, it can be reduced to a series of 1-dimensional problems. Write the vector $\vec{y}$ as $y\vec{\eta}$ with y the length of $\vec{y}$ and $\vec{\eta}$ a unit vector. Then functions of the form $u(\vec{y}) = f(y)\Omega(\vec{\eta})$, where Ω is a spherical harmonic of degree k , form an invariant subspace for each k , and the eigenvalue problem can be looked at separately on each such subspace. Integration over the unit vectors results in an expression for the energy as a quadratic form on the radial function $f(y)$ given by the following formulas derived by Fefferman. They are rather more complicated than the original 3-dimensional formulas

given in [2], but of course the reduction in dimension is worth the complication. The formulas involve two regions of integration, denoted A and B, that correspond to whether or not the sphere of radius y centered at the origin meets the boundary of the ball of radius R with center at distance x from the origin, as shown in Figure 1.

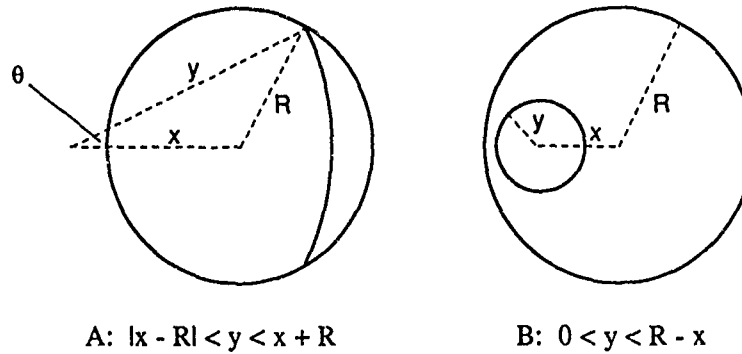


Figure 1

By the eigenvalues of Q , we mean the eigenvalues of the operator A such that $Q(u) = (u, Au)$, where $(\cdot, \cdot)$ is the inner product on L^2 of the ball where u is defined, so that $N(u) = (u, u)$ is the square of the L^2 norm of u . For any quadratic form F , let $v(F)$ be the maximum dimension of subspaces on which F is negative definite. Then for any λ , $v(Q - \lambda N)$ is the number of eigenvalues of Q that are less than λ , and the eigenvalues can be located by finding where v jumps. Note that v is invariant under arbitrary linear changes of coordinates, not just under unitary changes.

$Q(u)$ is given as $K(u) - V(u)$ where K corresponds to kinetic energy, and V to the Coulomb potential. K, V , and N are homogeneous of different degrees in δ . Multiplying Q and N by the same factor does not change v , and it is convenient to scale so that $N(u) = 1$ when u is the constant function 1. We give formulas below for $\delta = 1$. Then in general, $Q(u) = \delta^{-2}K(u) - \delta^{-1}V(u)$. (Actually the formulas below are valid only for a certain range of values of δ . For $\delta > 1/6$, an additional parameter h that depends on δ is introduced. V is modified by a term depending on h , and the region of integration in the definition of K is restricted by the additional inequality $R < h$. This makes the details a little more complicated but does not alter the general nature of the calculation.)

Regardless of the value of k ,

$$V(u) = Z \int_0^2 \left(\frac{1}{y} - \frac{1}{2} + \frac{y}{6} - \frac{4}{6+y} \right) f(y)^2 y^2 dy,$$

and

$$N(u) = \frac{3}{8} \int_0^2 f(y)^2 y^2 dy.$$

For the kinetic energy, $K(u) = \frac{75}{2}\pi \int \frac{E(f,x,R)}{R^6} dx dR$

where the integral is taken over the region $0 < x < 1$, $0 < R < 2 - x$ and

$$E(f,x,R) = \int_A [R^2 - (x-y)^2]xy f(y)^2 dy + 4 \int_B x^2 y^2 f(y)^2 dy \\ - 3x^2 R^{-3} G(f,x,R)^2 - \frac{15}{4}k(k+1)R^{-5} H(f,x,R)^2 - 15x^2 R^{-5} H'(f,x,R)^2$$

where

$$G(f,x,R) = \int_A y^2 M_k(x,y,R) f(y) dy + (\text{if } k=0) 2 \int_B y^2 f(y) dy , \\ H(f,x,R) = \int_A y^2 N_k(x,y,R) f(y) dy + (\text{if } k=1) \frac{4}{3} \int_B xy^3 f(y) dy \\ + (\text{if } k=0) 2 \int_B [R^2 - x^2 - y^2] f(y) dy ,$$

$H'(x,R)$ is the partial derivative of $H(x,R)$ with respect to x ,

$$\text{and } M_k(x,y,R) = \int_{\cos \theta}^1 P_k(t) dt , \\ N_k(x,y,R) = \int_{\cos \theta}^1 [R^2 - x^2 - y^2 + 2xyt] y^2 P_k(t) dt$$

where $P_k(t)$ is the Legendre polynomial of degree k , and θ is the angle shown in the diagram above for region A , so $\cos \theta = (x^2 + y^2 - R^2)/2xy$. Note that, although the details are complicated, the formula is quite elementary. Specifically, *the functions are rational with denominators that are simply products of powers of x , y , and R , and the regions of integration are given by linear inequalities.*

For simplicity, we have given formulas for quadratic forms. The associated bilinear forms are given by the same formulas with $f(y)^2$ replaced by $f_1(y)f_2(y)$ and $G(f,x,R)^2$, etc. replaced by $G(f_1,x,R)G(f_2,x,R)$, etc.

First approach

The restriction of Q to the finite dimensional subspace of L^2 spanned by polynomials of degree $\leq m$ is described by a matrix whose entries are the values of the bilinear form on the monomials $f_1(y) = y^i$, $f_2(y) = y^j$ with $0 \leq i, j \leq m$. Exact evaluation by symbolic integration, first with respect to the y variables, then x , then R , is reasonably straightforward. For $k=0$

the answers take the form of a rational number plus a rational multiple of $\ln(2)$, and for m as high as 15, the rational numbers involved have an only moderately large number of digits. (For higher values of k a few terms involving dilogarithms appear, but it is not difficult to separate them out for special treatment.)

The monomials are a notoriously ill-conditioned basis for the space of polynomials, however, and direct conversion to a floating-point matrix to use in a numerical eigenvalue routine gives hopelessly inaccurate results for m greater than four or five. Changing to a basis of orthogonal polynomials is a simple matter of matrix multiplication, but the number of digits needed for exact rational representation became too large for symbolic calculation to be feasible even for $m = 8$. The practical solution was to have Reduce put out numerical values with 40 significant digits (using the "bigfloat" option) and then take these as input to a Fortran program which carried out the change of basis using extended precision (28 hexadecimal digits). The result, even for $m = 15$, was a well-conditioned matrix accurate to at least 6 or 7 figures, on which a standard eigenvalue package gave results of acceptable accuracy.

The negative eigenvalues obtained in this way of course depend on m ; they change more and more slowly as m increases, and it seems clear that the limiting values are fairly close to the values at $m = 15$. The results are convincingly plausible, and are the basis for the tentative conclusions announced in [2]. We could not, however, find any rigorous justification for the extrapolation involved, and so looked for another approach. The results of the first approach were important, however, in that the values obtained were such as to give some improvement over previous methods in the results on the original problem and thus motivated further investigation.

Second approach

Formally, the formulas above for the kinetic energy can be integrated first with respect to x and R , to obtain a result of the form

$$K(u) = \int_0^2 D(y)f(y)^2 dy + 2 \iint K(y_1, y_2) f(y_1)f(y_2) dy_1 dy_2$$

where the double integral is over the region $0 < y_1 < y_2 < 2$. Here D is found by expressing the contribution to $K(u)$ from the first two terms in $E(f, x, R)$ as integrals over regions in yxR -space, and integrating out x and R . For the term involving $G(f, x, R)^2$, one writes $G(f, x, R)^2 = G_1(f, x, R)G_2(f, x, R)$, where G_1 and G_2 are the same as G , but written using distinct variables of integration y_1, y_2 instead of y . The terms involving H and H' are treated similarly. The contribution from all these terms is an integral in $y_1 y_2 x R$ -space, and $K(y_1, y_2)$ is obtained by integrating out x and R . (The formal integration presents no theoretical difficulties -- the practical difficulties are the same as with the modified problem below and are commented on later.)

The aim is to get estimates from some sort of finite-dimensional approximation (polynomial or piecewise linear, for example) to D and K . Unfortunately, D becomes infinite at $y = 1$, and K becomes infinite at the diagonal $y_1 = y_2$ for $y_2 \leq 1$ in such a way that neither integral converges, even if f is smooth. The problem does not come up in the first approach, because when the integrations in y are done first, cancellations occur (if f is smooth) and the integral over x and R converges even near $R = 0$, in spite of the R^6 in the denominator.

The divergences disappear if the regions of integration are modified to require $R > \epsilon > 0$. It turns out to be feasible to do the integrations with ϵ as a parameter and analyze the behavior as ϵ tends to zero. The result is that if $g(y)$ is defined to be $f'(y)$ on $(0,1)$ and $(f(y)-f(1))/(y-1)$ on $(1,2)$, then $K(u)$ can be expressed as

$$\int_0^2 \bar{D}(y)g(y)^2 dy + 2 \iint K(y_1, y_2) f(y_1)f(y_2) dy_1 dy_2 ,$$

where the integrals converge for g in L^2 . (The operator is related to the Laplacian, so the appearance of f' is not surprising. This reformulation could perhaps have been found "by hand", but the situation at $y = 1$ was not clear until the calculation was done.)

Even though the integrals converge and define a bounded form, $\bar{K}$ is singular at $(1,1)$ and does not define a compact operator, so it does not have a good finite-dimensional approximation. Using symbolic calculation, it is possible to write $\bar{K}$ as $S + K'$, where S is a fairly simple expression embodying the singularity and K' is in L^2 and does give a compact operator. S can be analyzed theoretically, and it can be shown that $\bar{D}$ can be written as $T + D'$, with T a small term such that S and T together give a positive definite form. Discarding T and S will then lower the eigenvalues, so a lower bound calculated using K' and D' will be a lower bound for the original problem.

Note that g determines f only to within an additive constant. Because the kinetic energy is zero for constant functions (a fact that is obvious in the three-dimensional formulation of [2], although not apparent from the formulas given here) it has an expression in terms of g alone. V and N do not vanish on constants, and to express them we have to use $a = f(1)$ as well as g . The result is that

$$N(u) = a^2 + 2a \int s(y)g(y) dy + \int d(y)g(y)^2 dy + \iint k(y_1, y_2)g(y_1)g(y_2) dy_1 dy_2$$

where $s(y) = -y^3/8$ on $(0,1)$ and $3y^2(y-1)/8$ on $(1,2)$

$$d(y) = 0 \quad \text{on } (0,1) \text{ and } 3y^2(y-1)^2/8 \text{ on } (1,2)$$

$$k(y_1, y_2) = \min(y_1, y_2)^3/8 \text{ on } [0,1] \times [0,1] ,$$

and $V(u)$ is given by a similar (but more complicated) formula (in which the coefficient of a^2 is no longer 1).

The final result is that if δ and λ are fixed, we get an expression for $Q - \lambda N$ of the form given just above for N , with s, d , and k linear combinations of the various expressions given above, and we want to determine $v(Q - \lambda N)$. The form is defined on pairs (a, g) , where a is a number and g a function. A change of coordinates replacing a by $a + \int s(y)g(y) dy$ eliminates the term in s , and changes $k(y_1, y_2)$ to $k(y_1, y_2) - s(y_1)s(y_2)$. The value of v is now the value of v for the form defined on g by d and the newly modified k , plus 1 if the

coefficient of a^2 is negative (as it may be, depending on the values of δ and λ). As a final (non-unitary) change of coordinates, replace g by $gd^{1/2}$. In terms of this new g we have a form Q' given as

$$Q'(g) = \int g(y)^2 dy + \iint k'(y_1, y_2) g(y_1) g(y_2) dy_1 dy_2$$

where $k'(y_1, y_2) = d(y_1)^{-1/2} k(y_1, y_2) d(y_2)^{-1/2}$. The kernel k' defines an operator P , and $v(Q')$ is the number of eigenvalues of P that are less than -1 . Now k' is in L^2 and can be approximated in L^2 by functions in a finite dimensional space (for example, piecewise linear functions). In this way we get a finite-dimensional operator P' , and can in principle evaluate v with a known possible error. In fact, sufficiently accurate results can be obtained using piecewise linear functions with about 30 segments. For the sake of reasonable efficiency of computation, symbolically generated formulas were converted to Pascal programs to carry out the numerical work.

Practical considerations

In the second approach, we essentially have to invert the order of integration in the formulas given earlier. Getting K involves the most elaborate calculation; Figure 2 shows the regions of integration in the xR -plane (x is the horizontal coordinate, R the vertical) that apply when $0 < y_1 < y_2 < 1$ (case (i)). Similar but slightly different diagrams apply to three other cases: (ii) $1 < y_1 < y_2 < 2$, (iii) $y_1 < 1 < y_2$ with $y_1 + y_2 < 2$, and (iv) $y_1 < 1 < y_2$ with $y_1 + y_2 > 2$. The a 's, b 's, etc., refer to terms of the integrand that are non-zero in regions where the symbols appear. Terms with the suffix 1 involve y_1 and those with the suffix 2 involve y_2 ; in general, we need the sum of the integrals of $a!$ products of a term of type 1 and a term of type 2. For example, the figure indicates that in case (i) the product $b_1 a_2$ is to be integrated over the rectangle bounded by the lines $x = 2 - R$, $x = R - y_2$, $x = y_2 - R$, and $x = R - y_1$.

Each term to be integrated is a rational function of y_1 , y_2 , x , and R . As a function of x and R , the denominator is simply a product of powers of x and R . Every limit of integration in x is a linear function of y_1 , y_2 , and R , so the result of integrating over x is a rational function with denominator a product of powers of such linear functions, plus terms involving logarithms. Integrating over R then gives functions whose denominators involve linear functions of y_1 and y_2 , again with some logarithms. The factors that appear in denominators (sometimes to powers as high as 6) are y_1 , y_2 , $y_1 + y_2$, $y_1 - y_2$, $y_1 - 1$, $y_2 - 1$, $y_1 + 1$, $y_2 + 1$, $y_1 + 2$, and $y_2 + 2$. Any attempt to add all the terms together over a common denominator chokes the system (and would give an incomprehensible result anyway). Not collecting terms at all loses the benefit of considerable cancellation. The workable approach appears to be to use partial fractions and combine terms whose denominators are powers of the same factor. Setting up a program to keep track of everything automatically is essential (there are too many items to keep track of by hand), but since the denominator factors of each term are known when it is generated, the computation is not expensive.

In fact, it is appropriate to use a kind of partial fraction representation for functions (easily implemented since list structures are now available in Reduce) instead of the conventional rational form. A structure consisting of an offset d and a list of coefficients c_i , $i = 1, 2, \dots$ represents the proper fraction $\sum c_i (x + d)^{-i}$, and a rational function (whose denominator has only linear factors)

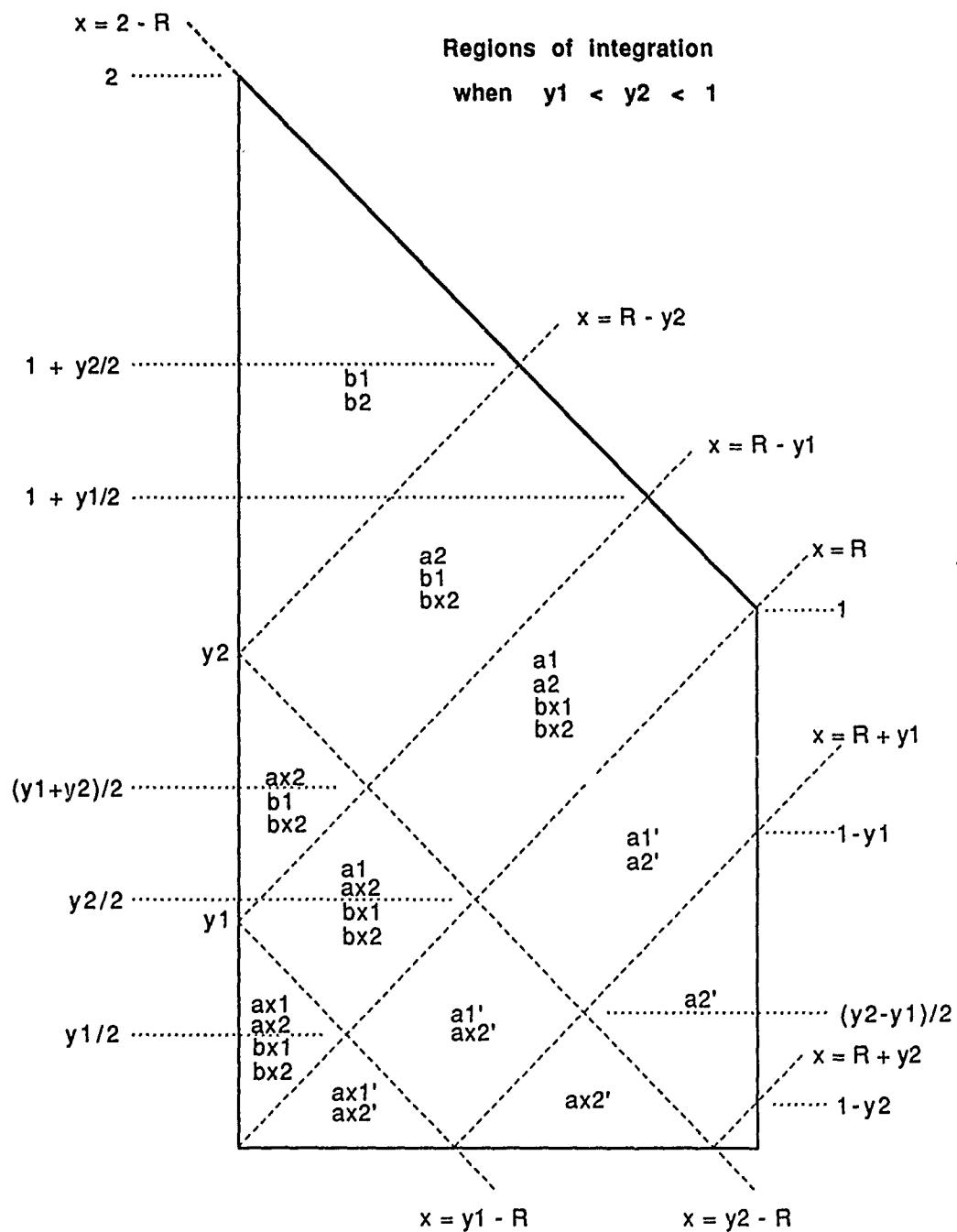


Figure 2

is then uniquely represented as the sum of a polynomial and proper fractions with distinct offsets. In addition, one needs to represent rational function multiples of a small number of special terms such as logarithms. Formal integration for functions in this representation is quite easy and can perhaps be done more efficiently than with the standard package, which must spend a lot of time converting to and from a partial fraction form. Whether this potential gain in efficiency is actually realized, however, has not been checked experimentally.

When $k = 0$, the final result is four similar sets of formulas, one for each of the cases (i)–(iv) above, each of which fits on one page if printed in a 10-point font. The numerical coefficients are integers of which very few have more than six digits, and many have only three or four. For higher values of k the formulas become a little more complicated, and the coefficients get a little larger. (It is hoped that $k > 3$ need not be considered, but this has not yet been proved.)

Error control

The main point of this computation is to obtain results that are *provably* correct. To what extent is this possible?

Symbolic calculation reduces the problem to a form in which truncation error can be bounded in terms of derivatives of explicitly given functions, and which is not highly sensitive to round-off error. Fairly straightforward arguments, aided by a little interval arithmetic, give satisfactory control of the "numerical analysis" error.

Certifying the correctness of the symbolic calculation remains a non-trivial problem. In part, this is no different from justifying a calculation done by hand. First, one is careful; second, one shows the work to a constructively skeptical colleague (if available); finally, enough details are exposed to public view to allow independent checking. This is all that is available to check the original formulas, scaling factors, and the like. It is also available, or should be in some form, for the author to write the calculation. (Interactive work with no audit trail is fine for exploration but not much use for producing verifiable results.)

Even if one takes pains to make sure of the correctness of one's own programming, there remains the question of how much a system like Reduce is to be trusted. I am personally confident that rational function manipulation and formal differentiation work properly all the time. They are used a great deal and the algorithms are well understood. Integration in Reduce appears to work very well, and I have never found it to be in error, but it does not inspire quite the same confidence. Factorization seems to be basically reliable, but I once spent several days tracking down an inconsistency that came up because in some cases the product of the factors found for a polynomial was the negative of the polynomial (an error that has been corrected in the present version of Reduce). Fortunately, the results of integration and factorization can be checked by differentiating and multiplying, and it is easy to make the check automatic.

There are also cases where the system makes no error, but the result cannot be used without correction. For example, the integral of $dx/(2-x)$ comes out as $-\log(x-2)$ even if one is working in the interval $(0,2)$.

There are special hazards in a project of the present kind that involves transfer of symbolic results to a numerical program. The input/output conventions involving line length and the handling of expressions extending over multiple lines can produce errors that are sometimes "invisible" in the

sense that they are syntactically legal. Hand editing needed, for example, to break up expressions too long to be handled in one statement by the Pascal compiler involves additional risks. With full-screen editors it is all too easy to alter a digit in the middle of the screen accidentally when intending to enter a command. Spot checks done by evaluating a formula for a few numbers in both systems give some reassurance. A stronger check used in the current project is to transfer the Pascal statements back to Reduce and verify that they compute the same symbolic values. (This is not difficult to do, thanks in part to the similarity of Pascal and Reduce syntax.)

The most complicated and error-prone part of the present work comes in calculating the kernel functions for the second approach. Fortunately, a very convincing check is available. With a little trouble, the form can be evaluated by exact symbolic calculation on the functions y^i, y^j for several values of i and j using the second approach, and the results compared with the same calculation using the first approach. The trouble is well worth taking. The procedure provides very convincing evidence of correctness and, if the truth be told, has exposed and aided in the diagnosis of several errors.

References

- [1] F. Dyson and A. Lenard, Stability of matter I, II, J. Math. Phys. 8, 1967, pp. 423-434 and 9, 1968, pp. 698-711 .
- [2] C. Fefferman, The N-body problem in quantum mechanics, Commun. Pure Appl. Math. 39 (Supplement), 1986, pp. S67-S109 .
- [3] E. Lieb, The number of bound states of one-body Schrödinger operators and the Weyl problem, AMS Proc. Symp. Pure Math. 36, 1980, pp. 241-252.
- [4] E. Lieb and W. Thirring, Bound for the kinetic energy of fermions which proves the stability of matter, Phys. Rev. Lett. 35, 1975, pp. 687-689 .

Computer Algebraic Methods for Investigating Plane Differential Systems of Center and Focus Type

Dongming Wang

Institute of Systems Science
Academia Sinica
Beijing 100080, China

Abstract

For plane differential systems of center and focus type, the author described an algorithmic procedure based on the principle of Poincaré's method and implemented a program DEMS for computing the Liapunov function and Liapunov constants. This function and these constants are used in the study of stability criteria, differentiation between center and focus and the construction of limit cycles. The solutions of the problems concerning the investigation of Liapunov constants then require that algebraic decision problem, algebraic simplification and algebraic equations solving, to which Wu's characteristic set method and Buchberger's Gröbner basis method are successfully applied. Using the program DEMS and these computer algebraic methods, the author studied some concrete differential systems and obtained the stability criteria and the relations between the computed Liapunov constants and other conditions. In particular, we discovered that Kukles' conditions for the existence of a center for a type of cubic differential systems are possibly incomplete, and presented a class of cubic differential systems with the origin as a 6-tuple focus from which one can create 6 limit cycles by a small perturbation. This paper is a summarization of our recent work.

1. Method and Program for Computing Liapunov Function and Liapunov Constants

For plane autonomous differential system

$$(1.1) \quad \frac{dx}{dt} = ax + by + P(x, y), \quad \frac{dy}{dt} = cx + dy + Q(x, y),$$

where $P(x, y)$ and $Q(x, y)$ are analytic functions beginning with terms of degree greater than 1, if the characteristic equation has no real roots, then the origin is either a center or a focal point. The origin is a center only if the characteristic roots are a pair of purely imaginary numbers, i.e., $p = a + d = 0$, $q = ad - bc > 0$. In the case where the roots are purely imaginary, the system (1.1) is called to be of center and focus type. The investigation of the stability criteria, the differentiation between a center and a focal point, the construction and the number of limit cycles in

this critical case occupy a very important position in the study of plane differential systems, since it is closely related to Hilbert's 16th problem [5] and Arnold's problem [2]. If $P(x, y)$ and $Q(x, y)$ are polynomials of degree n , we then call the system (1.1) an n -differential system.

In what follows, we consider only the case of center and focus type and suppose the system (1.1) is reduced by an appropriate nonsingular transformation to the form

$$(1.2) \quad \begin{aligned} \frac{dx}{dt} &= y + P_2(x, y) + \dots + P_n(x, y) + \dots, \\ \frac{dy}{dt} &= -x + Q_2(x, y) + \dots + Q_n(x, y) + \dots, \end{aligned}$$

where $P_i(x, y)$ and $Q_i(x, y)$ are homogenous polynomials of degree i . Applying the algorithmic method introduced by Poincaré [9], we may thus construct a Liapunov function $F(x, y)$ of system (1.2), requiring only the solution of algebraic equations. To do this, let

$$F(x, y) = \sum_{j=2}^{\infty} F_j(x, y)$$

and

$$F_2(x, y) = \frac{1}{2}(x^2 + y^2), \quad F_j = \sum_{k=0}^j f_{jk} x^{j-k} y^k, \quad j > 2.$$

Obviously, the function $F(x, y)$ is positive near the origin. Differentiating $F(x, y)$ along the integral curve of (1.2) with respect to t , we have

$$\begin{aligned} \frac{dF(x, y)}{dt} &= \frac{\partial F(x, y)}{\partial x} \frac{dx}{dt} + \frac{\partial F(x, y)}{\partial y} \frac{dy}{dt} \\ &= \sum_{j=2}^{\infty} \frac{\partial F_j}{\partial x} (y + \sum_{i=2}^{\infty} P_i) + \sum_{j=2}^{\infty} \frac{\partial F_j}{\partial y} (-x + \sum_{i=2}^{\infty} Q_i) \\ &= \sum_{j=2}^{\infty} \left(\frac{\partial F_{j+1}}{\partial x} y - \frac{\partial F_{j+1}}{\partial y} x + \sum_{i=2}^n \left(\frac{\partial F_{j-i+2}}{\partial x} P_i + \frac{\partial F_{j-i+2}}{\partial y} Q_i \right) \right), \end{aligned}$$

in which $F_i = 0$ for $i < 2$. Setting

$$(1.3) \quad \frac{dF(x, y)}{dt} = v_3 y^4 + v_5 y^6 + \dots + v_{2j+1} y^{2j+2} + \dots$$

and equating the coefficients of powers of x and y in this equality, then up to any homogenous terms of degree $2h$, we obtain a system of $(h+1)(2h+1) - 6$ linear equations in $2(h+1)^2 - 8$ variables f_{ij} , $j = 0, \dots, i$, $i = 3, \dots, 2h$ and $v_3, \dots, v_{2h-1}$. This system of linear equations has at least a set of polynomial solutions, in the indeterminate coefficients of each P_i, Q_i , $i = 2, \dots, 2h$, as variables with rational coefficients. In fact, the coefficients $f_{i0}, f_{i1}, \dots, f_{ii}$ of homogenous terms of degree i , and v_{i-1} when i is even, may be successively evaluated (see [9]).

The above computed $F(x, y)$ is called a Liapunov function of system (1.2). Each v_{2j+1} found from the linear equations is called the j th Liapunov constant. Again, the origin is called an m -tuple focus of differential system (1.2) if the first $m-1$ Liapunov constants of the system are 0 but the m th one is not. If all Liapunov constants of the system are 0, the origin is said to be a center.

Based on the above procedure, the author implemented a program DEMS for computing the Liapunov function and Liapunov constants on an HP1000 using Fortran77 and in the computer algebra system Scratchpad II on an IBM4341 [12, 13]. This program has been applied to compute the Liapunov constants of various concrete differential systems. The obtained constants are also used for the study of stability criteria, center and focus decision, and the construction and number of limit cycles [6, 12-15].

2. The Investigation of Liapunov Constants Using Computer Algebraic Methods

According to Liapunov's stability theorem, for the differential system (1.2) the stability of the origin may be determined by the sign of $dF(x, y)/dt$, i.e., the origin is unstable when $dF(x, y)/dt > 0$; the origin is asymptotically stable when $dF(x, y)/dt < 0$; the origin is a center (stable but not asymptotically stable) when $dF(x, y)/dt = 0$. Thus the stability of the origin may be successively determined by the signs of $v_3, v_5, \dots$.

From this criteria, we know that the differentiation between a center and a focal point requires usually infinitely many operations. If (1.2) is an n -differential system, the required infinitely many conditions $v_{2j+1} = 0, j = 1, 2, \dots$, relate but only to a finite number of indeterminates. The ideal consisting of the polynomials whose vanishing is both a necessary and a sufficient condition for the existence of a center, has a finite basis. Hence there is a minimal integral valued function $N(n)$ such that all the conditions $v_{2j+1} = 0$ for $j > N(n)$ are formal consequences of such conditions for $j \leq N(n)$. In order to make an effective use of the criteria, the problem for determining such a minimal $N(n)$ is undoubtedly important. This problem has not yet been solved up to now.

However, it is necessary to consider v_{2j+1} in the stability criteria only when $v_3 = v_5 = \dots = v_{2j-1} = 0$. Since the computed Liapunov constants are always too complicated to be applied to the stability criteria, if $v_{2j+1} = 0$ is not a consequence of the condition $v_3 = v_5 = \dots = v_{2j-1} = 0$, the next indispensable step is to simplify v_{2j+1} using this condition. On the other hand, if $v_3 = v_5 = \dots = v_{N(n)} = 0$, which is certainly complicated too, and possibly, does not have real solutions, is a necessary and sufficient condition for the existence of a center, then finding an equivalent but simpler condition instead of it is, of course, important.

For constructing the limit cycles, we have to investigate the multiplicity of the focus and search into the particular differential systems with higher multiple foci. A

fundamental theorem [1] indicates that if the multiplicity of a focus of the differential system (1.2) is m , then one can create m , and at most m , limit cycles from the focus by a small perturbation. In view of these reasons, we suggest the following four problems:

1. *Decide whether the condition $v_{2j+1} = 0$ is a formal consequence of the condition $v_3 = v_5 = \dots = v_{2j-1} = 0$;*
2. *If $v_{2j+1} = 0$ is not a consequence of $v_3 = v_5 = \dots = v_{2j-1} = 0$, simplify v_{2j+1} by using $v_3 = v_5 = \dots = v_{2j-1} = 0$;*
3. *Find an equivalent but simpler condition instead of the condition $v_3 = v_5 = \dots = v_{N(n)} = 0$;*
4. *Find a particular differential system such that $v_3 = v_5 = \dots = v_{2j-1} = 0$ but $v_{2j+1} \neq 0$.*

If the coefficients of $P(x, y)$ and $Q(x, y)$ satisfy certain known conditions given by sets of algebraic equations, we then have naturally a problem for determining the inferable relations between the given conditions and the Liapunov constants. Evidently, a complete solution of this problem is directly from that of problem 1.

For solving the above problems, we described several algorithms based on the algorithmic principles of the characteristic set method developed by Wu [16] and Gröbner basis method developed by Buchberger [4]. Problem 1 is actually an algebraic decision problem for determining whether a non-zero polynomial vanishes on an algebraic variety defined by a set of polynomial equations. By Hilbert Nullstellensatz, this problem is also equivalent to deciding whether a polynomial belongs to a radical ideal generated by the set of polynomials. Hence it may be completely solved by either the characteristic set method or Gröbner basis method.

In fact, let (PS) be the polynomial set consisting of $v_3, v_5, \dots, v_{2j-1}$ and y be a new indeterminate. To solve problem 1, we then need only determine whether the polynomial set $(PS) \cup \{y \cdot v_{2j+1} - 1\}$ has no zeros by the algorithm described in Wu's zero structure theorem, or to check whether the Gröbner basis of $(PS) \cup \{y \cdot v_{2j+1} - 1\}$ includes 1. Similarly, we may also compute the characteristic series of (PS) and check if the remainders of v_{2j+1} with respect to all ascending sets in the characteristic series are 0, or compute the Gröbner basis of (PS) and check whether the normal form of certain powers of v_{2j+1} with respect to the Gröbner basis is 0. Aided by polynomial factorization and the bound of power in Hilbert Nullstellensatz, problem 1 can be completely solved. For detailed discussions, see [15].

The problems 2 and 3 are problems concerning algebraic simplification, one of the basic techniques in algebraic manipulation. For computing unique representations of equivalent objects, there have been a lot of exciting results. However, for obtaining equivalent but simpler objects, the investigation is perplexing and seemingly more

difficult. Fortunately, for simplifying our obtained Liapunov constants, both the algorithmic principles of the characteristic set method and Gröbner basis method may be successfully applied.

To fix idea, we define for a non-zero polynomial F that

$$mt(F) = \min_{F=f_1^{a_1}\dots f_t^{a_t}} \sum_{i=1}^t term(f_i),$$

in which each $term(f_i)$ is the number of terms of polynomial f_i , and choose $mt(F)$ as the measure of F 's complexity. A polynomial F is said to be simpler than the polynomial G if $mt(F) < mt(G)$. For a polynomial set (PS) we define $mt(PS)$ to be the sum of all $mt(F)$ for F in (PS) . Then the concept *simpler* for polynomial sets or systems of polynomial sets is similarly defined. Again, the totality of zeros of all polynomials in a polynomial set (PS) will be denoted by $Zero(PS)$. If G is any other non-zero polynomial, then the subset of $Zero(PS)$ for which $G \neq 0$ will be denoted by $Zero(PS/G)$. Under these definitions, we consider the following alternative problems instead of the problems 2 and 3.

- 2'. For a given polynomial set (PS) and a non-zero polynomial g , find two polynomial sets $(HS) = \{h_1, \dots, h_r\}$ and $(DS) = \{D_1, \dots, D_t\}$ such that $(HS) \cup (DS)$ is simpler than g and

$$Zero(PS) = \bigcup_{i=1}^r Zero(PS_i/J_i),$$

$$g - h_i|_{Zero(PS_i/J_i)} = 0, \quad i = 1, \dots, r,$$

where $(PS_i) = (PS) \cup \{D_j : j \in \alpha_i\}$, $J_i = \prod_{j \in \beta_i} D_j$ and index sets $\alpha_i, \beta_i \subset \{1, \dots, t\}$.

- 3'. For a given polynomial set (PS) , find a system of polynomial sets $\Psi = \{(PS_1), \dots, (PS_r)\}$ and another polynomial set $(DS) = \{D_1, \dots, D_t\}$ such that $\Psi \cup (DS)$ is simpler than (PS) and

$$Zero(PS) = \bigcup_{i=1}^r Zero(PS_i/J_i),$$

where $J_i = \prod_{j \in \alpha_i} D_j$ and $\alpha_i \subset \{1, \dots, t\}$.

In the case $(DS) = \phi$ and $r = 1$, the above problem 2' is actually to find a polynomial h simpler than g such that g and h are equivalent modulo $Zero(PS)$, and problem 3' is to find another polynomial set (PS') with the same zeros as, but simpler than (PS) . Based on the algorithmic principles of characteristic set method and Gröbner basis method, some algorithms for solving these problems were presented in [15]. By those algorithms the computed Liapunov constants may be greatly simplified.

Since all $v_3, v_5, \dots, v_{2j+1}$ are polynomials in variables $x_1, \dots, x_e$, the indeterminate coefficients of P_i and Q_i , with rational coefficients, then problem 4 is equivalent to finding a solution $x_i = x_i^0, i = 1, \dots, e$ and $c = c^0 \neq 0$ of the polynomial equations $v_3 = 0, \dots, v_{2j-1} = 0, v_{2j+1} = c$ in a certain extension field of $\mathbb{Q}$ (especially, $\mathbb{Q}$ itself or real field $\mathbb{R}$). This solution may be found by any known methods. Generally speaking, only in the complex field the known methods can completely determine the solvability of algebraic equations. For our purpose, the solutions to be found basically have to be in $\mathbb{Q}, \mathbb{R}$ or a transcendental extension field of $\mathbb{Q}$. This makes the problem related to the decision problems for polynomial definiteness and Diophantos equation and thus uneasy to be solved immediately. On the other hand, the Liapunov constants computed from the given differential systems consist of many terms and usually, are very complicated. Hence it is almost impossible to solve such a set of algebraic equations using the known methods even on a big computer. However for our present concrete problems, we do not attempt to find all solutions of the equations and thus may use some specific techniques and do appropriate experiments to overcome a part of the difficulty. Combining with Wu's method, Buchberger's method and necessary trial, some particular differential systems with higher multiple focus have been found (see next section).

3. Some Concrete Results

Our program for computing Liapunov constants and the computer algebraic methods for solving the suggested problems have been applied to the investigation of some concrete differential systems. In this section, we sum up a part of our obtained results. For quadratic differential systems, the first three Liapunov constants were already given by Bautin in 1952 and $N(2)$ was also proved to be 3. Using our program DEMS, we may easily obtain these constants and recheck the sign error of Bautin's results.

For a particular cubic differential system of which the homogenous terms of degree 2 do not appear, we computed the Liapunov constants and obtained the relations between these constants and the conditions for the existence of a center given by Saharnikov [10]. We thus rediscovered independently the incompleteness of Saharnikov's conditions [12]. In fact, this error was already discovered by Sibirskii [11] early in 1965. For this particular system, let M be the minimal integer such that all the conditions $v_{2j+1} = 0$ for $j > M$ are formal consequences of such conditions for $j \leq M$. Then Sibirskii proved $M = 5$, which implies $N(3) \geq 5$. However, our computed Liapunov constants are still useful to the stability criteria and the construction of limit cycles.

In the case $P_2(x, y) = \dots = P_n(x, y) = 0$ of n -differential system, Kukles [7] established certain criteria for the existence of a center. He applied his criteria to the cases $n = 3, 5$ and got the necessary and sufficient conditions for the existence of a center. For $n = 3$ and writing $Q_2 = a_{20}x^2 + a_{11}xy + a_{02}y^2$, $Q_3 = a_{30}x^3 + a_{21}x^2y +$

$a_{12}xy^2 + a_{03}y^3$, Kukles' conditions are given by four sets of algebraic equations [9]. For this cubic differential system, the first five Liapunov constants consisting of 4, 19, 76, 218 and 527 terms were computed by the program DEMS. They may be simplified to be polynomials consisting of at most 4, 11, 30, 71 and 168 terms respectively. By investigating the relations between these constants and Kukles' conditions, Jin and the present author discovered that Kukles' conditions are possibly incomplete [6]. In particular, let

$$a_{11} = 0, a_{12} = 0, a_{21} = -3a_{03}, a_{20} = -\frac{1}{2}a_{02}, a_{30} = -\frac{1}{12}a_{02}^2$$

and $288a_{03}^2 - a_{02}^4 = 0$, $a_{03}a_{02} \neq 0$, then $v_3 = v_5 = \dots = v_{17} \equiv 0$ but none of Kukles' conditions holds. Hence either Kukles' conditions for the existence of a center are incomplete, or for this particular system the multiplicity of the origin as a focus is greater than or equal to 9. It seems that the origin is impossible to be a focus in this case and thus Kukles' conditions are incomplete. We conjecture that for this cubic differential system considered by Kukles, v_{2j+1} for $j \geq 6$ are formal consequences of $v_3 = v_5 = \dots = v_{11} = 0$.

By solving algebraic equations and aided with appropriate experiments, we presented in [14] a particular cubic differential system of the form

$$(3.1) \quad \begin{aligned} \frac{dx}{dt} &= y - ax^2 - 2bxy + ay^2 - Cx^3 + (7A - 6B)x^2y + \frac{5}{3}Cxy^2 - (A - 2B)y^3, \\ \frac{dy}{dt} &= -x - bx^2 + 2axy + by^2 - Ax^3 - \frac{5}{3}Cx^2y + (7A - 8B)xy^2 + Cy^3, \end{aligned}$$

for which the first four Liapunov constants are computed to be 0 and the next two are

$$v_{11} = \frac{448}{2673}(2a^2 + 2b^2 + 11B)w, \quad v_{13} = -\frac{896}{1563705}\Delta \cdot w,$$

where Δ is a polynomial consisting of 11 terms in variables a, b, A, B, C with integer coefficients and

$$w = -(a^2 - b^2)C[C^2 - 27(A - B)^2] + 18ab(A - B)[C^2 - 3(A - B)^2].$$

Moreover, we have

$$v_{13}|_{2a^2+2b^2+11B=0} = \frac{14336}{62799435}\Gamma \cdot w,$$

in which Γ is a positive polynomial consisting of 7 terms.

Hence for the differential system (3.1), if $2a^2 + 2b^2 + 11B = 0$ but $w \neq 0$, then the origin is a 6-tuple focus, from which we can thus create 6 limit cycles [14]. We guess that in the case $w = 0$ the origin is a center of this differential system. This guess was proved in the case $C = 0$. For differential system (3.1), if we let M be the minimal integer such that all the conditions $v_{2j+1} = 0$ for $j > M$ are formal consequences of such conditions for $j \leq M$, then our results show $M \geq 6$ ($M = 6$ in the case $C = 0$),

and thus show $N(3) \geq 6$. For this particular system, some stability criteria of the origin were also given in [14].

Note that the problem for deciding the maximal number of limit cycles of the plane n -differential system was proposed by Hilbert in 1900 as the second part of his 16th problem [5]. This problem has gained little progress in the past 88 years. If we denote the maximal numbers of limit cycles around the origin and on the whole plane of n -differential system by $H_0(n)$ and $H(n)$ respectively, then the known results are only $H_0(2) = N(2) = 3$, $4 \leq H(2) \leq \infty$, $H_0(3) = N(3) \geq 6$, $H(3) \geq 11$ and $H(n) \geq H_0(n) = N(n)$. So far in any other cases the problem remains open.

Due to the restriction of our computer memory, the Liapunov constants for general cubic differential system were unable to be computed and the stability criteria thus have not yet been given. It is very hopeful to get the complete results on a big computer now or in the near future. However, the described computer algebraic methods should make a step of progress in the study of this problem, especially for some concrete systems.

Acknowledgement

This work is partly done during my visit at NTZ, Karl-Marx University. I am grateful to Professor Wolfgang Lassner for the invitation.

References

- [1] A. A. Andronov, E. A. Leontovich, I. I. Gordon and A. G. Maier, *Theory of Bifurcations of Dynamic Systems on a Plane*, Israel Program for Scientific Translations, Jerusalem, 1971.
- [2] V. I. Arnold et al, *Problems of Present Day Mathematics*, Mathematical Developments Arising from Hilbert Problems, Proceedings of Symposia in Pure Mathematics, Amer. Math. Soc., 28(1976), 35-80.
- [3] N. N. Bautin, *On the Number of Limit Cycles Which Appear with the Variation of Coefficients from an Equilibrium Position of Focus or Center Type (in Russian)*, *Matematicheskii Sbornik (N.S.)*, 30(1952), 72, 181-196. Also in *Stability and Dynamic Systems, Translations, Series one*, Amer. Math. Soc., 5(1962), 396-413.
- [4] B. Buchberger, *Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory*, *Multidimensional Systems Theory* (N. K. Bose ed.), Reidel Publishing Company, Dordrecht-Boston-Lancaster, 1985, 184-232.
- [5] D. Hilbert, *Mathematische Probleme*, *Archiv der Math. u. Phys.* (3), 1(1901), 44-63; 213-237.

- [6] X. F. Jin and D. M. Wang, *On Kukles' Conditions for the Existence of a Center*, Submitted to Bulletin of London Mathematical Society, September 1988.
- [7] I. S. Kukles, *Sur les conditions nécessaires et suffisantes pour l'existence d'un centre*, Doklady Akad. Nauk, 42(1944), 160-163.
- [8] ———, *Sur quelques cas de distinction entre un foyer et un centre*, Doklady Akad. Nauk, 42(1944), 208-211.
- [9] V. V. Nemytskii and V. V. Stepanov, *Qualitative Theory of Differential Equations*, Princeton University Press, New Jersey, 1960.
- [10] N. A. Saharnikov, *Solution of the Problem of the Center and the Focus in One Case (in Russian)*, Akad. Nauk SSSR. Prikl. Mat. Meh., 14(1950), 651-658.
- [11] K. S. Sibirskii, *On the Number of Limit Cycles in the Neighborhood of a Singular Point (in Russian)*, Differencial'nye Uravnenija, 1(1965), 53-66.
- [12] D. M. Wang, *Mechanical Approach for Polynomial Set and its Related Fields (in Chinese)*, Ph.D thesis, Academia Sinica, July 1987.
- [13] ———, *Mechanical Manipulation for a Class of Differential Systems*, Submitted to J. of Symbolic Computation, May 1988.
- [14] ———, *A Class of Cubic Differential Systems with 6-tuple Focus*, Submitted to J. of Differential Equations, July 1988.
- [15] ———, *The Applications of Characteristic Sets and Gröbner Bases to Problems Concerning Liapunov Constants*, RISC-LINZ Series no. 88-49.0, Johannes Kepler University, September 1988.
- [16] W. T. Wu, *Basic Principles of Mechanical Theorem Proving in Elementary Geometries*, J. Sys. Sci. & Math. Scis., 4(1984), 207-235. Also in J. of Automated Reasoning, 2(1986), 221-252.

An Example of Computer Enhanced Analysis

Peter J. Costa and Ruth Hampton Westlake
Raytheon Company, 430 Boston Post Road, Wayland, MA 01778

Abstract. In this report, a first order nonlinear partial differential equation with a parameter dependent initial condition is examined. Even though an analytic solution of the equation is determined, a surprising bifurcation phenomenon is discovered via computer graphics. This "computer-discovered" bifurcation, in turn, leads to further mathematical analysis and deeper geometric understanding of the solution. Indeed, this is a simple example of an elementary catastrophe (in the sense of Thom) and demonstrates the usefulness of numerical computations in providing qualitative information even in the presence of exact solutions.

§1 The Problem

Equation (1.1), with initial condition (1.2), is studied in nonlinear optics where α acts as a "lens focusing" parameter.

$$\frac{\partial u}{\partial y} = - \left[\frac{\partial u}{\partial x} \right]^2 \quad (1.1)$$

$$u(x, 0) = \sin(\alpha x) \quad (1.2)$$

The solution of (1.1) – (1.2) is a straightforward exercise in the theory of characteristic curves. More precisely, let $p = \frac{\partial u}{\partial x}$, $q = \frac{\partial u}{\partial y}$, $z = u(x, y)$, $\lambda(x) = u(x, 0)$, and $\gamma(x) = \lambda'(x)$. Let $F(x, y, z, p, q) = 0$ describe (1.1) and let $\psi(x)$ be the solution of $F(x, 0, \lambda(x), \gamma(x), \psi(x)) = 0$. In this case, $F = p^2 + q$, $\lambda(x) = \sin(\alpha x)$, $\gamma(x) = \alpha \cos(\alpha x)$, and $\psi(x) = -\alpha^2 \cos^2(\alpha x)$.

If one solves the following system (1.3) of ordinary differential equations, then a parameterized solution of (1.1) – (1.2) is obtained. For complete details, see e.g., John [4] or Zachmanoglou and Thoe [7].

$$\left. \begin{aligned} \frac{dx}{dt} &= \frac{\partial F}{\partial p} \\ x(0) &= s \end{aligned} \right\} \quad (1.3a)$$

$$\left. \begin{aligned} \frac{dy}{dt} &= \frac{\partial F}{\partial q} \\ y(0) &= 0 \end{aligned} \right\} \quad (1.3b)$$

$$\left. \begin{aligned} \frac{dz}{dt} &= p \frac{\partial F}{\partial p} + q \frac{\partial F}{\partial q} \\ z(0) &= \lambda(s) = \sin(\alpha s) \end{aligned} \right\} \quad (1.3c)$$

$$\left. \begin{aligned} \frac{dp}{dt} &= -\frac{\partial F}{\partial x} - p \frac{\partial F}{\partial z} \\ p(0) &= \gamma(s) = \alpha \cos(\alpha s) \end{aligned} \right\} \quad (1.3d)$$

$$\left. \begin{aligned} \frac{dq}{dt} &= -\frac{\partial F}{\partial y} - q \frac{\partial F}{\partial z} \\ q(0) &= \psi(s) = -\alpha^2 \cos^2(\alpha s) \end{aligned} \right\} \quad (1.3e)$$

Equations (1.3a) – (1.3e) determine a complete system for (1.1) – (1.2) with solution $z = u(x, y)$, parameterized by s and t , given below.

$$x = X(s, t), \quad y = Y(s, t), \quad z = Z(s, t), \quad p = P(s, t), \quad q = Q(s, t) \quad (1.4)$$

From (1.4) it is seen that

$$z = U(s, t) = u(X(s, t), Y(s, t)) \quad (1.5)$$

is the parameterized solution.

§2 Solution and Bifurcation

Integration of equations (1.3a) – (1.3e) yields the parameterization

$$\left. \begin{aligned} x &= X(s, t) = 2\alpha t \cos(\alpha s) + s \\ y &= Y(s, t) = t \\ z &= Z(s, t) = \alpha^2 t \cos^2(\alpha s) + \sin(\alpha s) \\ p &= P(s, t) = \alpha \cos(\alpha s) \\ q &= Q(s, t) = -\alpha^2 \cos^2(\alpha s) \end{aligned} \right\} \quad (2.1)$$

Therefore, the desired solution of (1.1) – (1.2) is

$$\left. \begin{aligned} x &= x(s) = 2\alpha y \cos(\alpha s) + s \\ u(x, y) &= u(x(s), y) = \alpha^2 y \cos^2(\alpha s) + \sin(\alpha s) \end{aligned} \right\} \quad (2.2)$$

To gain insight into the geometry of the solution, we employed a computer to graph (2.2) for varying values of α . We kept y fixed ($y = \frac{1}{2}$) and took $s \in [-\pi, \pi]$. For $0 < \alpha < 1$, the solution (2.2) exhibits no remarkable behavior. See Fig. 1. For $\alpha = 1$, however, the derivative of the solution with respect to x becomes nonunique for $x \approx 1$; i.e., u loses smoothness. See Fig. 2. For $\alpha = 2$, not only does u lose smoothness but also becomes multivalued. As α increases so does the number of nondifferentiable points and number of multivalued solutions. See Figs. 3 – 6.

Furthermore, for $\alpha > 1$, we observe that u has locally minimal values of -1 and 1 . Numerical investigations led to the metatheorem following Figs. 1 – 6.

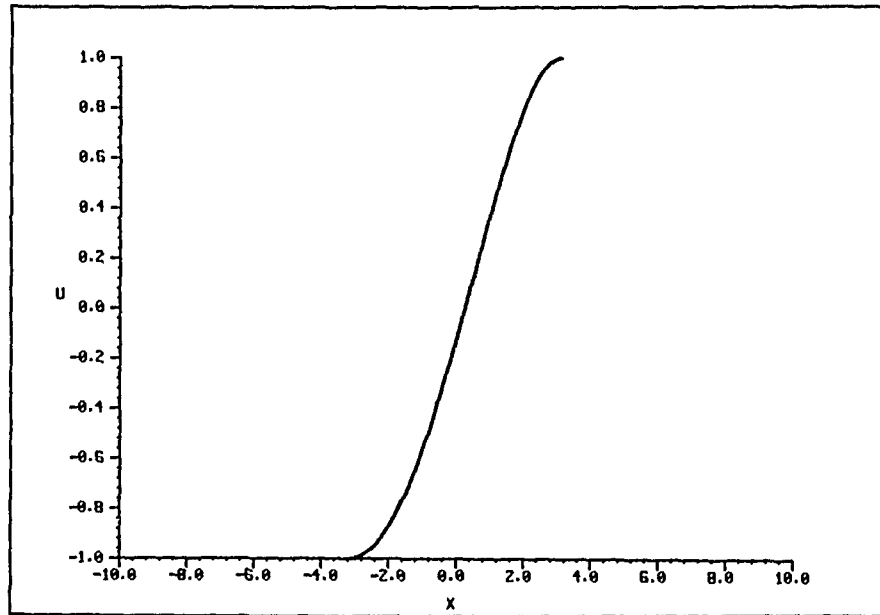


Figure 1. Single-valued solution ($\alpha = 0.5$)

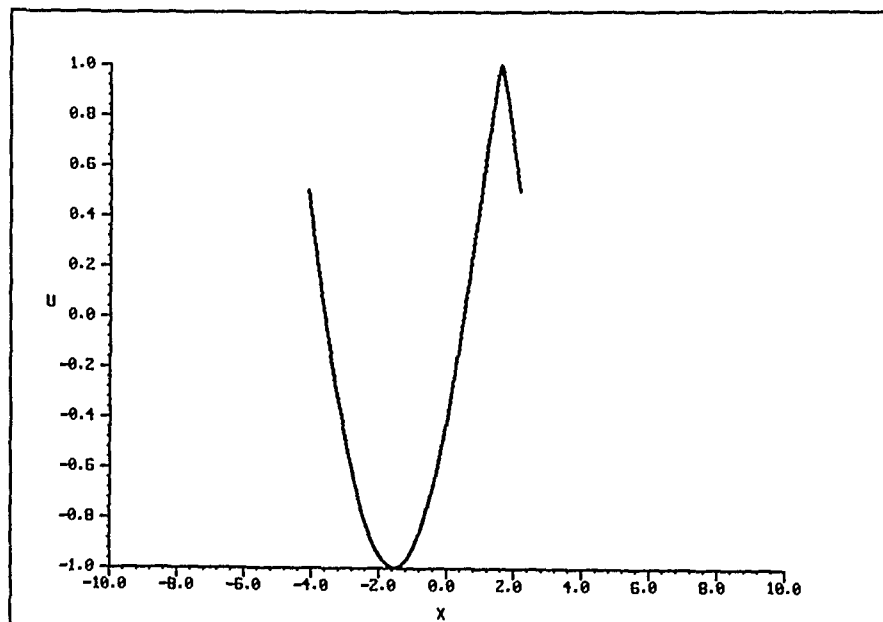


Figure 2. Single-valued solution; u is not differentiable near $x = 1.0$ ($\alpha = 1.0$)

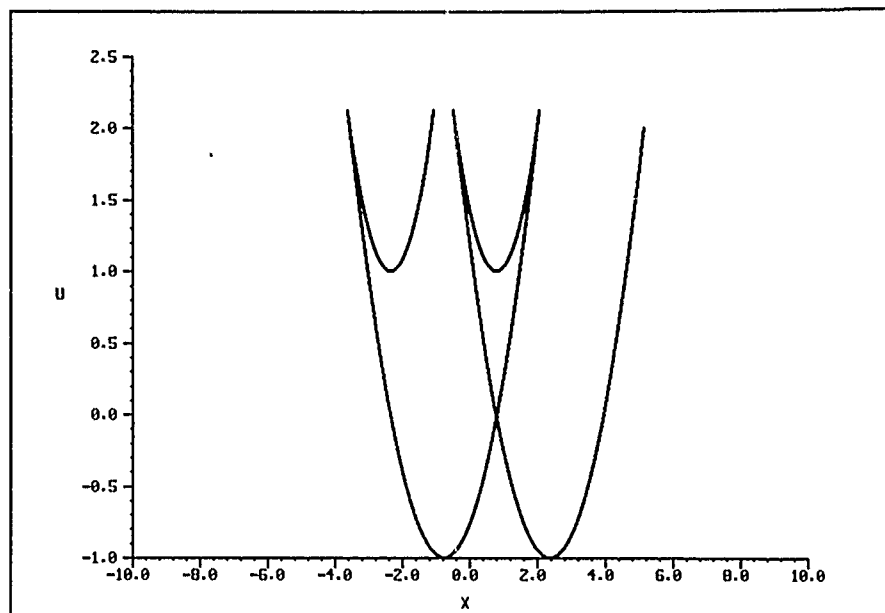


Figure 3. Triple-valued solution, five nondifferentiable points ($\alpha = 2.0$)

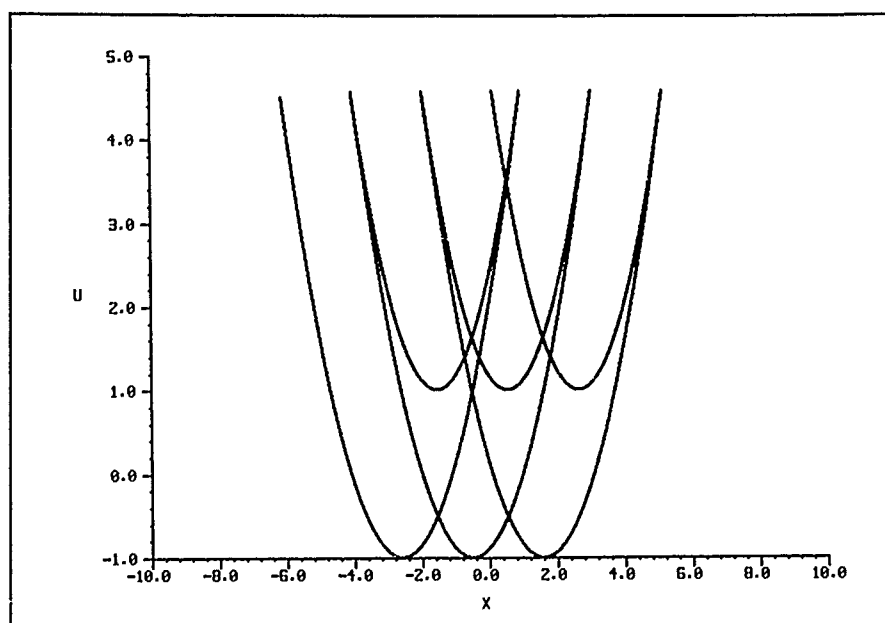


Figure 4. Solution with six values; seven nondifferentiable points ($\alpha = 3.0$)

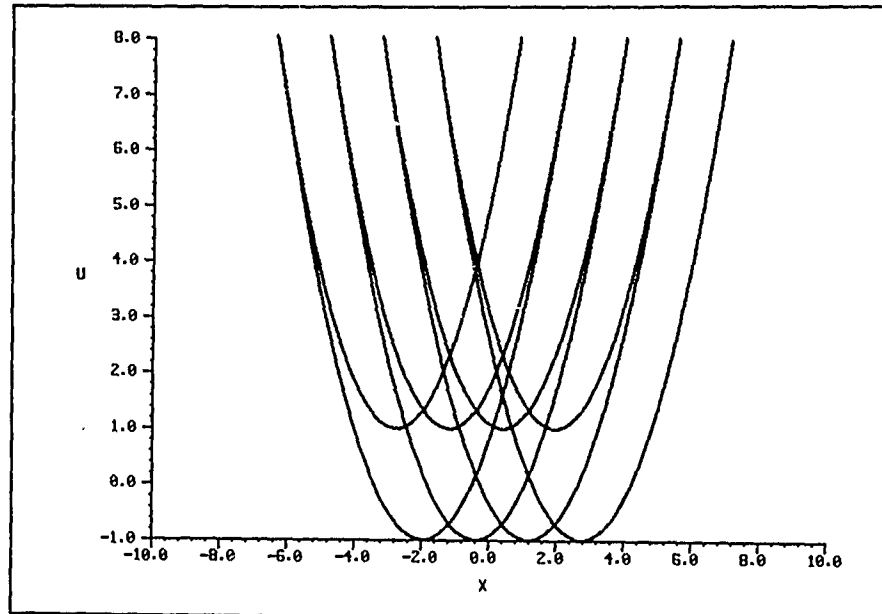


Figure 5. Solution with eight values; nine nondifferentiable points ($\alpha = 4.0$)

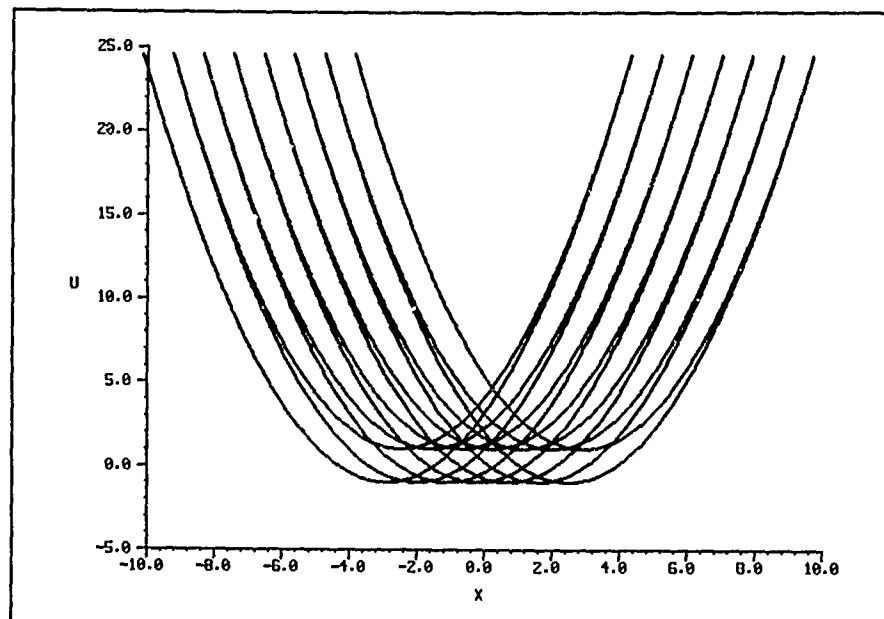


Figure 6. Solution with fourteen values; fifteen nondifferentiable points ($\alpha = 7.0$)

Empirical Result: For $0 < \alpha < 1$, (2.2) yields a smooth (i.e., C^∞) solution of (1.1) – (1.2). For $\alpha \geq 1$, (2.2) is a continuous, multivalued, not everywhere differentiable function with locally minimal values of -1 and 1 . Moreover, for $\alpha = 3, 4, 5, \dots$, the number of multivalued solutions is 2α , and the number of nondifferentiable points is $2\alpha + 1$, $s \in [-\pi, \pi]$. For $\alpha = 2$, u has 3 values and 5 nondifferentiable points, $s \in [-\pi, \pi]$.

The (incomplete) bifurcation diagram for (2.2) is given in Fig. 7.

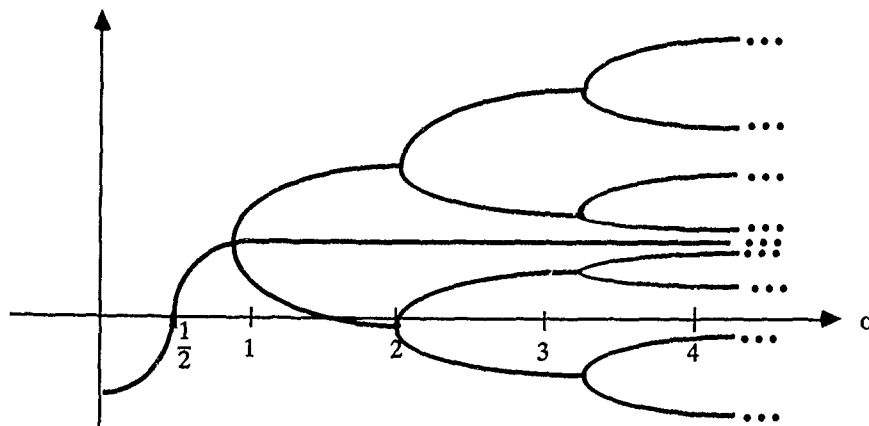


Figure 7. Bifurcation of u as a function of α

The natural question to ask is "What accounts for these effects?" To answer this query, further mathematical analysis is required and discussed in the next section.

§3 Hamilton-Jacobi Equations and Conservation Laws

Following Strang [5, 6], let $v = \frac{\partial u}{\partial x}$. Then the Hamilton-Jacobi equation (1.1) is equivalent to the Conservation Law

$$\frac{\partial v}{\partial y} + 2v \frac{\partial v}{\partial x} = 0 \quad . \quad (3.1)$$

By a direct application of the chain rule, it can be seen that $v_0(x) = \frac{\partial u}{\partial x}(x, 0) = \alpha \cos(\alpha x)$. The solution of (3.1) then is given implicitly by

$$v(x, y) = v_0(x - 2yv) = \alpha \cos(\alpha [x - 2yv]) \quad . \quad (3.2)$$

Since the solution surface (2.2) apparently is not smooth in x , it seems reasonable to examine the characteristic lines of the implicit solution for $\frac{\partial u}{\partial x}$. That is, consider the family of lines

$$x - 2yv = \text{constant} \quad . \quad (3.3)$$

Each characteristic line or pulse starts at the point $x = x^*$ when $y = 0$. Therefore, (3.3) is equivalent to

$$y = \frac{1}{2v^*}(x - x^*) \quad (3.4)$$

where $v^* = v_0(x^*) = \alpha \cos(\alpha x^*)$. As x^* is arbitrary, let it take the value π . Then the slopes of the characteristic lines, as a function of the parameter α , are given by the formula below.

$$m_\alpha = \frac{1}{2v^*} = \frac{1}{2\alpha \cos(\alpha\pi)} \quad (3.5)$$

§4 Geometry, Computing, and Analysis

Consider the graphs of the characteristic lines (3.4) with $x^* = \pi$ and the slope $m_\alpha = \frac{1}{2v^*}$ given in (3.5). For $0 < \alpha < \frac{1}{4}$, the slopes m_α decrease from $+\infty$ to some $m_{\alpha_0} \leq 2.8 = m_{1/4}$. Then the slopes m_α "fold over" and begin to increase back up to $+\infty$; $\frac{1}{4} < \alpha < \frac{1}{2}$ implies $2.8 < m_\alpha < +\infty$. At $\alpha = \frac{1}{2}$, a "catastrophe" occurs as m_α changes signs from $+\infty$ to $-\infty$. The opposite occurs for $\frac{1}{2} < \alpha < 1.1$. In this case, $-\infty < m_\alpha < m_{\alpha_1}$ where $m_{\alpha_1} \geq m_{1.1} = -0.48$. For some $\alpha_1 \geq 1.1$, the m_α begin to decrease: $1.1 \leq \alpha_1 < 1.5$ implies $-\infty < m_\alpha < -0.48 < m_{\alpha_1}$. Again, the "fold over" effect occurs. See Fig. 8.

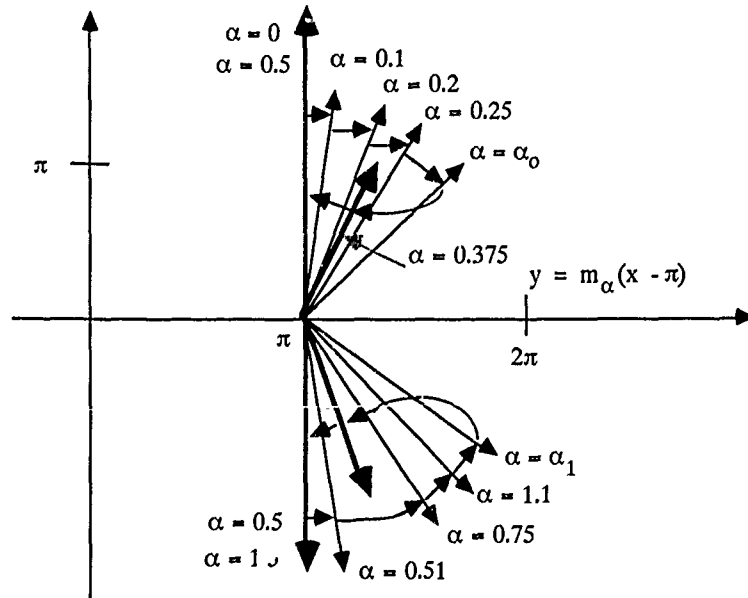


Figure 8. Slopes of Characteristic Lines ($0.0 < \alpha < 1.5$)

A similar pattern occurs for $1.5 < \alpha < \alpha_2$, for some $\alpha_2 \geq 2.0$: The slopes decrease from $+\infty$ to m_{α_2} . Then the slopes m_α begin to increase again toward $+\infty$ as $2.0 < \alpha < 2.5$: $\frac{1}{4} < m_\alpha < +\infty$. For $2.5 < \alpha < 3.5$, the slope changes signs, reaches a maximum value, and then decreases again toward $-\infty$. See Fig. 9.

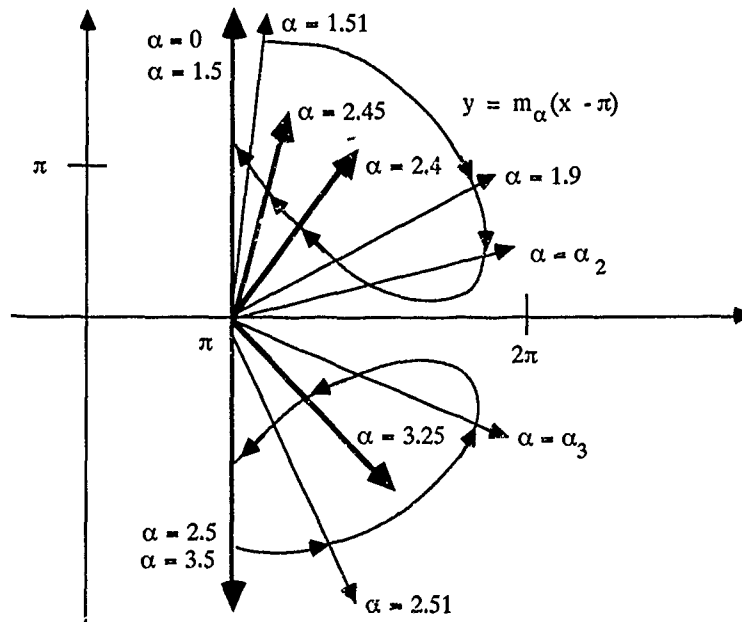


Figure 9. Slopes of Characteristic Lines ($1.5 < \alpha < 3.5$)

Our computations once again give us insight into the qualitative behavior of the solution surface (2.2). Namely, the smoothness of the derivative $\frac{\partial u}{\partial x}$ is destroyed for certain values of α since the slopes of the characteristic lines (of $\frac{\partial u}{\partial x}$) change signs. Hence, the points where u loses differentiability (i.e., smoothness) can be determined as a function of α .

Plainly, as $\alpha \rightarrow \infty$, $m_\alpha \rightarrow 0$ and the characteristic rays (of fixed length) fill in a semicircle in the right half plane. Moreover, the change in signs of the m_α can be modelled as an elementary catastrophe in the sense of Thom (see Ekeland [3]). In this case, the catastrophe is a cusp, as represented in Fig. 10 below.

§6 Summary

Our computations have enlightened our understanding of the qualitative nature of our solution, by showing us that the solution surface (2.2) has nondifferentiable points and multiple values for different values of α . Whereas, our analysis has enhanced and explained the results of some of our

computations. Such a rewarding symbiosis should not be overlooked. Indeed, that is the general thrust of this report: To give an example of a "simple" nonlinear partial differential equation with complex behavior whose qualitative nature is enhanced by computing.

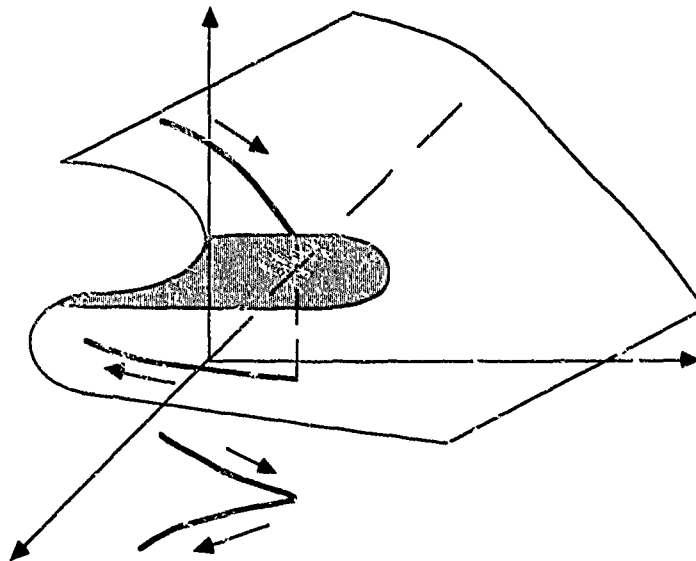


Figure 10. Cusp: As the slopes m_α change sign

Bibliography

- [1] G. F. Carey, B. N. Jiang, and R. Showalter. *A Regularization-Stabilization Technique for Nonlinear Conservation Equation Computations*, Num. Methods for Par. Diff. Eqns., Vol. 4, No. 3, pp. 165-171, Fall 1988.
- [2] P. J. Costa. *An Explicit Solution of a First Order Nonlinear Partial Differential Equation*. MIT Lincoln Laboratory Technical Memorandum No. 34L-0016, 9 June 1986.
- [3] I. Ekeland. *Mathematics and the Unexpected*, University of Chicago Press, 1988.
- [4] F. John. *Partial Differential Equations*, Fourth Edition, Springer-Verlag, 1982.
- [5] G. Strang. *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, 1986.
- [6] G. Strang. Private communication, 1988.
- [7] E. C. Zachmanoglou and D. Thoe. *Introduction to Partial Differential Equations*, Williams & Wilkins Company, 1976.

An Algorithm for Symbolic Computation of Hopf Bifurcation

Emilio Freire, Estanislao Gamero, Enrique Ponce

Department of Applied Mathematics, University of Sevilla, Spain

Abstract. *The Hopf bifurcation has become a widely used method in the study of periodic oscillations of nonlinear dynamical systems. The purpose of this paper is not to carry out a direct symbolic algebraic manipulation of formulae characterizing this bifurcation (direction, stability and amplitudes of bifurcating periodic orbits, ...). It is planned to develop a recursive algorithm well suited to symbolic computation implementation, which is based upon the normal form approach and supplies the necessary information to characterize generalized Hopf bifurcations.*

An efficient procedure to obtain the normal form corresponding to a Hopf bifurcation is presented; it is based upon the use of Lie transforms. The calculations are arranged in a recursive scheme using complex variables and so the computational effort is optimized. The devised algorithm is implemented on REDUCE 3.2 and applied to several examples.

1. Introduction

The Hopf bifurcation has become a widely used method in the study of periodic oscillations of nonlinear dynamical systems. Several authors ([4], [6], [7], [9], [10]) have derived formulae characterizing this bifurcation (direction, stability and amplitudes of bifurcating periodic orbits, ...). Center manifold reductions and normal form transformations are useful techniques to obtain those formulae ([6],[7]). There are other possibilities as, for example, the application of Lyapunov-Schmidt theory (see [4]).

In the study of degenerate Hopf bifurcations the hand calculation (as opposed to numerical evaluation) of very long expressions is required, when the corresponding bifurcation formulae are being used ([4], [6], [7], [15]). The purpose of this paper is not to carry out a direct symbolic algebraic manipulation of bifurcation formulae. It is planned to develop a recursive algorithm well suited to symbolic computation implementation, which is based upon the normal form approach and supplies the necessary information to characterize generalized Hopf bifurcations.

In section 2, the application of normal form theory to the Hopf bifurcation is presented; it is shown that the description of bifurcation phenomena is achieved from the knowledge of corresponding normal forms. Using MACSYMA, Rand et al. ([12], [13], [14]) have introduced computer algebra in bifurcation methods. For the case of Hopf bifurcation, they use the normal form approach, but their procedure is not optimized because they do not take full advantage of transformation theory leading up to normal forms.

In section 3, an efficient procedure to obtain the normal form corresponding to a Hopf bifurcation is presented; it is based upon the use of Lie transforms as contained in [1], [11]. The calculations are arranged in a recursive scheme using complex variables and so the com-

putational effort is optimized. Finally, in section 4, the devised algorithm is applied to several examples by means of a REDUCE program.

2. Normal forms and Hopf bifurcation

A widespread approach to characterize Hopf bifurcations consists in transforming the system to the so-called normal form. For this, it may be necessary to previously compute its center manifold which would lead to a reduced two-dimensional system. This task can be accomplished in an efficient way by means of the algorithm described in an earlier paper [2].

So, consider the system

$$\begin{aligned}\dot{x} &= F(x, y, \mu) \\ \dot{y} &= G(x, y, \mu)\end{aligned}\tag{2.1}$$

with isolated equilibrium point at the origin whose jacobian matrix for this point has the canonical form

$$A(\mu) = \begin{bmatrix} \alpha(\mu) & -\omega(\mu) \\ \omega(\mu) & \alpha(\mu) \end{bmatrix}$$

where μ is the bifurcation parameter and F, G are smooth. For $\mu = 0$ it is verified $\alpha(0) = 0$, $\omega(0) = \omega_0 > 0$ and $\alpha'(0) \neq 0$. The appearance of bifurcating periodic orbits for the system is named a Hopf bifurcation [10].

As it is outlined in the following, in view of the hypothesis $\alpha'(0) \neq 0$, to characterize this bifurcation (number and stability of bifurcating periodic solutions) it is enough to consider the system at $\mu = 0$:

$$\begin{aligned}\dot{x} &= -\omega_0 y + \sum_{k \geq 2} F_k(x, y) \\ \dot{y} &= \omega_0 x + \sum_{k \geq 2} G_k(x, y)\end{aligned}\tag{2.2}$$

where formal expansions are assumed for $F(x, y, 0), G(x, y, 0)$ and $F_k, G_k \in V(k)$, the linear space of all homogeneous polynomials in x, y of degree k .

It is possible to transform (2.2) by means of successive near-identity transformations into the normal form ([5], [6]):

$$\begin{aligned}\dot{x} &= -\omega_0 y + \sum_{j \geq 1} \{a_j(x^2 + y^2)^j x - b_j(x^2 + y^2)^j y\} \\ \dot{y} &= \omega_0 x + \sum_{j \geq 1} \{a_j(x^2 + y^2)^j y + b_j(x^2 + y^2)^j x\}\end{aligned}\tag{2.3}$$

which is expressed in polar coordinates as

$$\begin{aligned}\dot{r} &= r \sum_{j \geq 1} a_j r^{2j} \\ \dot{\theta} &= \omega_0 + \sum_{j \geq 1} b_j r^{2j}\end{aligned}\tag{2.4}$$

The parameterized system (2.1) can be brought to the following form

$$\begin{aligned}\dot{r} &= \alpha'(0)\mu + r \sum_{j \geq 1} a_j r^{2j} \\ \dot{\theta} &= \omega_0 + \sum_{j \geq 1} b_j r^{2j}\end{aligned}\quad (2.5)$$

For $|r| \ll 1$, the dominant term of the θ -equation is ω_0 and so the bifurcating behavior is determined by the r -equation. Thus a_j are essential for the characterization of Hopf bifurcation ([4], [7], [9]).

Here follows a brief description about the obtention of normal form (2.1). It is assumed that the normal form is already computed to the $k-1$ step:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = A(0) \begin{pmatrix} x \\ y \end{pmatrix} + J_{2k-1}(x, y) + R_{2k}(x, y) + \dots \quad (2.6)$$

where

$$J_{2k-1}(x, y) = \sum_{j=2}^{k-1} (x^2 + y^2)^j \begin{pmatrix} a_j & -b_j \\ b_j & a_j \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

and $R_{2k} \in V(2k, 2)$, the linear space of 2-vector homogeneous polynomials of $2k$ -degree. If the near identity transformation

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} + P_{2k}(\tilde{x}, \tilde{y}), \quad \text{where } P_{2k} \in V(2k, 2) \quad (2.7)$$

is performed the following is obtained:

$$\begin{pmatrix} \dot{\tilde{x}} \\ \dot{\tilde{y}} \end{pmatrix} = A(0) \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} + J_{2k-1}(\tilde{x}, \tilde{y}) + \{R_{2k}(\tilde{x}, \tilde{y}) - L_{2k}P_{2k}(\tilde{x}, \tilde{y})\} + \dots \quad (2.8)$$

where the linear operator $L_{2k} : V(2k, 2) \rightarrow V(2k, 2)$ defined by

$$L_{2k}P_{2k}(\tilde{x}, \tilde{y}) = DP_{2k}(\tilde{x}, \tilde{y})A(0) \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} - A(0)P_{2k}(\tilde{x}, \tilde{y})$$

has been introduced.

It turns out ([1], [5]) that L_{2k} is injective and so it is possible to solve uniquely

$$L_{2k}P_{2k} = R_{2k} \quad (2.9)$$

and therefore to remove $2k$ -degree terms.

Dropping the tildes, the normal form obtained is

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = A(0) \begin{pmatrix} x \\ y \end{pmatrix} + J_{2k-1}(x, y) + R_{2k+1}(x, y) + \dots \quad (2.10)$$

where $R_{2k+1} \in V(2k+1, 2)$. Now, to complete the k -step, a new near-identity transformation must be applied:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} + P_{2k+1}(\tilde{x}, \tilde{y}), \quad P_{2k+1} \in V(2k+1, 2) \quad (2.11)$$

transforming (2.10) into

$$\begin{pmatrix} \dot{\tilde{x}} \\ \dot{\tilde{y}} \end{pmatrix} = A(0) \begin{pmatrix} \tilde{x} \\ \tilde{y} \end{pmatrix} + J_{2k-1}(\tilde{x}, \tilde{y}) + \{R_{2k+1}(\tilde{x}, \tilde{y}) - L_{2k+1}P_{2k+1}(\tilde{x}, \tilde{y})\} + \dots \quad (2.12)$$

The linear operator L_{2k+1} is no longer injective. Let $V^r(2k+1, 2)$ denote the range of L_{2k+1} , and $V^c(2k+1, 2)$ its kernel; it then holds that:

$$\begin{aligned} \text{a) } V(2k+1, 2) &= V^r(2k+1, 2) \oplus V^c(2k+1, 2) \\ \text{b) } V^c(2k+1, 2) &= \text{span}\{(x^2 + y^2)^k \begin{pmatrix} x \\ y \end{pmatrix}, (x^2 + y^2)^k \begin{pmatrix} -y \\ x \end{pmatrix}\} \\ \text{c) } L_{2k+1}V^r(2k+1, 2) &= V^r(2k+1, 2) \end{aligned} \quad (2.13)$$

According to (2.13.a), one can decompose $R_{2k+1} = R_{2k+1}^r + R_{2k+1}^c$ (the superscripts denote range and kernel components respectively); also, it is possible to choose uniquely $P_{2k+1} \in V^r(2k+1, 2)$ such that

$$L_{2k+1}P_{2k+1} = R_{2k+1}^r \quad (2.14)$$

and so the $(2k+1)$ -degree terms of (2.12) becomes R_{2k+1}^c . Now, dropping the tildes, the system (2.12) takes the form:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = A(0) \begin{pmatrix} x \\ y \end{pmatrix} + J_{2k+1}(x, y) + \dots \quad (2.15)$$

It should be noticed that the normal form obtained is equivariant under arbitrary rotations, which is a consequence of the symmetry of linear part $(-\omega_0 y, \omega_0 x)^t$ with respect to the rotation group. As it has been shown above, this normal form has a simple representation in polar coordinates (see 2.4).

Summarizing, to compute a_k, b_k , one must:

- (1) Calculate R_{2k} in (2.6) which represents the $2k$ -degree terms produced by previous transformation on the original system;
- (2) Solve the $4k$ -dimensional linear equation (2.9) so obtaining P_{2k} ;
- (3) Calculate R_{2k+1} in (2.10) taking into account the previous transformations and the corresponding one to P_{2k} ;
- (4) Decompose R_{2k+1} according to (2.13.a);
- (5) Solve the $(4k+2)$ -dimensional linear equation (2.14) to obtain P_{2k+1} .

A direct translation of this computational scheme using MACSYMA can be found in [13]. The authors of this work perform (2) and (5) in real coordinates; as it will be seen below, use of complex coordinates results in both a halving of dimension and a simpler structure of matrix representation of linear operators L_{2k}, L_{2k+1} , making the projection involved in (4) also easier. In the quoted work the calculations of R_{2k}, R_{2k+1} are performed by direct substitution of previous transformations; moreover, no profit is made from the corresponding

computations in the previous steps. To sum up, it seems that proceeding in such a way the computational effort is not being optimized.

In this paper an algorithm for Hopf bifurcation well suited to symbolic computation is presented. The algorithm is organized according to an iterative scheme making good use of the previous steps, thereby minimizing the number of operations and the memory requirements.

3. An efficient way to compute a_k, b_k

As it has been mentioned, to increase efficiency in the Hopf bifurcation computation, it is natural to introduce complex variables ([7], [9]). But what is more relevant is the possibility of using Lie transforms in the theory of normal forms ([1], [11]), which leads to a recursive way of obtaining the transformed equations from original ones. With these ideas in mind, an efficient procedure for symbolic computation of Hopf bifurcation normal forms can be derived.

Thus, making $z = x + iy$, $\bar{z} = x - iy$ in (2.2), where the bars denote conjugation, one obtains

$$\begin{aligned}\dot{z} &= \omega_0 iz + \sum_{k \geq 2} Z_k(z, \bar{z}) \\ \dot{\bar{z}} &= -\omega_0 i \bar{z} + \sum_{k \geq 2} \overline{Z_k(z, \bar{z})}\end{aligned}\quad (3.1)$$

where

$$Z_k(z, \bar{z}) = F_k\left(\frac{z + \bar{z}}{2}, \frac{z - \bar{z}}{2i}\right) + iG_k\left(\frac{z + \bar{z}}{2}, \frac{z - \bar{z}}{2i}\right), \quad k \geq 2$$

and it will be taken later that $Z_1(z, \bar{z}) = \omega_0 iz$. Note that $Z_k \in \mathcal{V}(k)$, the linear space of complex homogeneous polynomials in $z, \bar{z}$ of degree k .

Now, consider the near-identity transformation

$$z = w + \sum_{k \geq 2} u_k(w, \bar{w}) \quad (3.2)$$

where $u_k \in \mathcal{V}(k)$. Conjugation of (3.2) provides the transformation to be considered for $\bar{z}$. This change of variables yields:

$$\dot{w} = \omega_0 iw + \sum_{k \geq 1} W_k(w, \bar{w})/k! \quad (3.3)$$

where $W_k \in \mathcal{V}(k+1)$. It is clear that it suffices to work only with the equations and transformations involving the variables z, w , for the conjugation operation produces the corresponding ones for $\bar{z}, \bar{w}$.

The key observation is that W_k can be obtained by recursive expressions as follows: For $k = 0, 1, \dots$; $l = 1, 2, \dots, k$, let the following be defined

$$\begin{aligned}W_k^0 &= k! Z_{k+1} \\ W_{k-l}^l &= W_{k-l+1}^{l-1} + \sum_{j=0}^{k-l} \binom{k-l}{j} W_{k-l-j}^{l-1} \odot U_j\end{aligned}\quad (3.4)$$

where $U_j \in \mathcal{V}(j+2)$ are related with the transformation (3.2). Also, the $\odot$ -product for a pair $W(w, \bar{w})$ and $U(w, \bar{w})$ has been introduced:

$$W \odot U = \frac{\partial W}{\partial w} U + \frac{\partial W}{\partial \bar{w}} \bar{U} - \frac{\partial U}{\partial w} W - \frac{\partial U}{\partial \bar{w}} \bar{W} \quad (3.5)$$

From this it is obvious that $W'_{k-l} \in \mathcal{V}(k+1)$ for all l . With these definitions it can be proved that

$$W_k = W_0^k, \quad k = 1, 2, \dots \quad (3.6)$$

It must be remarked that the computations in (3.4) can be accomplished by considering the so-called Lie triangle ([1], [11]). The searched-for elements W_k^0 appear on the diagonal of the Lie triangle:

$$\begin{array}{ccccccc} W_0^0 & & & & & & \\ W_1^0 & W_0^1 & & & & & \\ W_2^0 & W_1^1 & W_0^2 & & & & \\ W_3^0 & W_2^1 & W_1^2 & W_0^3 & & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & & \end{array}$$

where each element can be calculated by using the entries in the column one step to the left and up.

In order to simplify W_k —so putting (3.3) in normal form—an appropriate choice of U_k must be made. It is easy to verify—see (2.8), (2.10)—that

$$W_k = W_0^k = \mathcal{R}_k - \mathcal{L}_k U_{k-1}, \quad k \geq 1 \quad (3.7)$$

where $\mathcal{L}_k U = -W_0^0 \otimes U$, $U \in \mathcal{V}(k+1)$. $\mathcal{L}_k : \mathcal{V}(k+1) \rightarrow \mathcal{V}(k+1)$ is a linear operator and $\mathcal{R}_k \in \mathcal{V}(k+1)$, which depends on the $U_j, 0 \leq j \leq k-2$ and so, if these are known, then $\mathcal{R}_k$ is known.

Since $\mathcal{L}_k$ results nonsingular for k odd, it is possible to choose $U_{k-1} \in \mathcal{V}(k+1)$ such that W_k vanishes in these cases (cf. 2.9).

If k is even, the linear operator $\mathcal{L}_k$ is singular. Let $\mathcal{V}^r(k+1)$ denote its range and $\mathcal{V}^c(k+1)$ its kernel. It then holds (see 2.13) that:

$$\begin{aligned} \text{a) } \mathcal{V}(k+1) &= \mathcal{V}^r(k+1) \oplus \mathcal{V}^c(k+1) \\ \text{b) } \mathcal{V}^c(k+1) &= \text{span}\{(w\overline{w})^{k/2}w\} \\ \text{c) } \mathcal{L}_k(\mathcal{V}^r(k+1)) &= \mathcal{V}^r(k+1) \end{aligned} \quad (3.8)$$

According to (3.8) it is possible now to choose uniquely $U_{k-1} \in \mathcal{V}^r(k+1)$ such that $W_k \in \mathcal{V}^c(k+1)$. In other words, if this is done likewise for $k = 2m$, then $W_k(w, \bar{w}) = d_m(w\bar{w})^m w$, $m \geq 1, d_m \in \mathbb{C}$. Therefore, it has been shown that (3.3) adopts the following normal form:

$$\dot{w} = \omega_0 i w + \sum_{m \geq 1} \frac{d_m}{(2m)!} (w \bar{w})^m w \quad (3.9)$$

Comparing (3.9) with the real normal form (2.3), it is concluded that

$$\frac{d_m}{(2m)!} = a_m + ib_m, \quad m \geq 1 \quad (3.10)$$

Using this computing process, several improvements are achieved. On the one hand the dimension of the equations to be solved is halved and the corresponding operators take a diagonal form. On the other, only the operations needed are performed, lowering the possibility

of exhausting the computing facilities with unnecessary terms that must later be truncated [13].

4. Programing aspects and computational results

To carry out the above algorithm on a computer algebra system several observations are in order. It is possible to implement the algorithm by selecting the appropriate primitives of the computer algebra system, merely reproducing the mentioned steps. However, it is more efficient and less expensive to use a vectorial representation of the polynomial functions involved.

So, for each $k \geq 1$, a lexicographic ordered basis in $\mathcal{V}(k+1)$ can be used. Let $\mathcal{P}_{k+1}$ be the ordered set of 2-indices with module $k+1$. Then

$$\mathcal{B}_{k+1} = \{(z, \bar{z})^p : p \in \mathcal{P}_{k+1}\}$$

is a basis of $\mathcal{V}(k+1)$, where $(z, \bar{z})^p = z^{p_1} \bar{z}^{p_2}$, for $p = (p_1, p_2) \in \mathcal{P}_{k+1}$.

To determine the matrix representation of $\mathcal{L}_k$ over $\mathcal{B}_{k+1}$ observe that:

$$\mathcal{L}_k \{(z, \bar{z})^p\} = -(\omega_0 i z) \odot (z, \bar{z})^p = (p_1 - p_2 - 1) \omega_0 i (z, \bar{z})^p \quad (4.1)$$

and so $\mathcal{L}_k$ can be identified with a diagonal matrix

$$\mathcal{L}_k \equiv \omega_0 i \cdot \text{diag}\{-k-2, -k, -k+2, \dots, k-2, k\} \quad (4.2)$$

In order to compute the vectorial representations in $\mathcal{B}_{k+1}$ of the Lie triangle entries, the $\odot$ -operation must be considered (see 3.4). From (3.5) it is easily verified that the $\odot$ -operation is additive (but non-homogeneous) and therefore it suffices to analyze it for monomials. If $\alpha, \beta \in \mathbb{C}$ and $p = (p_1, p_2) \in \mathcal{P}_{l_1}$, $q = (q_1, q_2) \in \mathcal{P}_{l_2}$ with $l_1, l_2 \geq 2$ then:

$$\begin{aligned} \{\alpha(z, \bar{z})^p\} \odot \{\beta(z, \bar{z})^q\} &= \alpha\beta(p_1 - q_1) z^{p_1+q_1-1} \bar{z}^{p_2+q_2} + \\ &+ \alpha\bar{\beta}p_2 z^{p_1+q_2} \bar{z}^{p_2+q_1-1} - \\ &- \bar{\alpha}\beta q_2 z^{p_2+q_1} \bar{z}^{p_1+q_2-1} \end{aligned} \quad (4.3)$$

Thus, the computation of successive rows in the Lie triangle is a straightforward task. For each k , to obtain $\mathcal{R}_k$ it is a practical approach to segregate it from W_k in (3.7). That can be accomplished by neglecting the term $W_0^0 \odot U_{k-1}$ in W_{k-1}^1 , thereby eliminating this term in the corresponding row of the Lie triangle. Once $\mathcal{R}_k$ is obtained, the row must be corrected appropriately.

For k odd, the matrix representation of $\mathcal{L}_k$ permits a direct computation of U_{k-1} such that $\mathcal{L}_k U_{k-1} = \mathcal{R}_k$ and so W_k vanishes. For $k = 2m$, there is one diagonal element equal to zero in the matrix (4.2), and now U_{k-1} can be selected in such a way that $\mathcal{L}_k U_{k-1} = \mathcal{R}_k^r$, where the superscript r denotes the projection of $\mathcal{R}_k$ over the range of $\mathcal{L}_k$. As it can be seen, this projection is an elementary task. Therefore $W_k = \mathcal{R}_k^r$, which is spanned by the element $w^{m+1}\bar{w}^m$, and the corresponding coordinate is the d_m searched for.

With these ideas, a first implementation of the algorithm described in the previous section has been obtained on REDUCE [8]. A vectorial representation of both polynomials and operators has been used. Some previously known examples have been solved to test the program and demonstrate its performance. A selection of the results achieved is presented in the following.

Example 1: Computing formulas for a_1, b_1

Consider the general system

$$\begin{aligned}\dot{x} &= -\omega_0 y + f(x, y) \\ \dot{y} &= \omega_0 x + g(x, y)\end{aligned}\tag{4.4}$$

with $f(0, 0) = g(0, 0) = 0$, $Df(0, 0) = Dg(0, 0) = (0, 0)$. After 61 seconds of μ VAX-II time, the well-known formulas ([5], [15]) for a_1, b_1 are obtained:

$$\begin{aligned}a_1 &= (\omega_0 g_{yyy} + \omega_0 f_{xyy} + \omega_0 g_{xxy} + \omega_0 f_{xxx} + f_{yy} g_{yy} + f_{yy} f_{xy} \\ &\quad - g_{yy} g_{xy} + f_{xy} f_{xx} - g_{xy} g_{xx} - f_{xx} g_{xx}) / (16\omega_0) \\ b_1 &= (-3\omega_0 f_{yyy} + 3\omega_0 g_{xyy} - 3\omega_0 f_{xxy} + 3\omega_0 g_{xxx} - 5f_{yy}^2 + f_{yy} g_{xy} - 5f_{yy} f_{xx} - 2g_{yy}^2 \\ &\quad + 5g_{yy} f_{xy} - 5g_{yy} g_{xx} - 2f_{xy}^2 + f_{xy} g_{xx} - 2g_{xy}^2 + 5g_{xy} f_{xx} - 2f_{xx}^2 - 5g_{xx}^2) / (48\omega_0)\end{aligned}$$

Example 2: Computing formulas for a_1, b_1, a_2, b_2 in presence of a symmetry

Consider the system (4.4) with the following additional properties:

$$f(-x, -y) = -f(x, y), \quad g(-x, -y) = -g(x, y)$$

After 794 seconds of μ VAX-II time, the procedure derives the general formulas for a_1, b_1, a_2, b_2 in this case:

$$\begin{aligned}a_1 &= (g_{yyy} + f_{xyy} + g_{xxy} + f_{xxx}) / 16 \\ b_1 &= (-f_{yyy} + g_{xyy} - f_{xxy} + g_{xxx}) / 16 \\ a_2 &= (\omega_0 g_{yyyy} + \omega_0 f_{xyyy} + 2\omega_0 g_{xxyy} + 2\omega_0 f_{xxxy} + \omega_0 g_{xxxxy} + \omega_0 f_{xxxxx} + f_{yyy} g_{yyy} \\ &\quad + f_{yyy} f_{xyy} - f_{yyy} g_{xxy} - f_{yyy} f_{xxx} - g_{yyy} g_{xyy} - g_{yyy} f_{xxy} + g_{yyy} g_{xxx} \\ &\quad + 3f_{xyy} g_{xyy} + 3f_{xyy} f_{xxy} + f_{xyy} g_{xxx} - 3g_{xyy} g_{xxy} + g_{xyy} f_{xxx} - 3f_{xxy} g_{xxy} \\ &\quad + f_{xxy} f_{xxx} - g_{xxy} g_{xxx} - f_{xxx} g_{xxx}) / (384\omega_0) \\ b_2 &= (-8\omega_0 f_{yyyy} + 8\omega_0 g_{xyyy} - 16\omega_0 f_{xxyy} + 16\omega_0 g_{xxxy} - 8\omega_0 f_{xxxxy} + 8\omega_0 g_{xxxxx} \\ &\quad - 17f_{yyy}^2 - 22f_{yyy} g_{xyy} - 10f_{yyy} f_{xxy} - 30f_{yyy} g_{xxx} - 9g_{yyy}^2 + 38g_{yyy} f_{xyy} \\ &\quad - 26g_{yyy} g_{xxy} + 14g_{yyy} f_{xxx} - 33f_{xyy}^2 + 30f_{xyy} g_{xxy} - 26f_{xyy} f_{xxx} - 9g_{xyy}^2 + 18g_{xyy} f_{xxy} \\ &\quad - 10g_{xyy} g_{xxx} - 9f_{xxy}^2 - 22f_{xxy} g_{xxx} - 33g_{xxy}^2 + 38g_{xxy} f_{xxx} - 9f_{xxx}^2 - 17g_{xxx}^2) / (3072\omega_0)\end{aligned}$$

These formulas are interesting in the study of degenerate Hopf bifurcations in dynamical systems with the symmetry expressed, as it will be shown in a later example.

Example 3: Van der Pol's equation

This classical example has been studied by Hassard et al. [6], who performed a calculation by hand of the corresponding a_1, b_1, a_2, b_2 , using formulae previously derived in [6], [7]. In order to contrast their results, the algorithm is applied to the equation:

$$\begin{aligned}\dot{x} &= -y + \alpha x^3 \\ \dot{y} &= x\end{aligned}$$

obtaining (after 12 cpu-minutes approximately):

$$a_1 = \frac{3}{8}\alpha, \quad b_1 = 0, \quad a_2 = 0, \quad b_2 = -\frac{27}{256}\alpha^2$$

Observe that the system has the symmetry described in example 2 and is in full accordance with the results obtained in [6].

Example 4: An autonomous electronic oscillator

This example arises from the study of an electronic circuit partially analyzed in [2], [3]. The corresponding state equation is a three-dimensional autonomous system (see [3]). After a center manifold reduction performed by the symbolic algorithm devised in [2], the two-dimensional reduced system takes the following form:

$$\begin{aligned}\dot{x} &= -\omega_0 y + \frac{\eta K_1}{r} x^3 + \frac{\eta K_1}{r} (3\alpha x^5 + 3\beta x^4 y + 3\gamma x^3 y^2 + 3\delta x^2 y^3) \\ \dot{y} &= \omega_0 x + \frac{\eta K_2}{r} x^3 + \frac{\eta K_2}{r} (3\alpha x^5 + 3\beta x^4 y + 3\gamma x^3 y^2 + 3\delta x^2 y^3)\end{aligned}$$

where x, y are related with the state variables corresponding to voltages and currents in the circuit and $\omega_0, r, \eta, K_1, K_2, \alpha, \beta, \gamma, \delta$, are parameters related with its physical configuration. By using the program written in REDUCE, the following coefficients for the normal form are obtained in about 12 cpu-minutes:

$$\begin{aligned}a_1 &= (3\eta K_1)/(8r) \\ a_2 &= (3\eta(2r\omega K_1\gamma + 10r\omega K_1\alpha + 2r\omega K_2\delta + 2r\omega K_2\beta - \eta K_1 K_2))/(32r^2\omega) \\ b_1 &= (3\eta K_2)/(8r) \\ b_2 &= (3\eta(-16r\omega K_1\delta - 16r\omega K_1\beta + 16r\omega K_2\gamma + 80r\omega K_2\alpha - 9\eta K_1^2 - 17\eta K_2^2))/(256r^2\omega)\end{aligned}$$

It is noted that this system possesses the symmetry considered in example 2 and the normal form obtained permits a convenient characterization of the underlying degenerate Hopf bifurcation.

5. Concluding remark

In our opinion this algorithm is a good exponent of the way computer algebra must be guided to a effective long calculation. We have tested the algorithm with different examples, some of them already solved by other means and it has overcome several computational difficulties in previous approaches. The REDUCE 3.2 program used is available on request to authors.

Acknowledgements

This investigation was supported by the Spanish Ministry for Education and Science (MEC) in the frame of CAICYT Project 1102/84.C.3.

Bibliography

- [1] S. Chow and J. K. Hale: *Methods of Bifurcation Theory*. Springer, New York (1982).

- [2] E. Freire, E. Gamero, E. Ponce and L.G. Franquelo: An Algorithm for Symbolic Computation of Center Manifolds. In *Proceedings ISSAC '88, Lecture Notes in Computer Science*, Springer-Verlag, New York, to appear.
- [3] E. Freire, L.G. Franquelo and J. Aracil: Periodicity and Chaos in an Autonomous Electronic System, *IEEE Transactions on CAS*, vol 31, pp 237-247 (1984).
- [4] M. Golubitsky and W.F. Langford: *Classification and Unfoldings of Degenerate Hopf Bifurcations*, J. Diff. Eqns. 41, 375-415 (1981).
- [5] J. Guckenheimer and P. Holmes: *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Appl. Math. Sci. 42, Springer-Verlag, New York (1986).
- [6] B.D. Hassard, N.D. Kazarinoff and Y.-H. Wang: *Theory and Applications of the Hopf Bifurcation*, Cambridge University Press, Cambridge (1980).
- [7] B.D. Hassard and Y.-H. Wang: *Bifurcation formulae derived from center manifold theory*, J. Math. Anal. Appl. 63, 297-312 (1978).
- [8] A.C. Hearn (ed.): *REDUCE 3.2*, The Rand Corporation, Santa Monica (1985).
- [9] I.D. Hsü and N.D. Kazarinoff: *An Applicable Hopf Bifurcation Formula and Instability of Small Periodic Solutions of the Field-Noyes Model*, J. Math. Anal. Appl. 55, 61-89 (1976).
- [10] J.E. Marsden and M. McCracken: *The Hopf Bifurcation and its Applications*, Springer-Verlag, New York (1976).
- [11] K. R. Meyer and D. S. Schmidt: *Entrainment Domains*, Funkcialaj Ekvacioj 20, 171-192 (1977).
- [12] R. H. Rand: Derivation of the Hopf Bifurcation Formula using Lindstedt's Perturbation Method and MACSYMA, in *Applications of Computer Algebra*, R. Pavelle (ed.), Kluwer Academic Publishers, Boston (1985).
- [13] R. H. Rand and D. Armbruster: *Perturbation Method, Bifurcation Theory, and Computer Algebra*, Appl. Math. Sci. 65, Springer-Verlag, New York (1987).
- [14] R. H. Rand and W. L. Keith: Normal Form and Center Manifold Calculations on MACSYMA, in *Applications of Computer Algebra*, R. Pavelle (ed.), Kluwer Academic Publishers, Boston (1985).
- [15] F. Takens: *Unfoldings of certain Singularities of Vector Fields: Generalized Hopf Bifurcations*, J. Diff. Eqns. 14, 476-493 (1973).

Escuela Superior Ingenieros Industriales. Avda. Reina Mercedes. 41012-Sevilla, Spain.

Application of the REDUCE Computer Algebra System to Stability Analysis of Difference Schemes

Victor G. Ganzha *

*Institute of Theoretical and Applied Mechanics
Institutskaya 4/1, 630090 Novosibirsk, USSR*

Institut f. Informatik, Techn. Univ. München, Fed. Rep. Germany

Richard Liska

*Faculty of Nuclear Science and Physical Engineering,
Technical University of Prague, Brehova 7,
115 19 Praha 1, Czechoslovakia*

Abstract

The stability regions of difference schemes approximating systems of linear partial differential equations are automatically obtained by using the Computer algebra system REDUCE and numerical methods for polynomial roots location. The stability analysis is performed by the Fourier method and polynomial root location is based on the Routh algorithm. Several practical examples show the usefulness of the programs described.

Introduction

Many of the tasks from mathematical physics solved by numerical methods need to determine the stability conditions of the difference schemes used [1-3]. As the number of utilised difference schemes is very large, it is necessary to efficiently find the stability conditions of these schemes. Using computer algebra together with numerical methods allows to efficiently obtain practical stability criteria of the investigated difference schemes. At present a number of works [4-5] dealing with these methods of symbolic-numeric interface for the investigation of the stability of difference schemes already exists. Most of these works use the well known Fourier method of stability analysis with von Neumann stability conditions [1-3].

The second method suitable for automation by using computer algebra systems is the method based on obtaining a differential equation which is called the modified equation [15] or the differential approximation [14] of the difference scheme and which is derived from the difference scheme by substituting the Taylor expansion of discrete values into the difference scheme and eliminating the time derivatives from the differential equation obtained. From properties of this differential approximation stability condition are derived. This method can be used heuristically also for non-linear difference schemes [6,16] and in this case has the advantage over other methods in the fact that it takes into account also gradients of functions which are computed.

The analytical-numerical technique for the stability studies realized in ref. [4] has been based on a numerical solution of the dispersion equation with complex coefficients

* Supported by the Alexander-von-Humboldt-Stiftung

where

$$f(\lambda) = 0$$

$$f(\lambda) = \sum_{j=1}^n a_j \lambda^{n-j},$$

$\lambda = e^{-i\omega\tau}$, ω is the frequency in the Fourier harmonic, a_j are complex coefficients, $a_0 \neq 0$, $n \geq 1$. In particular, at $n = 3$ the equation $f(\lambda) = 0$ has been solved in ref. [5] by using Cardano's formulae. In ref. [8] a system IBSTAB has been presented for an automatic stability analysis of difference schemes approximating the initial- and boundary-value problem for hyperbolic systems depending on t and on one spatial variable x . The IBSTAB system combines symbolic manipulations in the LISP language and digital FORTRAN computations, with it the equation $f(\lambda) = 0$ for $n = 3$ and 4 was solved numerically; in particular at $n = 3$ Cardano's formulae were used as in the methods of ref. [5].

In [5] the symbolic-numeric interface for investigating the stability of difference schemes approximating systems of partial differential equations with constant coefficients is described. The automation of the stability analysis algorithms is based on the Fourier method, the Lienard-Chipard criterion and methods of optimization theory and allows to determine the boundaries of a stability region with sufficiently high precision. In this interface, the languages REFAL [20] and FORTRAN are used.

We present here the symbolic-numeric interface for stability analysis of difference schemes based on the Fourier method and Routh algorithm for determining if all roots λ of a polynomial fall into the $\text{Re}\lambda < 0$ halfplane. This interface is divided in three steps. In the first step the amplification matrix of a difference scheme and its characteristic polynomial are computed.

In the second step the characteristic polynomial is transformed by conformal mapping to another polynomial to which the numerical method of locating its roots in order to test stability conditions can be applied. On the base of this transformed polynomial a numerical FORTRAN program is generated which performs the third step by investigating the region of the roots of the polynomial. The numerical program determines for which parameters of the given scheme the stability conditions hold.

Note, that when using the Routh method we do not need to analytically or numerically calculate the principal minors of the Hurwitz matrix, as was done for example in [5,7,17]. Symbolic determinant calculation is nearly always, except for quite small or sparse matrices, a very time and storage consuming task. Our experience shows that using the Routh method for determining stability regions is more effective from the symbolic-numeric computation point of view than using the Routh-Hurwitz or Lienard-Chipard methods. However we have to mention that the Routh-Hurwitz and Lienard-Chipard criterion directly give analytical conditions of stability which can be used for stability testing. These analytical algorithms for polynomial roots locating were implemented in REFAL [20] and in REDUCE by the second author but are not subject of this paper.

The description of the numerical methodology is based on the work [16]. Now we briefly describe the theories used in the three steps of the symbolic-numeric interface.

1. Fourier Stability Analysis of Difference Schemes

One of the most commonly used methods for investigating the stability of difference schemes is the Fourier method of stability analysis [3]. This method offers the possibility of automation of the algebra it needs by computer algebra systems. The Fourier method can be used for stability analysis of any linear, two-level (in time) difference scheme approximating a system of linear partial differential equations.

Let the difference scheme

$$\sum_{j \in N_1} B_1^j \bar{u}_j^{n+1} + \sum_{j \in N_2} B_2^j \bar{u}_j^n = 0 \quad (1)$$

$$j = (j_1, j_2, \dots, j_d), \bar{u}_j^n = (\bar{u}_{j_1}^n, \bar{u}_{j_2}^n, \dots, \bar{u}_{j_d}^n)$$

$$\bar{u}^n = \bar{u}(t_0 + n \Delta t, x_{10} + j_1 \Delta x_1, x_{20} + j_2 \Delta x_2, \dots, x_{d0} + j_d \Delta x_d)$$

be a two-level system of m difference equations in d spatial coordinates $(x_1, x_2, \dots, x_d)$. To perform the Fourier stability analysis of this difference scheme we substitute

$$\bar{u}^n = \bar{u}^0 \exp(ikx_j), \quad \bar{u}_j^{n+1} = \bar{u}^1 \exp(ikx_j)$$

$$x_j = (x_{10} + j_1 \Delta x_1, x_{20} + j_2 \Delta x_2, \dots, x_{d0} + j_d \Delta x_d) \quad (2)$$

$$k = (k_1, k_2, \dots, k_d)$$

into the scheme (1). k is the wave vector of real wave numbers k_l . After dividing the equation by

$$\exp(ik \cdot x_0) = \exp(i(k_1 x_{10} + k_2 x_{20} + \dots + k_d x_{d0}))$$

we obtain

$$H_1 \bar{u}^1 + H_0 \bar{u}^0 = 0, \quad (3)$$

where matrices H_l are given by

$$H_l = \sum_{j \in H_l} B_l^j \exp(i(k_1 j_1 \Delta x_1 + k_2 j_2 \Delta x_2 + \dots + k_d j_d \Delta x_d)), \quad l = 0, 1. \quad (4)$$

Equation (3) can be written as

$$\bar{u}^1 = G \bar{u}^0, \quad G = -H_1^{-1} H_0. \quad (5)$$

Matrix $G(k, \Delta t, \Delta x)$ is the amplification matrix of the difference scheme (1). The von Neumann necessary conditions for stability of the difference scheme (1) is

$$|\lambda_i(k)| \leq 1 + O(\tau), \quad i=1, 2, \dots, p, \quad \text{for all } k \in R^d, \quad (6)$$

where $\lambda_1, \dots, \lambda_p$ are the eigenvalues of the amplification matrix G , i.e. they are roots of characteristic polynomial $f(\lambda)$ of the amplification matrix G :

$$f(\lambda) = \det(\lambda E - G)$$

$$f(\lambda_i) = 0, \quad i=1, 2, \dots, p, \quad (7)$$

where E is the unit matrix. If we do not consider the exponential growth in time of the solution $\bar{u}$, we can write the necessary stability condition as

$$|\lambda_i(k)| \leq 1 + O(\tau), \quad i=1, 2, \dots, p \quad \text{for all } k \in R^d, \quad (8)$$

which is the basic stability condition used in this paper.

2. Conformal mapping

This section treats the conformal mapping between the unit disk and the left halfplane of the complex plane. This mapping is necessary for treating von Neumann stability conditions as Routh-Hurwitz problem.

Let us prove the following assertion: if $\lambda = 1$ is a root of multiplicity k of the polynomial equation $f(\lambda) = 0$ and n is the degree of $f(\lambda)$, $1 \leq k \leq n$, then the polynomial

$$g(z) = (z-1)^n f\left(\frac{z+1}{z-1}\right) \quad (9)$$

has degree $n-k$. Indeed, since λ is a root of multiplicity k of the equation $f(\lambda) = 0$, the polynomial $f(\lambda)$ may be represented in the form

$$f(\lambda) = (\lambda-1)^k f_1(\lambda), \quad (10)$$

where $f_1(\lambda)$ is a polynomial of degree $n_1 = n-k$,

$$f_1(\lambda) = \sum_{j=0}^{n_1} b_j \lambda^{n-k-j}. \quad (11)$$

If $k = n$, this means that all roots of the polynomial $f(\lambda)$ are equal to unity. Consider now the case when $k < n$. Let $\lambda_1, \dots, \lambda_p$ be the roots of the polynomial $f_1(\lambda)$ determined by formula (11) where $1 \leq p \leq n_1$, and let z_j , ($j=1, \dots, p$, $1 \leq p \leq n_1$) be the roots of the polynomial

$$g(z) = (z-1)^{n_1} f_1\left(\frac{z+1}{z-1}\right) \quad (12)$$

It is well known (see, for example, ref. [14]) that all the roots of the polynomial $f_1(\lambda)$ lie in the unit disk in the plane of complex variable if and only if all the roots z_j of the polynomial (12) satisfy the condition $\operatorname{Re}(z_j) < 0$. Thus, with the aid of the conformal mapping $\lambda = \frac{z+1}{z-1}$ the original problem of determining whether all the roots of the polynomial (11) belong to a unit disk reduces to the Routh-Hurwitz problem [9] for the polynomial $g(z)$ defined by formula (12).

3. Numerical locating of polynomial roots by the Routh method

3.1 Routh method

By the conformal mapping described in the previous section the characteristic polynomial $f(\lambda)$ (7) of the amplification matrix G (5) is transformed into the polynomial $g(z)$ (12). The polynomial $g(z)$ is called stable if and only if for all roots z_j of the equation $g(z) = 0$, $\operatorname{Re}(z_j) < 0$. The Routh-Hurwitz problem is the task to determine if a given polynomial is stable or not. So to decide if the stability conditions (8) hold we have to solve the Routh-Hurwitz problem for the polynomial (12). This methodology can also be used for determining the set of difference scheme parameters for which the difference scheme can be stable and namely the boundary of this set.

For solving of the Routh-Hurwitz problem we have chosen the Routh method. The reasons for this choice are discussed in the introduction. To apply the Routh theory [9,10] to the polynomial (12) it is necessary for the coefficient d_0 in $g(z)$ to be positive. Let

$$\tilde{g}(z) = \sum_{v=0}^{n_1} \tilde{d}_v z^{n_1-v}, \quad \tilde{d}_v = d_v \operatorname{sign}(d_0), \quad v=0, \dots, n_1 \quad (13)$$

According to [10] the Routh table

$$\begin{array}{cccc} \tilde{d}_0^{(1)} & \tilde{d}_1^{(1)} & \tilde{d}_2^{(1)} & \dots \\ \tilde{d}_0^{(2)} & \tilde{d}_1^{(2)} & \tilde{d}_2^{(2)} & \dots \\ \dots & \dots & \dots & \dots \\ \tilde{d}_0^{(p)} & \tilde{d}_1^{(p)} & \tilde{d}_2^{(p)} & \dots \end{array} \quad (14)$$

for the polynomial $\tilde{g}(z)$ (13) is built. This table is constructed by the following rules:

1. In the first row of the table (14) are written the even coefficients of the polynomial (13), i.e. $\tilde{d}_0^{(1)} = \tilde{d}_{n_1}$, $\tilde{d}_1^{(1)} = \tilde{d}_{n_1-2}, \dots$
2. In the second row of the table (14) are written the odd coefficients of the polynomial $\tilde{g}(z)$, i.e. $\tilde{d}_0^{(2)} = \tilde{d}_{n_1-1}$, $\tilde{d}_1^{(2)} = \tilde{d}_{n_1-3}, \dots$
3. Elements of the k -th row for $k \geq 3$ are computed from the two previous rows according to the formula

$$\tilde{d}_i^{(k)} = \tilde{d}_{i+1}^{(k-2)} - \frac{\tilde{d}_0^{(k-2)}}{\tilde{d}_0^{(k-1)}} \tilde{d}_{i+1}^{(k-1)} \quad (15)$$

4. Building of the table (14) stops on the p -th row if the number $\tilde{d}_0^{(k+1)}$ is equal to zero.

If during the building of the Routh scheme (14) in the sequence $\tilde{d}_0^{(2)}, \tilde{d}_0^{(3)}, \tilde{d}_0^{(4)}, \dots$ the number $\tilde{d}_0^{(k)} = 0$ is obtained and all elements of the row containing $\tilde{d}_0^{(k)}$ are equal to zero, then this row has to be replaced by the row

$$(n_1-m+1) \tilde{d}_0^{(k-1)}, (n_1-m-1) \tilde{d}_1^{(k-1)}, (n_1-m-1) \tilde{d}_2^{(k-1)}, \dots$$

where $\tilde{d}_0^{(k-1)}, \tilde{d}_1^{(k-1)}, \tilde{d}_2^{(k-1)}, \dots$ are the elements of the previous row and m is the number of rows already obtained. After this replacement the building of the Routh table continues.

The Routh theorem [13] states the relationship between the stability of a polynomial and its Routh table: The polynomial $\tilde{G}(z)$ (13) of degree n_1 with real coefficients and positive mean coefficient $\tilde{d}_0$ is stable if and only if the construction of the Routh table (14) does not stop until the (n_1+1) -th row and all elements of the first column of this table are positive.

3.2 Numerical Realization of the Method

Usually, it is possible to write a difference scheme approximating the system of partial differential equations in the form in which the coefficients of the difference equation depend on certain dimensionless complexes (for example, the Courant number, the Reynolds number, etc.), on dimensionless weight coefficients and on some other dimensionless parameters (for example on the ratio of gas specific heats γ) which may enter the difference scheme. Let us denote all these dimensionless quantities by $\kappa_1, \dots, \kappa_d$, $d \geq 1$. Substituting the right-hand side of (2) into the difference scheme we obtain certain formulas which now contain alongside with $\kappa_1, \dots, \kappa_d$ the quantities $k_1 h_1, \dots, k_N h_N$ where $h_1, \dots, h_N$ are the step sizes of a uniform computational grid along the axes $x_1, \dots, x_N$, respectively. Thus, the coefficients a_j in (2) prove to be dependent on the quantities $\kappa_1, \dots, \kappa_d$ and on the spectral coordinates $\xi_1, \dots, \xi_N$ where $\xi_l = k_l h_l$, $l=1, \dots, N$. Usually, the coefficients a_j prove to be periodic functions of the variables $\xi_1, \dots, \xi_N$ with periods $T_1, \dots, T_N$, respectively. Let us consider in the N -dimensional Euclidean space $(\xi_1, \dots, \xi_N)$ a parallelepiped $G: \{0 \leq \xi_j \leq T_j, j=1, \dots, N\}$. Let us construct

in the domain G a rectangular uniform grid having J_l nodes along the coordinate ξ_l , $l=1, \dots, N$. Denote by R_d the d -dimensional Euclidean space of points $(\kappa_1, \dots, \kappa_d)$. Let the functions $\kappa_1 = \kappa_1(t), \dots, \kappa_d = \kappa_d(t)$ determine parametrically a certain smooth curve in R_d which intersects the boundary of the stability domain of a difference scheme under consideration. Let us introduce by analogy with ref. [7] the function

$$F(t) = \sum_{j_1=1}^{J_1} \dots \sum_{j_N=1}^{J_N} S(\xi_{1,j_1}, \dots, \xi_{N,j_N}, \kappa_1(t), \dots, \kappa_d(t)) - n \prod_{k=1}^N J_k + 0.5, \quad (16)$$

where

$$S(\cdot) = K(\xi_{1,j_1}, \dots, \xi_{N,j_N}, \kappa_1(t), \dots, \kappa_d(t)) + \sum_{l=1}^{n_1} \text{sign } \bar{d}_0^l \quad (17)$$

In the formula (17) $K(\cdot)$ is the multiplicity of the root $\lambda = 1$ of the equation $f(\lambda) = 0$ at the point under consideration $P(\xi_{1,j_1}, \dots, \xi_{N,j_N}, \kappa_1, \dots, \kappa_d)$, $(\xi_{1,j_1}, \dots, \xi_{N,j_N})$ are coordinates of the grid node in the domain G of the space of spectral coordinates $(\xi_1, \dots, \xi_N)$. Denote by D_S the domain in the R_d space at the points of which the roots of the polynomial $f(\lambda)$ determined by (7) satisfy the inequalities $|\lambda_j| \leq 1$, $j=1, \dots, p$, at arbitrary values of the spectral variables ξ_l . It is obvious that if $P \in D_S$ then $F(t) = 0.5$; if $P \notin D_S$, then $F(t) \leq -0.5$. Consequently, the function $F(t)$ changes its sign and, in addition, has a jump at the boundary Γ_S of the domain D_S . Taking into account this behaviour of the function $F(t)$ we have realized the following two-stage algorithm for the calculation of the value t determining that the corresponding point $(\kappa_1, \dots, \kappa_d)$ belongs to the boundary Γ_S . The *first stage* consists of determining the interval $[t_{\min}, t_{\max}]$ at the ends of which the function $F(t)$ has opposite signs. To reduce the number of computation of the function F at this stage the value of t is used that was obtained at the neighbouring point of the boundary Γ_S . If the above interval is not found by the program along the given line $\kappa_1 = \kappa_1(t)$, ..., $\kappa_d = \kappa_d(t)$, then the program prints out a message that along the line under consideration the scheme is stable or instable depending on whether the function (16), (17) is everywhere in the interval $[t_{\min}, t_{\max}]$ positive or negative, respectively. At the *second stage* the bisection process is applied in the interval $[t_{\min}, t_{\max}]$. This process is terminated as soon as the length $b_k - a_k$ of an interval $[a_k, b_k] \subset [t_{\min}, t_{\max}]$ obtained at the k th bisection step becomes less than a given value δ . The quantities $\bar{d}_0^l$ are obtained from the Routh scheme (14).

4. Description of the programs

4.1 Algebraic programs in the REDUCE computer algebra system

All analytical calculations needed in the stability analysis are performed in the REDUCE computer algebra system [18]. Several new statements and operators for this purpose are introduced into the system. Most of the REDUCE programs are written in the symbolic mode of the system, i.e. on the LISP language level and are designed to be easy to use, even for users not familiar with REDUCE. We describe a few new statements and operators.

The statement COORDINATES has the syntax

```
COORDINATES <coordinates> INTO <indices>;
<coordinates> ::= <coordinate> | <coordinate>, <coordinates>
<coordinate> ::= identifier — the name of the coordinate
<indices> ::= <index> | <index>, <indices>
<index> ::= identifier
```

and denotes the indices which will correspond to the coordinates in difference schemes. The name of the time coordinate is supposed to be T . As an example statement,

COORDINATES T, X INTO N, J ;

means that N is the time index, and J is the index of the spatial coordinate X .

The statement UNFUNC declares the names of variables used in the difference scheme. Its syntax is

```
UNFUNC <variables>;
<variables> ::= <variable> | <variable>, <variables>
<variable> ::= identifier — the name of the variable
```

The identifiers used as variables should be declared as operators by the OPERATOR statement and are used in difference schemes as operator with one or more arguments to denote discrete values of functions. Each their argument is an expression built up from one index and integers. If some index is omitted in variable operator arguments, the variable is supposed to be in the point given only by this index. After declaring the variable names

UNFUNC U, V ;

we can use $(U(N+1, J) - U(N+1, J-1)) / HX$ to denote the expression $\frac{u_j^{n+1} - u_{j-1}^{n+1}}{\Delta x}$, or $(U(N+1) - U(N)) / HT$ to denote $\frac{u_j^{n+1} - u_j^n}{\Delta t}$. As only two-level difference schemes are allowed, the time index, here N , can appear in arguments of variable operators only as N or $N+1$. Difference schemes (1) have to be entered into the system in the described manner as elements of a matrix with dimensions $(1, m)$, where m is the number of difference equations in the given scheme.

The operator CHARPOL computes the characteristic polynomial of a given square matrix. The variable λ of the characteristic polynomial (7) is denoted by the identifier LAM. The operator takes one argument — a square matrix — and its value is the polynomial in LAM.

The operator HURW transforms the polynomial in LAM, which is its argument, by conformal mapping (9) to another polynomial again in LAM (z from (9) is also denoted by LAM). This procedure is used to transform the investigated region of polynomial roots from the unit disk to the left halfplane where the Routh method of polynomial roots locating works.

The work of all the statements and operators described here can best be seen in the heavily commented example from the next section which was chosen especially for the purpose of demonstrating how to use the presented programs. The programs are obtainable from the authors on request. Programs for using this method for more than two-level and non-linear (linearization is necessary before stability analysis) difference schemes are in preparation.

4.2 Numerical FORTRAN program for the Routh algorithm

Now we briefly describe the FORTRAN programs that implement the above numerical algorithms for the examination of the stability of difference schemes. FORTRAN programs can be generated in the REDUCE system using the flag ON FORT and the WRITE command. We show the structure of a REDUCE program that generates the FORTRAN program RAUS and describe some input parameters.

```
ON FORT
OUT ofile.for
```

```
WRITE "      SUBROUTINE RAUS";
...
WRITE "      READ 1,NY,NI";
WRITE "      READ 1,NH1,NH2,...";
WRITE "      READ 2,DX,EX";
...
WRITE "      CALL BISEC(EX,K1,NF)";
...
WRITE "      END";
```

```
WRITE "      SUBROUTINE BISEC(EX,X,NF)";
...
WRITE "      X=(A+B)/2";
WRITE "      Y=F(X)";
...
WRITE "      END";
```

```
WRITE "      FUNCTION F(X)";
...
WRITE "      CALL KOEFF(A,X,NY1)";
...
WRITE "      CALL POLDIV(NY1,AW,2,BW,CW,K)";
...
WRITE "      F=SUM+SUM0";
...
WRITE "      END";
```

```
WRITE "      SUBROUTINE KOEFF(CC,X,N)";
...
for i:=1:NC+1 do WRITE A(i):=CC(i-1);
...
WRITE "      END";
```

```
WRITE "      SUBROUTINE POLDIV(D1,C1,C2,C3,D3)";
...
WRITE "      END";
```

```
OFF SHUT
END
```

The result is a FORTRAN source text in file `ofile.for`. The program RAUS consists of four subroutines and one function subprogram. In the main program some variables are read that determine the operation of the program. The variable NY contains the degree of the characteristic polynomial $f(\lambda) = 0$, NI contains the number of nodes in direction of each of the spectral coordinates ξ_i , $i=1, \dots, N$. In the variables NH_i , $i=1, \dots, M$, the number of nodes in each direction κ_i , $i=1, \dots, M$ are stored. The stepsize is DX . If NH_i is 1 for some i (e.g. $i=3$) then the corresponding polynomial depends on κ_1 and κ_2 only. The unknown parameter κ_1 is determined by a process "dichotomie". This algorithm is implemented in a program BISEC. The maximal error of this iteration is EX . The function subprogram $F(x)$ calculates the function $F(t)$ by the above formulae (16-17) and the algorithm RAUS that has been described above. The subroutine KOEFF calculates the coefficients of the characteristic polynomial $F(\lambda)$ that has been obtained in analytic form from the REDUCE program. It is possible that $F(\lambda)$ has a zero at $\lambda = 1$. Then the factor $(\lambda - 1)$ must be cancelled using the subroutine POLDIV.

The result is shown graphically or as a table of the unknowns $\kappa_1, \dots, \kappa_n$.

5. Example

An example demonstrates the practical usefulness of the presented interface. Input to the REDUCE system is denoted by "(I)" and output of the system by "(O)".

As an example we choose the stability analysis of the explicit difference scheme for acoustic equations [6,10]:

$$\begin{aligned} & \frac{u_{m,k}^{n+1} - u_{m,k}^n}{\tau} + \frac{c}{2} \left[\frac{-u_{m-1,k}^n + 2u_{m,k}^n - u_{m+1,k}^n}{h_x} + \right. \\ & \left. \frac{u_{m,k}^{n+1} - u_{m,k}^{n-1}}{h_x} - \frac{u_{m,k-1}^n - 2u_{m,k}^n + u_{m,k+1}^n}{h_y} + \frac{u_{m,k+1}^n - u_{m,k-1}^n}{h_y} \right] = 0, \\ & \frac{u_{m,k}^{2n+1} - u_{m,k}^{2n}}{\tau} + \frac{c}{2} \left[\frac{u_{m+1,k}^{2n} - u_{m-1,k}^{2n}}{h_x} - \frac{u_{m,k}^{2n+1} - 2u_{m,k}^{2n} + u_{m,k}^{2n-1}}{h_x} \right] = 0, \\ & \frac{u_{m,k}^{3n+1} - u_{m,k}^{3n}}{\tau} + \frac{c}{2} \left[\frac{u_{m,k+1}^{3n} - u_{m,k-1}^{3n}}{h_y} - \frac{u_{m,k+1}^{3n} - 2u_{m,k}^{3n} + u_{m,k+1}^{3n-1}}{h_y} \right] = 0. \end{aligned} \quad (18)$$

In this example we present here for demonstration the whole process of determining stability regions, so we proceed with a commented REDUCE dialogue: declaration of coordinates, grids and variables.

```
(I) COORDINATES T,X,Y INTO N,M,K;
(I) GRID UNIFORM,T,X,Y;
(I) UNFUNC U1,U2,U3;
(I) OPERATOR U1,U2,U3;
      definition of the scheme (30); HT,HX,HY are grid steps in coordinates T,X,Y
(I) MATRIX AA(1,3),BB(3,3);
(I) AA(1,1):=(U1(N+1)-U1(N))/HT+C/2*((-U1(M-1)+2*U1(M)-U1(M+1))/HX +
      (U2(M+1)-U2(M-1))/HX - (U1(K-1)-2*U1(K)+U1(K+1))/HY +
      (U3(K+1)-U3(K-1))/HY)$
(I) AA(1,2):=(U2(N+1)-U2(N))/HT+C/2*((U1(M+1)-U1(M-1))/HX -
      (U2(M-1)-2*U2(M)+U2(M+1))/HX)$
(I) AA(1,3):=(U3(N+1)-U3(N))/HT+C/2*((U1(K+1)-U1(K-1))/HY -
      (U3(K-1)-2*U3(K)+U3(K+1))/HY)$
      We do not want to print the matrices H0,H1(4):
(I) OFF PRFORMAT;
      calculation of amplification matrix of the difference scheme:
(I) BB:=AMPMAT AA;
      KX,KY are wave numbers in coordinates X,Y
(O) AX:=KX*HX
(O) AY:=KY*HY
(O) BB(1,1):=(COS(AX)*HT*HY*C + COS(AY)*HT*HX*C -
      HT*HX*C - HT*HY*C + HX*HY)/(HX*HY)
      BB(1,2):=(-SIN(AX)*HT*C*I)/HX
      BB(1,3):=(-SIN(AY)*HT*C*I)/HY
      BB(2,1):=(-SIN(AX)*HT*C*I)/HX
      BB(2,2):=(COS(AX)*HT*C-HT*C+HX)/HX
      BB(2,3):=0
      BB(3,1):=(-SIN(AY)*HT*C*I)/HY
      BB(3,2):=0
      BB(3,3):=(COS(AY)*HT*C-HT*C+HY)/HY
      calculation of the characteristic polynomial of the amplification matrix:
(I) POL:=CHARPOL BB$
      several substitutions useful in this example;
      CAP1,CAP2 are introduced dimensionless quantities
(I) LET
      COS AX=COS AX2**2-SIN AX2**2,
      COS AY=COS AY2**2-SIN AY2**2,
```

```

      SIN AX=2*SIN AX2*COS AX2,
      SIN AY=2*SIN AY2*COS AY2,
      COS AX2**2=1-SIN AX2**2,
      COS AY2**2=1-SIN AY2**2,
      SIN AX2=S1,
      SIN AY2=S2,
      HX=C*HT/CAP1,
      HY=C*HT/CAP2;
statements for formatting output of algebraic expressions
(I) FACTOR LAM;
(I) ORDER C,R,S1,S2;
(I) POL:=POL;
(O) POL:=LAM**3+LAM**2*(4*S1**2*CAP1+4*S2**2*CAP2-3)+LAM*(12*S1**2*
      S2**2*CAP1*CAP2+4*S1**2*CAP1**2-8*S1**2*CAP1+4*S2**2*CAP2**2
      -8*S2**2*CAP2+3)+8*S1**2*S2**2*CAP1**2*CAP2+8*S1**2*S2**2*
      CAP1*CAP2**2-12*S1**2*S2**2*CAP1*CAP2-4*S1**2*CAP1**2+
      4*S1**2*CAP1-4*S2**2*CAP2**2+4*S2**2*CAP2-1
cancelling of the substitutions
(I) CLEAR COS AX,COS AY,SIN AY,COS AX2**2,COS AY2**2,
      SIN AX2,SIN AY2,HX,HY;
testing on the root LAM=1 and eventually transforming complex polynomial to the real one
(I) POL:=COMPLEXPOL POL$
(O) If 8*S1**2*S2**2*CAP1*CAP2*(CAP1+CAP2) = 0,
      a root of the polynomial could be equal to 1
definition of the beginning letters of newly created identifiers during denotation
(I) DENOTID CP;
denotation of the coefficients of the polynomial in LAM (not necessary here,
but present for the purpose of demonstration)
(I) POL:=DENOTERPOL POL;
(O) POL:=LAM**3*CPR03+LAM**2*CPR02+LAM*CPR01+CPR00
performing of the conformal mapping (9)
(I) POL:=HURW POL;
(O) POL:=LAM**3*(CPR00+CPR01+CPR02+CPR03)+
      LAM**2*(-3*CPR00-CPR01+CPR02+ 3*CPR03)+
      LAM*(3*CPR00-CPR01-CPR02+3*CPR03)-
      CPR00+CPR01-CPR02+CPR03
calculation of the coefficients of the polynomial in LAM
(I) ARRAY CC(10),A(10);
(I) NC:=COEFF(POL,LAM,CC);
(O) NC:=3
generation of a part of a FORTRAN numerical program
(I) ON FORT;
(I) OFF ECHO;
(I) OUT ofile;
      writing the values of identifiers created during denotation:
(I) PRDENOT T;
      writing the coefficients of the resulted polynomial:
(I) FOR I:=1:NC+1 DO WRITE A(I):=CC(I-1);

```

The last two statements generate in the file ofile the following FORTRAN program:

```

CPR00=8.*S1**2*S2**2*CAP1**2*CAP2+8.*S1**2*S2**2*CAP1
*CAP2**2-12.*S1**2*S2**2*CAP1*CAP2-4.*S1**2*CAP1**2+
. 4.*S1**2*CAP1-4.*S2**2*CAP2**2+4.*S2**2*CAP2-1.
CPR01=12.*S1**2*S2**2*CAP1*CAP2+4.*S1**2*CAP1**2-8.*
. S1**2*CAP1+4.*S2**2*CAP2**2-8.*S2**2*CAP2+3.
CPR02=4.*S1**2*CAP1+4.*S2**2*CAP2-3.
CPR03=1.
A(1)=-CPR00+CPR01-CPR02+CPR03
A(2)=3.*CPR00-CPR01-CPR02+3.*CPR03
A(3)=-3.*CPR00-CPR01+CPR02+3.*CPR03
A(4)=CPR00+CPR01+CPR02+CPR03

```

The numerical program calculates the border of the stability region in the CAP2, CAP1 plane for difference scheme (18). The dimensionless quantities CAP1, CAP2 are defined by

$$CAP1 = c \frac{\tau}{h_x}, \quad CAP2 = c \frac{\tau}{h_y}.$$

From [4] we know that the stability condition of (18) is $CAP1 + CAP2 \leq 1$. The numerical results of our interface are in very good agreement with this condition.

References

- [1] Samarsky A.A and Gulin A.A., Stability of Difference Schemes. Nauka, Moscow, 1973.
- [2] Roache P.J., Computational Fluid Dynamics, Hermosa, Albuquerque, 1976.
- [3] Richtmyer R.D., Morton K.W., Difference Methods for Initial-Value Problems, 2nd ed., Wiley-Interscience, New York, 1967.
- [4] Vorozhtsov E.V., Ganzha V.G. and Gorsky N.M., Preprint No. 23, Inst. Theor. Appl. Mech., Novosibirsk, 1985.
- [5] Vorozhtsov E.V., Ganzha V.G. and Mazurik S.I., Symbolic-Numerical Interface Investigations by Computers about the Stability of Difference Schemes. Numerical Methods of Fluid Mechanics, vol. 17, No. 5, Novosibirsk, 1986.
- [6] Liska R., Comput. Phys. Commun. 34 (1984) 175.
- [7] Mazurik S.I., Preprint No. 24 Inst. Theor. Appl. Mech., Novosibirsk, 1985.
- [8] Thune M., IBSTAB-A Software System for Automatic Stability Analysis of Difference Methods for Hyperbolic Initial-Boundary Value Problems. Ph.D. Thesis, Uppsala University, Uppsala, 1984.
- [9] Gantmacher F.R., Theory of Matrices, 3rd ed., Nauka, Moscow, 1967.
- [10] Postnikov M.M., Stable Polynomials, Nauka, Moscow, 1981.
- [11] Warming R.F., Beam R.M. and Hyett B.J., Math. Comput. 29 (1975) 1037.
- [12] Hearn A.C., REDUCE Users Manual, Version 3.3, The Rand Corporation, CP 78, Santa Monica 1987.
- [13] Gantmacher F.R., Application of the Theory of Matrices, Interscience Publishers Inc. New York 1959.
- [14] Shokin Y.I., The Method of Differential Approximation, Springer-Verlag, Berlin 1983.
- [15] Hirt C.W., Heuristic stability Theory for Finite-difference equation, J. Comput. Phys. (4), 339-355, 1968.
- [16] Cloutman L.D. and Fullerton L.W., Automated Heuristic Methods for Nonlinear Equations, Los Alamos Scientific Laboratory, Los Alamos 1977.
- [17] Ganzha V.G. and Vorozhtsov E.V., The Stability Analysis of Difference Schemes by Numerical Solution of the Generalised Routh-Hurwitz Problem, Computer Physics Communications 43 (1987) 209-216.
- [18] Mazurik S.I., Algorithms of Solving the Task about Locating Roots of Symbolic Polynomials, Their Realization on Computer and Applications, Institute of Theoretical and Applied Mechanics, No. 24-85, Novosibirsk 1985.
- [19] Wirth M.C., On the Automation of Computational Physics (Ph.D. Thesis), Lawrence Livermore Laboratory, UCRL-52996, Livermore 1980.
- [20] Basic REFAL, A Description of the Language and Fundamental Programming Methods (Systematic Recommendations), 44, Moscow, 1974.

Signs of Algebraic Numbers

Takis Sakkalis

New Mexico State University, Las Cruces

Abstract. *This paper presents an algorithm for the computation of the sign of the value of a polynomial at an algebraic number.*

1. Introduction

An algebraic number x_0 is a real root of an integer polynomial. It is usually given by the following data:

- (i) A square factor free integer polynomial $p(x)$, and
- (ii) A rational interval $[a, b]$, $a < b$, such that $p(a)p(b) \neq 0$ and x_0 is the only root of $p(x)$ in (a, b) .

Let $a_1, a_2, \dots, a_n$ be algebraic numbers defined by $p_1(x_1), p_2(x_1), \dots, p_n(x_n), [\alpha_1, \beta_1], \dots, [\alpha_n, \beta_n]$, $n \geq 1$. In this paper we present an inductive procedure for the computation of the sign of $F(a_1, \dots, a_n)$, where $F(x_1, \dots, x_n) \in \mathbb{Q}[x_1, \dots, x_n]$. Our method is based on the notion of Cauchy index (§2), and the main result of this paper is Proposition 2.6.

2. Preliminaries and the Case $n = 1$

DEFINITION 2.1. Let $R(x)$ be a rational function, and $[a, b]$ a closed interval, $a < b$, so that $R(a)$ and $R(b)$ are finite. The Cauchy index $I_a^b R$ of $R(x)$ over $[a, b]$ is defined as $I_a^b R = N_+^+ - N_+^-$, where N_+^+ and N_+^- denote the number of points in (a, b) at which $R(x)$ jumps from $-\infty$ to $+\infty$ and from $+\infty$ to $-\infty$ respectively, as x increases from a to b . By convention $I_a^b R = -I_b^a R$.

Example 2.2. According to the definition if $R(x) = \sum_{i=1}^m \frac{A_i}{x-a_i} + R_1(x)$, where $A_i, a_i \in \mathbb{R}$, $i = 1, \dots, m$, and $R_1(x)$ is a rational function without real poles, then $I_a^b R = \sum_{a < a_i < b} \text{sign} A_i$. In particular, if $f(x)$ is a real polynomial with $f(a)f(b) \neq 0$, then $I_a^b \frac{f'}{f}$ is simply the number of distinct real roots of $f(x)$ in (a, b) . Moreover, $I_{-\infty}^{+\infty} \frac{f'}{f}$ is the number of distinct real roots of $f(x)$.

One of the methods of calculating the Cauchy index is based on Sturm's theorem. Consider polynomials $r(x), s(x)$ over the reals, and let $R = \frac{r}{s}$, $a, b \in \mathbb{R}$, $a < b$. We are going to compute $I_a^b R$ by constructing a sequence of polynomials $q_1, \dots, q_k$, $q_i \in \mathbb{R}[x]$, using the Euclidean algorithm for finding the greatest common divisor of two polynomials.

(The sequence $q_1, \dots, q_k$ is often called a generalized Sturm sequence for the pair (s, r)). First, we may suppose that $\deg(s) \geq \deg(r)$. For if $\deg(s) < \deg(r)$, we write $r(x) = r_2(x)s(x) + r_1(x)$, $\deg(r_1) < \deg(s)$, and replace r with r_1 since $I_a^b \frac{r}{s} = I_a^b \frac{r_1}{s}$, as Example 2.2 shows. Then, we set $q_1 = s, q_2 = r$ and $q_i = f_i q_{i+1} - q_{i+2}$, $\deg(q_{i+2}) < \deg(q_{i+1})$, $i = 1, \dots, k-2$, $q_k = \gcd(r, s)$. We now have:

THEOREM 2.3 (STURM) [2]. Let $r, s, R, q_1, \dots, q_k$ be as above. Let $V(x)$ denote the number of sign changes in the sequence of numbers $q_1(x), \dots, q_k(x)$, $x \in \mathbb{R}$. Then

$$I_a^b R = I_a^b \frac{r}{s} = V(a) - V(b).$$

Now consider polynomials $p(x, y), q(x, y)$ over $\mathbb{Q}[x, y]$ with no common factors, and let $f = (p, q)$. A point $z_0 = (x_0, y_0) \in \mathbb{R}^2$ is a zero of f if $f(z_0) = (0, 0)$. Let $\Gamma = [a, b] \times [c, d]$, $a < b, c < d$ be a rational rectangle so that no zero of f lies on its boundary $\partial\Gamma$, and $p \cdot q \neq 0$ at its vertices. We set

$$R_3 = \frac{q(x, c)}{p(x, c)}, R_2 = \frac{q(b, y)}{p(b, y)}, R_4 = \frac{q(x, d)}{p(x, d)}, R_1 = \frac{q(a, y)}{p(a, y)}, \text{ and}$$

$$I_\Gamma f = I_a^b R_3 + I_c^d R_2 + I_b^a R_4 + I_d^c R_1.$$

Also, consider the Gauss map $G = \frac{f}{\|f\|} : \partial\Gamma \rightarrow S^1$, where S^1 is the unit circle, and both $\partial\Gamma$ and S^1 carry the counterclockwise orientation. Then the degree d of G is an integer which, roughly speaking, tells how many times $\partial\Gamma$ is wrapped around S^1 by G . We have:

PROPOSITION 2.4. [3]. For G, f, Γ, d as above, $d = -\frac{1}{2} I_\Gamma f$.

Let $J = \frac{\partial p}{\partial x} \frac{\partial q}{\partial y} - \frac{\partial q}{\partial x} \frac{\partial p}{\partial y}$ be the Jacobian determinant of f and z_0 be a zero of f . We say that z_0 is non-degenerate if $J(z_0) \neq 0$. Suppose that all zeros of f which lie in the interior, $\text{Int}\Gamma$, of Γ are non-degenerate. Then the above Proposition yields the following:

COROLLARY 2.5. Under the above considerations,

$$\sum_{\substack{f(z_0)=(0,0) \\ z_0 \in \text{Int}\Gamma}} \text{sign } J(z_0) = -\frac{1}{2} I_\Gamma f.$$

We now proceed with the first step of our inductive procedure. Recall that we are given an algebraic number x_0 defined by $p(x)$, $[a, b]$. Let $F(x) \in \mathbb{Q}[x]$. Our aim is to determine the sign of $F(x_0)$.

First, by replacing F , if necessary, with $F \pm p$ we may suppose that $F(a) \cdot F(b) \neq 0$. Next, we consider $D = \gcd(p, F)$ and for $x \in \mathbb{R}$ we set

$$V_\infty(x) = \begin{cases} 1 & \text{if } p(x) < 0 \\ 0 & \text{otherwise} \end{cases}, \quad V_0(x) = \begin{cases} 1 & \text{if } p(x)F(x) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Finally, define an integer I as follows:

$$I = V_{\infty}(a) - V_0(a) - I_a^b \frac{p}{F} + V_0(b) - V_{\infty}(b).$$

The following provides the basis for the computation of $\text{sign } F(x_0)$.

PROPOSITION 2.6. (i) $F(x_0) = 0$ if and only if $D(a)D(b) < 0$.

(ii) If $D(a)D(b) \geq 0$ then $F(x_0) > 0$, $F(x_0) < 0$ if and only if $I \neq 0$, $I = 0$ respectively.

PROOF: (i) Suppose $D(a)D(b) < 0$. Then there exists a point x' inside (a, b) so that $D(x') = 0$. But since D is the greatest common divisor of p and F , we conclude that $p(x') = F(x') = 0$ and therefore $x' = x_0$ because $[a, b]$ isolates the root x_0 of $p(x)$.

(ii) Let f be the vector field defined by $f = (p(x), y - F(x))$, and let M be a positive integer so that $\max_{a \leq x \leq b} |F(x)| < M$. Also, consider the rectangle $\Gamma = [a, b] \times [0, M]$. First, we observe that $z_0 = (x_0, F(x_0))$ is the only zero of f within the region $a < x < b$. Further, z_0 is non-degenerate since x_0 is a simple root of $p(x)$. A calculation now verifies that $I = -I_{\Gamma} f$, and therefore as Corollary 2.5 shows, z_0 is inside, outside Γ , if and only if $I \neq 0$, $I = 0$, respectively. ■

3. The General Case

Consider algebraic numbers $a_1, \dots, a_n$ given by $p_1(x_1), \dots, p_n(x_n)$, $[\alpha_1, \beta_1], \dots, [\alpha_n, \beta_n]$, $n \geq 2$, $p_i(x_i) \in \mathbb{Z}[x_i]$, $\alpha_i, \beta_i \in \mathbb{Q}$, $i = 1, \dots, n$, and let $F(x_1, \dots, x_n) \in \mathbb{Q}[x_1, \dots, x_n]$. In this section we are going to determine the sign of $F(a_1, \dots, a_n)$ using induction on n ; that is we suppose that given a polynomial $\Phi(x_1, \dots, x_k)$, $k < n$, we can decide the sign of $\Phi(a_1, \dots, a_k)$.

For notation purposes we set $x = (x_1, \dots, x_{n-1})$, $a = (a_1, \dots, a_{n-1})$. Denote by $I = \mathbb{Q}[x]$. We also regard F as a member of $I[x_n]$, and write $F = \sum_j F_j(x) x_n^j$.

We may suppose that $F(a, x_n) \not\equiv 0$, since $F(a, x_n) \equiv 0$ is decidable by induction. We are first going to construct a sequence of polynomials $q_1, \dots, q_k$, $q_i \in I[x_n]$, so that the sequence $q_1(a, x_n), \dots, q_k(a, x_n)$ is a generalized Sturm sequence for the pair $(F(a, x_n), p_n(x_n))$.

To achieve that we invoke the process of pseudo-division in the ring $I[x_n]$. Let F_1, F_2 be non-zero members of $I[x_n]$, $d_i = \deg(F_i)$, $i = 1, 2$, and suppose $d_1 > d_2 > 0$. Then we can find a unique pseudo-quotient $Q = pquo(F_1, F_2)$, and a unique pseudo-remainder $R_1 = prem(F_1, F_2)$ such that $f_2^{\delta+1} F_1 = Q F_2 + R_1$, and $\deg(R_1) < \deg(F_2)$, where f_2 is the leading coefficient of F_2 and $\delta = d_1 - d_2$ [1].

For a polynomial $G = \sum_j G_j(x) x_n^j \in I[x_n]$ we denote by $G^{\alpha} = \sum_{j=0}^{\lambda} G_j(x) x_n^j$, where $\lambda = \deg(G(a, x_n))$. Let now m, d be the degrees of $F(a, x_n)$, $p_n(x_n)$ and assume

that $d \geq m$. We construct $q_1, q_2, \dots, q_k$ inductively as follows: $q_1 = F^a$, $q_2 = [\text{sign } F_m(a)]^{d-m+1} \cdot R_2^a$, where $R_2 = \text{prem}(p_n, F)$, and $q_{i+1} = -[\text{sign } L_i(a)]^{\delta_i+1} \cdot R_{i+1}^a$, $i = 1, \dots, k-1$, and $L_i =$ leading coefficient of q_i , $R_{i+1} = \text{prem}(q_{i-1}, q_i)$, $\delta_i = \deg(q_i) - \deg(q_{i+1})$.

We now proceed with the computation of $\text{sign } F(a, a_n)$. First, we may suppose that $F(a, \alpha_n) \cdot F(a, \beta_n) \neq 0$.

Next we define $V_\infty^n(x_n)$, $V_0^n(x_n)$, I_n , for $x_n \in \mathbb{R}$ as follows:

$$V_\infty^n(x_n) = \begin{cases} 1 & \text{if } p_n(x_n) < 0 \\ 0 & \text{otherwise} \end{cases}, \quad V_0^n(x_n) = \begin{cases} 1 & \text{if } p_n(x_n)F(a, x_n) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$I_n = V_\infty^n(\alpha_n) - V_0^n(\alpha_n) - I_{\alpha_n}^{\beta_n} \frac{p_n(x_n)}{F(a, x_n)} + V_0^n(\beta_n) - V_\infty^n(\beta_n).$$

We observe that if γ is a rational number, the quantities $V_\infty^n(\gamma)$, $V_0^n(\gamma)$, I_n are all computable by the induction hypothesis.

Now Proposition 2.6 yields the following:

Remark 3.1. (i) $F(a, a_n) = 0 \Leftrightarrow q_k(\alpha_n)q_k(\beta_n) < 0$

(ii) If $q_k(\alpha_n)q_k(\beta_n) \geq 0$ then $F(a, a_n) > 0$, $F(a, a_n) < 0 \Leftrightarrow I_n \neq 0$, $I_n = 0$, respectively.

We close this section with an observation. Let $G \in I[x_n]$. Then by determining the sign of $(F - G)(a, a_n)$ we can compare the algebraic numbers $F(a, a_n)$ and $G(a, a_n)$. Moreover, we can calculate the number of distinct real roots of $F(a, x_n)$.

4. An Application

I. Zeros of a Polynomial Vector Field. Let $p(x, y)$, $q(x, y)$ be polynomials over $\mathbb{Q}[x, y]$ with no common factors and consider the vector field $f = (p, q)$. A point $z_0 = (x_0, y_0) \in \mathbb{R}^2$ is called a (real) zero of f if $f(z_0) = (0, 0)$. In this paragraph we describe a method for isolating the zeros of f . Consider $u(x) = \text{Res}_y(p, q)$ and $v(y) = \text{Res}_x(p, q)$ and let $w = (u, v)$. We observe that every zero of f is also a zero of w . However, the converse is not always true. Consider then $z_0 = (x_0, y_0) \in \mathbb{R}^2$ so that $w(z_0) = (0, 0)$. We are going to decide whether $f(z_0) = (0, 0)$. To achieve that, let $\Gamma = [a, b] \times [c, d]$ be a rational rectangle isolating z_0 and let $I = I_\Gamma f$. We observe (Proposition 2.4) that if $I \neq 0$ then $f(z_0) \neq (0, 0)$. On the other hand, if $I = 0$, then by calculating the sign of $f(z_0)$ (§3), we can decide whether z_0 is a zero of f . Finally, by repeating the above procedure over all such rectangles Γ , isolating the zeros of w , we construct k mutually disjoint rectangles, $k \geq 0$, so that each such rectangle contains exactly one zero of f in its interior.

II. A Decision Method for an Algebraic Curve. In [4], A. Seidenberg gave a decision method for an algebraic curve. His method is based on elimination procedures, such as resultants, and an intelligent change of coordinates. In this paragraph we describe another method which is based on the ideas of previous sections. Let $g(x, y)$ be an integer polynomial of degree m , $m > 0$ and let the curve C be defined as $C = \{(x, y) \in \mathbb{R}^2 | g(x, y) = 0\}$. We will then give a procedure which decides whether C is empty.

We may assume that $g(x, y)$ is square factor free and it is not divisible by a non-constant $s(x)$, $s(x) \in \mathbb{R}[x]$. We then consider the following two cases:

- A. m is odd. Let $\phi(x, y)$ be the homogeneous part of $g(x, y)$ of degree m . Consider an integer k so that $\phi(k, 1) \neq 0$, and let $t(y) = g(ky, y)$. We observe that the coefficient of y^m in $t(y)$ is the non-zero constant $\phi(k, 1)$. Therefore, $t(y)$ has at least one real root, since it is of odd degree, and thus C is non-empty.
- B. m is even. In this case we consider $h(x) = \text{Res}_y \left(g, \frac{\partial g}{\partial y} \right)$, and let M be a positive integer so that all real roots of $h(x)$ are inside $(-M, M)$. Then we can determine the number of real roots of $g(-M, y)$ and $g(M, y)$. If both of the above numbers are zero, we note that C is a bounded subset of $\mathbb{R}^2$. If C is empty, we are done. Suppose then that C is non-empty. Then, there exists a real pair $(x_0, y_0) = z_0$ for which $g(z_0) = \frac{\partial g}{\partial y}(z_0) = 0$. To see that it is enough to observe that if C is (real) non-singular then it has at least one component which is diffeomorphic to the unit circle. Now let $f = \left(g, \frac{\partial g}{\partial y} \right)$. But then we can decide (§4.I) whether f has any (real) zeros. That finishes our decision procedure.

We close this section with an example which was carried out using the SCRATCHPAD II Computer Algebra System.

Example. Let $p(x, y) = -y^4 + 4xy^2 + 4x^2y + 2xy + x^4 + x - 1$, $q(x, y) = y^3 + x^2y + x + 4$, and $f = (p, q)$. Then f has two zeros, namely z_1, z_2 , and they are such $z_1 \in [-2, -1] \times [-1, 0]$ and $z_2 \in [1, 2] \times [-2, -1]$.

References

- [1] Brown, W.S. On Euclid's Algorithm and the Computation of Polynomial Greatest Divisors, *JACM*, 18, pp.478-504 (1971).
- [2] Gantmacher, F.R. *Applications of the Theory of Matrices*, Interscience, New York (1959).
- [3] Sakalis, T. The Euclidean Algorithm and the Degree of the Gauss Map, IBM TR, RC-13303, (1987).
- [4] Seidenberg, A. A new decision method for elementary algebra, *Annals of Math.* 60, (1954), pp.365-374.

Efficient Reduction of Quadratic Forms

Neil W. Rickert*

Department of Computer Science
Northern Illinois University, DeKalb, IL 60115

Abstract

The positive definite integer quadratic form, $ax^2 + bxy + cy^2$, is of some importance in number theory. For example such quadratic forms have been shown useful in factorization of large integers. For many applications it is important to be able to recognize when two quadratic forms are equivalent, so it is useful to be able to reduce these quadratic forms to a canonical representation.

For applications in factorization, the quadratic forms used have large coefficients, which must be represented as multiple computer words. This paper shows how to efficiently reduce such multi precision quadratic forms.

1 Introduction

Two quadratic forms $ax^2 + bxy + cy^2$ and $Ax^2 + Bxy + Cy^2$ are said to be *equivalent* if there is a unimodular substitution

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \alpha & -\beta \\ -\gamma & \delta \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} \quad (1)$$

which transforms the form $ax^2 + bxy + cy^2$ into $AX^2 + BXY + CY^2$. By *unimodular* we mean that the matrix coefficients are integers and the determinant is 1. A very readable discussion of quadratic forms may be found in [2].

For economy of notation we follow [2], and refer to the form $ax^2 + bxy + cy^2$ as $[a, b, c]$. A positive definite quadratic form $[a, b, c]$ is said to be *reduced* if either $-a < b \leq a < c$ or $0 \leq b \leq a = c$. As is shown in [2], each positive definite quadratic form is equivalent to exactly one reduced form.

It is sometimes important to determine whether two positive definite quadratic forms are equivalent. The standard approach is to convert the forms to their equivalent reduced forms, a process known as *quadratic reduction*. Once in reduced form a direct comparison of coefficients can be made to test for equivalence. Quadratic reduction has been shown useful in factorization of large integers [3].

In this paper we are interested in the efficient reduction of quadratic forms whose coefficients are too large to fit into a single computer word. This has application to the prime factorization of large integers. Our approach is somewhat analogous to that used Lehmer [1] to compute the GCD of large numbers.

The basic approach is to use the most significant parts of the coefficients as a shorter precision approximation to the multi precision quadratic form. We use the standard reduction

*I am grateful to A. O. L. Atkin for his encouragement in this work. Most of this work was completed at the University of Illinois at Chicago in conjunction with Atkin's project on the factorization of large integers.

procedure on this shorter precision form, and record the substitution matrix as in equation (1) above. The matrix is then applied to the multi-precision quadratic form to partially reduce it. This procedure is repeated until the reduction is complete. However careful analysis is needed, since if at each such shorter precision step we go too far in our reduction, we risk overshooting and thus failing to satisfactorily reduce the multi precision form.

2 Background

We shall henceforth assume that $[a, b, c]$ is positive definite, or equivalently that $4ac - b^2 > 0$, $a > 0$, (whence $c > 0$). The transformation (1) yields the equivalent positive definite form $[A, B, C]$, where $4AC - B^2 = 4ac - b^2$ and

$$\begin{aligned} A &= a\alpha^2 - b\alpha\gamma + c\gamma^2 \\ B &= -2a\alpha\beta + b(\alpha\delta + \beta\gamma) - 2c\gamma\delta \\ C &= a\beta^2 - b\beta\delta + c\delta^2 \end{aligned} \tag{2}$$

For convenience we say that a form is semi-reduced if $|b| < \min(2a, 2c)$. The process of reducing a quadratic form will be carried out in two phases. In the first phase the form is transformed into an equivalent semi-reduced form. In the final phase the semi reduced form is converted to a reduced form. Since most of the computation is involved with the first phase, it is there that we shall concentrate our discussion. However it is convenient to first describe the final phase.

3 Reduction of a semi-reduced form

It is relatively easy to go from a semi-reduced form to a reduced form. The reduction procedure is well known, and can be carried out in several simple stages.

Stage 1. If $|b| \leq \min(a, c)$ we bypass this stage. Otherwise, assume for simplicity's sake that $a \leq c$ and $b \geq 0$ whence $b > a$. Then the substitution $x = X - Y$, $y = Y$ transforms the form $[a, b, c]$ into the form $[A, B, C] = [a, b - 2a, a - b + c]$. Clearly $-A < B < 0$ so that $|B| \leq A$. Likewise $|B| \leq C$ as can be seen by noting that $B + C \geq 0$. If, on the other hand, $a \leq c$ but $b < 0$ the substitution $x = X + Y$, $y = Y$ yields a similar result. If $a > c$ a symmetric substitution can be used.

Stage 2. By now we have reached the stage where $|b| \leq \min(a, c)$. If $a > c$ or if $a = c$, $b < 0$, then the substitution $x = Y$, $y = -X$ transforms the form $[a, b, c]$ into the form $[c, -b, a]$.

Stage 3. The form is now reduced except in the special case of a form $[a, -a, c]$ where $a < c$. The substitution $x = X + Y$, $y = Y$ transforms this into the form $[a, a, c]$, and our positive definite quadratic form is now reduced.

4 Reduction of a form to semi-reduced form

It is convenient to make the assumption that $b \geq 0$. Since the substitution $x = Y$, $y = -X$ transforms the form $[a, b, c]$ into the form $[c, -b, a]$, there is no loss of generality in making this assumption. In practice it is unnecessary to make such a substitution. We merely record the sign of b for later use, and then use the absolute value of b in the reduction to semi-reduced form. Once it is semi-reduced we again make use of the sign of b . Under the assumption that $b \geq 0$, we do our reduction in a manner which maintains this inequality throughout. This turns out to greatly simplify the analysis.

If $a \leq c$, we divide b by $2a$, yielding a quotient q and a remainder r . The substitution $x = X - qY$, $y = Y$ transforms $[a, b, c]$ into the form $[A, B, C] = [a, b - 2aq, c - q(b - aq)]$. If $A \leq C$, our form is then semi-reduced. Similarly if $c < a$ we divide b by $2c$ yielding quotient q' , and make the corresponding substitution $x = X$, $y = -q'X + Y$.

We repeat these procedures until the form is semi-reduced. If, in the process, a or c is changed to a non-positive value, the original form was not positive definite. The positive value of the discriminant $4ac - b^2$ is invariant, and this limits how small a and c may become. Since b is reduced at every step, the process must terminate. This approach is somewhat reminiscent of the Euclidean algorithm for computing the greatest common divisor of two integers.

5 Analysis of the quadratic reduction procedure

At each step in our procedure we can consider the substitutions we are making as special cases of the substitution (1), where the unimodular matrix is either

$$\begin{pmatrix} 1 & -q \\ 0 & 1 \end{pmatrix} \text{ or } \begin{pmatrix} 1 & 0 \\ -q' & 1 \end{pmatrix}$$

We may effectively combine several such steps by multiplying the corresponding matrices, to yield a combined substitution (1), where:

$$\alpha > 0, \beta \geq 0, \gamma \geq 0, \delta > 0. \quad (3)$$

It turns out that substitution (1) and inequalities (3) are all that is needed for an effective analysis of the procedure. Since the matrix in (1) is unimodular, we can easily invert it, yielding:

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} \delta & \beta \\ \gamma & \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad (4)$$

As we have seen, substitution (1) transforms the form $[a, b, c]$ into $[A, B, C]$, where A, B, C are determined by equations (2). Likewise using substitution (4), we see that we can calculate $[a, b, c]$ from $[A, B, C]$ using:

$$\begin{aligned} a &= A\delta^2 + B\gamma\delta + C\gamma^2 \\ b &= 2A\beta\delta + B(\alpha\delta + \beta\gamma) + 2C\alpha\gamma \\ c &= A\beta^2 + B\alpha\beta + C\alpha^2 \end{aligned} \quad (5)$$

Given the inequalities (3), together with our assumption that the computations always proceed so that $A > 0$, $B \geq 0$, $C > 0$ then equations (5) imply that A, B, C are considerably smaller than a, b, c as required. It is not necessary that we follow the exact procedure we described in order for this to be true. We must, however, be careful to proceed so as to maintain the validity of the inequalities we are assuming.

6 Reduction of multiple precision forms

A large integer is represented as a binary number. With the hardware we used (machines in the IBM 370 series) a normal integer word is 32 bits. Our long numbers thus use one word for each 32 bits, and are represented as integer arrays. In a sense we can think of a number

as being represented internally with base 2^{32} , so that each computer word is one digit in this representation.

Assume now that $2a$, b and $2c$ are n -bit numbers; here we allow some of the numbers to have leading zero bits. If binary representation is used it is not necessary to actually compute $2a$ and $2c$; they are already implicitly available in the binary representations of a and c . We use the first k bits of each, and refer to these as $2a_0$, b_0 and $2c_0$. It is convenient to think of a, b, c as having each been divided by 2^{n-k} so that for example b_0 represents the integer part of b and $2a_0$ represents the integer part of $2a$. Then we can write

$$a = a_0 + a_1, \quad b = b_0 + b_1 \text{ and } c = c_0 + c_1$$

where

$$0 \leq a_1 < \frac{1}{2}, \quad 0 \leq b_1 < 1, \quad 0 \leq c_1 < \frac{1}{2} \quad (6)$$

The idea is to (semi-) reduce the form $[a_0, b_0, c_0]$ rather than the original form, but to record the matrix of the substitution (1) used, and then apply the same substitution to the original form $[a, b, c]$. This greatly reduces the number of multiple precision steps required and so results in a significant increase in the speed of reduction. While applying this procedure we must observe a few precautions. Although the quadratic form $[a, b, c]$ is positive definite, it is possible that $[a_0, b_0, c_0]$ is not. If we apply a substitution which transforms $[a_0, b_0, c_0]$ into $[A_0, B_0, C_0]$, it is not sufficient that $B_0 \geq 0$. In order for our analysis to apply we must also ensure that $B \geq 0$, even though we avoid actually computing B at each stage.

At any partial stage we can apply equations (2) and inequalities (6) to see that

$$\begin{aligned} 2A &< 2A_0 + \alpha^2 + \gamma^2 \\ B &\geq B_0 - (\alpha\beta + \gamma\delta) \\ 2C &< 2C_0 + \beta^2 + \delta^2 \end{aligned}$$

While reducing $[A_0, B_0, C_0]$, instead of dividing B_0 by $2A_0$, we divide $B_0 - (\alpha\beta + \gamma\delta)$ by $2A_0 + \alpha^2 + \gamma^2$. This ensures that, if anything, our quotients may be too small, and so guarantees that always $B \geq 0$. We continue working with $[A_0, B_0, C_0]$ until the quotients are 0, meaning that no further progress can be made. At this stage we apply the accumulated matrix substitution to the full numbers, then we again extract the leading parts of each and recommence the procedure.

When we extract the leading k bits of our coefficients, there is always a question as to what should be k . We chose $k = 56$, based on the pragmatic consideration that this number of bits would permit us to still use the double precision floating point divide instruction, thereby simplifying our computation. From equations (5), we conclude that by the time $\alpha, \beta, \gamma, \delta$ have grown to be about 14 bits, the values of A_0, B_0, C_0 will have been reduced to about 28 bit numbers. Since in our division we are using $2A_0 + \alpha^2 + \gamma^2$ in place of $2A_0$, this is about where we would expect to stop making progress. To apply the matrix substitution to the full long numbers entails evaluating the right hand sides of (2). The multipliers here are each products of two matrix entries, so we would expect them to be about 28 bits, which is close to the maximum 31 bit numbers we wish to handle in that stage of reduction. When we do this reduction of the full form, equations (5) again imply that we should cut about 28 bits from each of the coefficients of the full quadratic form.

7 Special considerations

During the reduction process, when dividing b by $2a$ it may occasionally happen that the two numbers are of substantially different orders of magnitude. When this occurs there is little

choice but to do a division using the full lengths of the numbers. This occurs rarely, and signals that this step will make very substantial progress toward the semi-reduced form.

Likewise it may occasionally happen that the two numbers are almost equal in value, so that the leading parts of the numbers are inadequate to find a non-zero quotient. In this case either the form is already semi-reduced, or the quotient is 1, and the computation with the full coefficients is relatively straight forward. This condition also occurs infrequently. It may occur in the last one or two steps of the semi-reduction phase, indicating that we are almost done. If it occurs at other times, it signifies a large reduction in the values of the resulting coefficients, so again substantial progress is made in such a step.

References

- [1] D. H. Lehmer. *Euclid's Algorithm for Large Numbers*. Amer. Math. Monthly 45 (1938) pp. 227-233.
- [2] W. J. Leveque. *Topics in Number Theory*, vol 2. Addison-Wesley, 1956
- [3] D. Shanks. *Class number, a theory of factorization, and genera*. Proc. Symp. Pure Math. 20 (1971) 415-440.

A Story About Computing with Roots of Unity

F.Bergeron*, Dép. Maths et Info,
Université du Québec à Montréal,

Abstract. *In the course of studying idempotents of the group algebra of the symmetric group that characterize Lie elements of the free symmetric algebra, we show how we obtained new unexpected results through computer algebra experiments. This was the direct result of computing in the ring of polynomials modulo the cyclotomic polynomial, instead of computing with roots of unity.*

1. Introduction

In the course of studying the n^{th} homogeneous component of the free Lie algebra over an alphabet A (see [1] and [5]), we were led to investigate certain idempotents of the group algebra, $\mathbb{Q}(\mathbb{S}_n)$, of $\mathbb{S}_n$. Recall that this algebra is the linear span of permutations with the product obtained by linear extension of composition. We wanted, in part, to understand combinatorially why the element

$$\kappa_n = \frac{1}{n} \sum_{\sigma \in \mathbb{S}_n} \omega^{\text{maj}(\sigma)} \sigma,$$

of $\mathbb{Q}(\mathbb{S}_n)$ is an idempotent. Here, ω is a n^{th} primitive root of unity, and $\text{maj}(\sigma)$ stands for the major index of the permutation σ , that is

$$\text{maj}(\sigma) = \sum_{i=1}^{n-1} i \chi(\sigma(i) > \sigma(i+1)).$$

We further considered another idempotent θ_n of this same algebra:

$$\theta_n = \frac{1}{n} \sum_{\sigma \in \mathbb{S}_n} (-1)^{\text{des}(\sigma)} \sigma,$$

with

$$\text{des}(\sigma) = \#\{i \mid \sigma(i) > \sigma(i+1)\}.$$

Then, we wanted to show that multiplication (through the action of the symmetric group on positions)

(* with support from grants A9041 NSERC-Canada, and EQ1608 FCAR-Québec)

of words on A by κ_n gives Lie elements. This also corresponds to proving the identities:

- i) $\theta_n \kappa_n = \theta_n$
- ii) $\kappa_n \theta_n = \kappa_n$.

In the course of experiments with these idempotents and identities using MAPLE, the basic difficulties we encountered were connected with the limitations of the simplification algorithm for expressions involving complex numbers. Our answer to this problem was to substitute computations with complex expressions involving n^{th} primitive roots of unity, by computations in the ring of polynomials $\mathbb{Q}[q]$ in one variable q , modulo the n^{th} cyclotomic polynomial $\phi_n(q)$. This approach had already been considered by others (see [3]), but what is noteworthy in the present case, is that it gave us unexpected new results.

2. The Story

As we have briefly outlined in the previous section, we wanted to deal with expressions of the form

$$\sum_{\sigma \in T} \omega^{\text{maj}(\sigma)},$$

for particular subsets T of $\mathfrak{S}_n$, and ω an n^{th} root of unity. For the choice of subsets that we had in mind, we expected that the value of such an expression would be simple; typically 0, -1 or some power of ω . Thus the simplification problem was crucial to us. Our first experiments, using the built-in simplification procedures of MAPLE, gave expressions filling easily pages of output. Even a simple case such as:

$$\left(e^{\frac{2i\pi}{5}}\right)^4 + \left(e^{\frac{2i\pi}{5}}\right)^3 + \left(e^{\frac{2i\pi}{5}}\right)^2 + \left(e^{\frac{2i\pi}{5}}\right)^1 + 1,$$

we would obtain:

$$\begin{aligned} & 8 \cos(2/5 \text{ Pi})^4 - 6 \cos(2/5 \text{ Pi})^2 + 1 + 4 \cos(2/5 \text{ Pi})^3 - 2 \cos(2/5 \text{ Pi}) \\ & + (8 \cos(2/5 \text{ Pi})^3 \sin(2/5 \text{ Pi}) - 2 \cos(2/5 \text{ Pi}) \sin(2/5 \text{ Pi}) \\ & + 4 \cos(2/5 \text{ Pi})^2 \sin(2/5 \text{ Pi})) I, \end{aligned}$$

instead of the expected value 0. Getting useful information out of this was clearly hopeless. Thus we had to resort to other means.

As is well known, the n^{th} cyclotomic polynomial, $\phi_n(q)$, is monic and irreducible over $\mathbb{Q}$, and by definition every primitive roots of unity is a root of $\phi_n(q)$. Thus, $\phi_n(q)$ is the minimal polynomial for n^{th} primitive roots of unity. Hence, the field $\mathbb{A} = \mathbb{Q}[q]/\phi_n(q)$ of polynomials in q modulo $\phi_n(q)$, is isomorphic to the field $\mathbb{Q}(\omega)$ obtained by adjunction of an n^{th} primitive roots of unity ω to the field $\mathbb{Q}$ of rationals. There is clearly a computable (via Euclid's Algorithm) canonical form for expressions in this field $\mathbb{A}$.

For each integer k , let us define the polynomial $r_k(q)$ to be the (monic) remainder of the division of q^k by $\phi_n(q)$. Now, we shall do computation in the group algebra $\mathbb{A}(\mathbb{S}_n)$ with

$$\kappa_n(q) = \frac{1}{n} \sum_{\sigma \in \mathbb{S}_n} r_{\text{maj}(\sigma)}(q) \sigma,$$

instead of κ_n . The first thing we did check was that $\kappa_n(q)$ (for all n) is indeed an idempotent modulo the cyclotomic polynomial:

$$\kappa_n(q) \kappa_n(q) \equiv_{(\text{mod } \phi_n(q))} \kappa_n(q).$$

But while trying to understand the role of the variable q in this context, we computed the square of $\kappa_n(0)$ (observe that 0 is not a root of $\phi_n(q)$). We were quite surprised when the result came out to be $\kappa_n(0)$. This could only mean that $\kappa_n(q)$ is an idempotent *without* taking modulus:

$$\kappa_n(q) \kappa_n(q) = \kappa_n(q). \quad (1)$$

In fact, further experiments showed that: $\kappa_n(q) \kappa_n(p) = \kappa_n(q)$, with p and q independent variables. This led us to the following observation. Let us expand $\kappa_n(q)$ with respect to powers of q and with coefficients in the group algebra $\mathbb{A}(\mathbb{S}_n)$:

$$\kappa_n(q) = \tau_0 + \tau_1 q + \tau_2 q^2 + \dots + \tau_s q^s,$$

where $s = \deg(\phi_n(q)) - 1$. Then one has:

$$\tau_i \tau_j = \begin{cases} \tau_j, & \text{if } i = 0, \\ 0, & \text{otherwise.} \end{cases}$$

which explains our previous observations.

For another aspect of our study, we wanted to explore the products $\kappa_n(q) \theta_n$ and $\theta_n \kappa_n(q)$. At that time we only used the fact that ω is an n^{th} root of unity, hence we did computations modulo

the polynomial $q^n - 1$. Let us define $\mathcal{K}_n(q)$ in the following manner:

$$\mathcal{K}_n(q) = \frac{1}{n} \sum_{\sigma \in \mathbb{S}_n} q^{\text{maj}(\sigma)} \sigma.$$

Clearly $\kappa_n(q)$ and $\mathcal{K}_n(q)$ are the same modulo $\phi_n(q)$, but as we have just mentioned, we were then working modulo $(q^n - 1)$. We already knew enough about the subject to be able to show that

$$\theta_n \mathcal{K}_n(q) = \frac{(1-q)(1-q^2) \dots (1-q^{n-1})}{n} \theta_n. \quad (2)$$

which gave us the desired identity, since the numerator $(1-q)(1-q^2) \dots (1-q^{n-1})$ is equal to n modulo $\phi_n(q)$. But we did not even know what would be the outcome of $\mathcal{K}_n(q) \theta_n$ other than the fact that $\mathcal{K}_n(\omega) \theta_n = \mathcal{K}_n(\omega)$, for ω an n^{th} primitive root of unity. Thus we tried to reduce the problem to something more manageable in the following way. For a subset $T = \{t_1, t_2, \dots, t_{k-1}\}$ of the set $\{1, 2, \dots, n-1\}$, let D_T stand for the set of permutations:

$$D_T = \{\sigma \in \mathbb{S}_n \mid \sigma(i) \geq \sigma(i+1) \text{ implies } i \in T\}.$$

It is not too hard to show that (2) would follow from the identity (with α any permutation)

$$\sum_{\sigma \in D_T} q^{\text{maj}(\alpha\sigma^{-1})} \equiv_{(\text{mod } \phi_n(q))} 0.$$

But we were really trying at the time to compute these expressions modulo $(q^n - 1)$. Further experimentations suggested the following beautiful evaluation which implies the preceding assertion:

$$\sum_{\sigma \in D_T} q^{\text{maj}(\alpha\sigma^{-1})} \equiv_{(\text{mod } q^n - 1)} q^{\text{maj}(\alpha)} \left[\begin{matrix} n \\ p_1 p_2 \dots p_k \end{matrix} \right]_q. \quad (3)$$

This identity was later shown to be true by M. Wachs (personal communication), by a combinatorial argument involving q -enumeration. Here, the brackets stand for the q -multinomial coefficient, and the p_i 's are as follows: $p_1 = t_1$, $p_2 = t_2 - t_1$, $p_3 = t_3 - t_2$, $\dots$, $p_k = n - t_{k-1}$. The multinomial coefficient in question is equal to 0 modulo the cyclotomic polynomial $\phi_n(q)$.

In fact, we went a little further in our investigations, and through more computer experiments and mathematical manipulations we were led to the following relation:

$$\mathcal{K}_n(q) \theta_n \equiv_{(\text{mod } q^n - 1)} \frac{(1-q)(1-q^2) \dots (1-q^{n-1})}{n} \mathcal{K}_n(q). \quad (4)$$

which we were then able to prove.

3. Conclusion

In this note, we have outlined how we conducted our research with the help of a computer algebra system. As was shown, the use of algebraic devices (working modulo some polynomial) to compensate the weaknesses of the simplification algorithm(s) for complex expressions, gave us much more than just a convenient solution. One of the outstanding effect of this use of MAPLE was the unveiling of unexpected formulas such as (1), (2), (3), and (4). That is why this approach ought to be publicized.

Bibliography

- [1] F. Bergeron, N. Bergeron and A.M. Garsia, *Idempotents for the Free Lie Algebra and q -Enumeration*, (to appear in IMA 1988 Combinatorics Workshop Proceedings, Springer-Verlag).
- [2] A. Bjorner and M. Wachs, *q -Hook Formulas for Trees and Forests*, (to appear in Journ. Combin. Theory, series A).
- [3] C. Dicescenzo and D. Duval, *Algebraic Computation on Algebraic Numbers*, in *Computers and Computing*, Ed.: P.Chenin, C. di Crescenzo and F.Robert, Wiley-Masson, 1986.
- [4] B.W. Char, K.O. Geddes, G.H. Gonnet and S.W. Watt, *MAPLE User's Guide*, WATCOM, 1985.
- [5] A.M. Garsia, *Combinatorics of the Free Lie Algebra and the Symmetric Group*, (to appear in a volume commemorating J. Moser's 60th birthday).

Exact Algorithms for the Matrix-Triangularization Subresultant PRS Method

Alkiviadis G. Akritas
University of Kansas
Department of Computer Science
Lawrence, Kansas, 66045

Abstract. In [2] a new method is presented for the computation of a greatest common divisor (gcd) of two polynomials, along with their polynomial remainder sequence (prs). This method is based on our generalization of a theorem by Van Vleck (1899)[12] and uniformly treats both normal and abnormal prs's, making use of Bareiss's (1968)[4] integer-preserving transformation algorithm for Gaussian elimination; moreover, for the polynomials of the prs's, this method provides the smallest coefficients that can be expected without coefficient gcd computations. In this paper we present efficient, exact algorithms for the implementation of this new method, along with an example where bubble pivot is needed.

1. Introduction

In this note we restrict our discussion to univariate polynomials with integer coefficients and to computations in $\mathbb{Z}[x]$, a unique factorization domain. Given the polynomial $p(x) = c_n x^n + c_{n-1} x^{n-1} + \dots + c_0$, its degree is denoted by $\deg(p(x))$ and c_n , its leading coefficient, by $\text{lc}(p)$; moreover, $p(x)$ is called *primitive* if its coefficients are relatively prime.

Consider now $p_1(x)$ and $p_2(x)$, two primitive, nonzero polynomials in $\mathbb{Z}[x]$, $\deg(p_1(x)) = n$ and $\deg(p_2(x)) = m$, $n \geq m$. Clearly, the polynomial division (with remainder) algorithm, call it **PD**, that works over a field, cannot be used in $\mathbb{Z}[x]$ since it requires exact divisibility by $\text{lc}(p_2)$. So we use *pseudo-division*, which always yields a pseudo-quotient and pseudo-remainder; in this

process we have to premultiply $p_1(x)$ by $lc(p_2)^{n-m+1}$ and then apply algorithm PD. Therefore we have:

$$lc(p_2)^{n-m+1}p_1(x) = q(x)p_2(x) + p_3(x), \quad \deg(p_3(x)) < \deg(p_2(x)). \quad (1)$$

Applying the same process to $p_2(x)$ and $p_3(x)$, and then to $p_3(x)$ and $p_4(x)$, etc. (Euclid's algorithm), we obtain a *polynomial remainder sequence* (prs)

$$p_1(x), p_2(x), p_3(x), \dots, p_h(x), p_{h+1}(x) = 0,$$

where $p_h(x) \neq 0$ is a greatest common divisor of $p_1(x)$ and $p_2(x)$, $\gcd(p_1(x), p_2(x))$. If $n_i = \deg(p_i(x))$ and we have $n_i - n_{i+1} = 1$, for all i , the prs is called *normal*, otherwise, it is called *abnormal*. The problem with the above approach is that the coefficients of the polynomials in the prs grow exponentially and hence slow down the computations. We wish to control this coefficient growth. We observe that equation (1) can also be written more generally as

$$lc(p_{i+1})^{n_i - n_{i+1} + 1} p_i(x) = q_i(x)p_{i+1}(x) + \beta_i p_{i+2}(x), \quad \deg(p_{i+2}(x)) < \deg(p_{i+1}(x)), \quad (2)$$

$i = 1, 2, \dots, h-1$. That is, if a method for choosing β_i is given, the above equation provides an algorithm for constructing a prs. The obvious choice $\beta_i = 1$, for all i , is called the *Euclidean prs*; it was described above and leads to exponential growth of coefficients. Choosing β_i to be the greatest common divisor of the coefficients of $p_{i+2}(x)$ results in the *primitive prs*, and it is the best that can be done to control the coefficient growth. (Notice that here we are dividing $p_{i+2}(x)$ by the greatest common divisor of its coefficients before we use it again.) However, computing the greatest common divisor of the coefficients for each member of the prs (after the first two, of course) is an expensive operation and should be avoided. So far, in order both to control the coefficient growth and to avoid the coefficient gcd computations, either the *reduced* or the (improved) *subresultant prs* have been used. In the reduced prs we choose

$$\beta_1 = 1 \text{ and } \beta_i = lc(p_i)^{n_i - n_{i+1} + 1}, \quad i = 2, 3, \dots, h-1, \quad (3)$$

whereas, in the subresultant prs we have

$$\beta_1 = (-1)^{n_1 - n_2 + 1} \text{ and } \beta_i = (-1)^{n_i - n_{i+1} + 1} lc(p_i) H_i^{n_i - n_{i+1}}, \quad i = 2, 3, \dots, h-1, \quad (4)$$

where

$$H_2 = lc(p_2)^{n_1 - n_2} \text{ and } H_i = lc(p_i)^{n_{i-1} - n_i} H_{i-1}^{1 - (n_{i-1} - n_i)}, \quad i = 3, 4, \dots, h-1.$$

That is, in both cases above we divide $p_{i+2}(x)$ by the corresponding β_i before we use it again. The reduced prs algorithm is recommended if the prs is normal, whereas if the prs is abnormal the subresultant prs algorithm is to be preferred. The proofs that the β_i 's shown in (3) and (4) exactly divide $p_{i+2}(x)$ are very complicated [7] and have up to now obscured simple divisibility properties [10], (see also [5] and [6]). For a simple proof of the validity of the reduced prs see [1]; analogous proof for the subresultant prs can be found in [8].

In contrast with the above prs algorithms, the matrix-triangularization subresultant prs method avoids explicit polynomial divisions (explained below). In what follows we present efficient, exact algorithms for the implementation of this method. We also present an example where bubble pivot is needed.

2. Gaussian elimination and Sylvester's form of the resultant

Consider the two polynomials in $Z[x]$, $p(x) = c_n x^n + c_{n-1} x^{n-1} + \dots + c_0$ and $p_2(x) = d_m x^m + d_{m-1} x^{m-1} + \dots + d_0$, $c_n \neq 0$, $d_m \neq 0$, $n \geq m$. Contrary to established practice, we choose to call Sylvester's form of the resultant of $p_1(x)$ and $p_2(x)$ the one described below; this form was "buried" in Sylvester's 1853 paper [11] and is only once mentioned in the literature in a paper by Van Vleck [12]. Sylvester indicates ([11], p.426) that he had produced this form in 1839 or 1840 and some years later Cayley unconsciously reproduced it as well. It is Sylvester's form of the resultant that forms the foundation of our new method for computing polynomial remainder sequences; however, we first present the following theorem concerning Bruno's form of the resultant (the form encountered most often in the literature under the Sylvester's name):

Theorem 1 (Laidacker[9]). If we transform the matrix corresponding to $\text{res}_B(p_1(x), p_2(x))$ into its upper triangular form $T_B(R)$, using row transformations only, then the last nonzero row of $T_B(R)$ gives the coefficients of a greatest common divisor of $p_1(x)$ and $p_2(x)$.

The above theorem indicates that we can obtain only a greatest common divisor of $p_1(x)$ and $p_2(x)$ but none of the remainder polynomials. In order to compute both a $\gcd(p_1(x), p_2(x))$ and all the polynomial remainders we have to use Sylvester's form of the resultant; this is of order $2n$ (as opposed to $n+m$ for the other forms) and of the following form ($p_2(x)$ has been transformed into a polynomial of degree n by introducing zero coefficients):

$$\text{res}_S(p, q) = \begin{bmatrix} c_n & c_{n-1} & \dots & c_0 & 0 & 0 & \dots & 0 \\ d_n & d_{n-1} & \dots & d_0 & 0 & 0 & \dots & 0 \\ 0 & c_n & \dots & c_0 & 0 & 0 & \dots & 0 \\ 0 & d_n & \dots & d_0 & 0 & 0 & \dots & 0 \\ & & \dots & & & & & \\ 0 & \dots & 0 & c_n & c_{n-1} & \dots & c_0 \\ 0 & \dots & 0 & d_n & d_{n-1} & \dots & d_0 \end{bmatrix} \quad (S)$$

In general, if we have the polynomial remainder sequence $p_1(x), p_2(x), p_3(x), \dots, p_h(x)$, $\deg(p_1(x)) = n$, $\deg(p_2(x)) = m$, $n \geq m$, we can obtain the (negated) coefficients of the $(i+1)$ th member of the prs, $i = 0, 1, 2, \dots, h-1$, as minors formed from the first $2i$ rows of (S) by successively associating with the first $2i - 1$ columns (of the $(2i)$ by $(2n)$ matrix) each succeeding column in turn.

On the other hand, we transform the matrix corresponding to the resultant (S) into its upper triangular form using Bareiss's integer-preserving transformation algorithm [4]. That is: let $r_{00}^{(-1)} = 1$, and $r_{ij}^{(0)} = r_{ij}$, $i, j = 1, \dots, n$; then for $k < i, j \leq n$,

$$r_{ij}^{(k)} := (1 / r_{k-1, k-1}^{(k-2)}) \cdot \begin{vmatrix} r_{kk}^{(k-1)} & r_{kj}^{(k-1)} \\ r_{ik}^{(k-1)} & r_{ij}^{(k-1)} \end{vmatrix} \quad (5)$$

Of particular importance in Bareiss's algorithm is the fact that the determinant of order 2 is divided *exactly* by $r_{k-1, k-1}^{(k-2)}$ (the proof is very short and clear and is described in Bareiss's paper [4]) and that the resulting coefficients are the smallest that can be expected without coefficient gcd computations and without introducing rationals. Notice how all the complicated expressions for β_i in the reduced and subresultant prs algorithms are mapped to the simple factor $r_{k-1, k-1}^{(k-2)}$ of this method.

It should be pointed out that using Bareiss's algorithm we will have to perform pivots (interchange two rows) which will result in a change of signs. We also define the term *bubble pivot* as follows: if the diagonal element in row i is zero and the next nonzero element down the column is in row $i+j$, $j > 1$, then a row $i+j$ will become row i after pairwise interchanging it with the rows above it. Bubble pivot preserves the symmetry of the determinant.

We have the following theorem.

Theorem 2 ([2]). Let $p_1(x)$ and $p_2(x)$ be two polynomials of degrees n and m respectively, $n \geq m$. Using Bareiss's algorithm transform the matrix corresponding to $\text{res}_S(p_1(x), p_2(x))$ into its upper triangular form $T_S(R)$; let n_i be the degree of the polynomial corresponding to the i th row of $T_S(R)$, $i = 1, 2, \dots, 2n$, and let $p_k(x)$, $k \geq 2$, be the k th member of the (*normal or abnormal*) polynomial remainder sequence of $p_1(x)$ and $p_2(x)$. Then if $p_k(x)$ is in row i of $T_S(R)$, the coefficients of $p_{k+1}(x)$ (within sign) are obtained from row $i+j$ of $T_S(R)$, where j is the smallest integer such that $n_{i+j} < n_i$. (If $n = m$ associate both $p_1(x)$ and $p_2(x)$ with the first row of $T_S(R)$.)

Notice that as a special case of the above theorem we obtain Van Vleck's theorem for normal prs's. We see, therefore, that based on Theorem 2, we have a new method to compute the polynomial remainder sequence and a greatest common divisor of two polynomials. This new method uniformly treats both normal and abnormal prs's and provides the smallest coefficients that can be expected without coefficient gcd computation.

3. Our method and its implementation

The inputs are two (primitive) polynomials in $\mathbb{Z}[x]$, $p_1(x) = c_n x^n + c_{n-1} x^{n-1} + \dots + c_0$ and $p_2(x) = d_m x^m + d_{m-1} x^{m-1} + \dots + d_0$, $c_n \neq 0$, $d_m \neq 0$, $n \geq m$.

Step 1: Form the resultant (S) , $\text{res}_S(p_1(x), p_2(x))$, of the two polynomials $p_1(x)$ and $p_2(x)$.

Step 2: Using Bareiss's algorithm (described above) transform the resultant (S) into its upper triangular form $T_S(R)$; then the coefficients of all the members of the polynomial remainder sequence of $p_1(x)$ and $p_2(x)$ are obtained from the rows of $T_S(R)$ with the help of Theorem 2.

For this method we have proved [2] that its computing time is:

Theorem 3. Let $p_1(x) = c_n x^n + c_{n-1} x^{n-1} + \dots + c_0$ and $p_2(x) = d_m x^m + d_{m-1} x^{m-1} + \dots + d_0$, $c_n \neq 0$, $d_m \neq 0$, $n \geq m$ be two (primitive) polynomials in $\mathbb{Z}[x]$ and for some polynomial $P(x)$ in $\mathbb{Z}[x]$ let $|P|_\infty$ represent its maximum coefficient in absolute value. Then the method described above computes a greatest common divisor of $p_1(x)$ and $p_2(x)$ along with all the polynomial remainders in time

$$O(n^5 L(|p|_\infty)^2)$$

where $|p|_\infty = \max(|p_1|_\infty, |p_2|_\infty)$.

Below we present efficient exact (maple-like) algorithms for the matrix-triangularization subresultant prs method. A subalgorithm call is the name of the subalgorithm in all **bold** letters. All subalgorithm calls are from the main algorithm. Parameters (arguments) are not shown. Comments are made within braces $\{\}$. An explanation of the variables is found after the algorithms.

```

start                {deg(p1(x)) >= deg(p2(x))}
  initialize         {set resultant matrix to zero, and initialize the variables used}
  getpolys           {get coefficients of the first two polynomials}
  buildmatrix        {build the matrix corresponding to Sylvester's form of the resultant}
  set k to 1          {k is the index for the transformation loop}
  while k < n do      {loop n-1 times, unless gcd is found (see pivot)}
    if (r[k,k] = 0) then pivot fi {need to put a non-zero element into r[k,k]}
    if k < n then      {in pivot, if gcd is found k is set to n+1}
      do transform; set d to r[k,k] od
    fi
    set k to k+1 {increment main loop index}
  od
end

initialize
  n1 := deg(p1(x)); {deg(p1(x)) >= deg(p2(x))}
  n := 2*n1;
  for i from 1 to n do
    for j from 1 to n do
      r[i,j] := 0 {see notes on variables below}
    od
    tran[i] := false {see notes on variables below}
  od
  d := 1 {no division for first transformation}
end

getpolys
{this is dependent on the language used and whether the program is interactive or reads data from a
data file; the function coeff(p(x), i) computes the coefficient of xi in the polynomial p(x)}
  for i from 1 to n1+1 do
    r[1,i] := coeff(p1(x), n1+1-i) {put the coefficients of p1(x) in row 1 of the matrix}
    r[2,i] := coeff(p2(x), n1+1-i) {put the coefficients of p2(x) in row 2; remember that we have
                                     to include leading zero if deg(p2(x)) < deg(p1(x))}
  od
end

```

buildmatrix

```

k := 2;
for i from 3 to n do           {loop to put values in rows 3 to n}
  for j from k to n do       {loop to put values across each row}
    r[i,j] := r[i-2, j-1]
  od
  if (i mod 2) = 0 then k := k+1 fi
od
{the following will build array L; L[i] is the location of the last polynomial element in row i}
j := 1;
for i from 1 to n1 do
  L[j] := i + n1;    {last position is based on first plus degree}
  L[j+1] := i + n1;
  j := j + 2        {go down two rows}
od
end

```

pivot

```

{check across row k for all zeros, this means row k-1 is gcd}
ck4gcd := true;
i := k+1;           {i is the index for loop}
while (i <= L[k]) and ck4gcd do {loop across row}
  if r[k,i] <> 0 then ck4gcd := false fi
  i := i + 1        {increment loop index}
od
if ck4gcd then      {need to zero matrix below row k and stop processing}
  for i from k+1 to n do
    for j from k to n do
      r[i,k] := 0
    od
  od
  k := n + 1        {this stops main loop}
else                {need to find a row s without a zero in column k to pivot up}
  s := k + 1        {start looking one row below k}
  while r[s,k] = 0 do {loop while value in column k is zero}
    s := s + 1
  od
  {move row s to row k with bubble pivot}

```

```

    tempbool := trans[s];           {need to pivot tran with rows}
    tempint := L[s];
    for j from 1 to n do temprow[j] := r[s,j] od;
    for i from s by -1 to k + 1 do {this needs to step backwards (s is > k+1)}
        tran[i] := tran[i-1];
        L[i] := L[i-1];
        for j from 1 to n do r[i,j] := r[i-1,j] od
    od;
    tran[k] := tempbool;
    L[k] := tempint;
    for j from 1 to n do r[k,j] := temprow[j] od;
fi
end

```

transform

{Find the last row s with a non-zero element in column k or the last row which has been transformed (whichever is higher)}

```

    s := k;
    for i from k+1 to n do
        if (r[i,k] <> 0) or tran[i] then s := i fi
    od
    {s is now the last row with a non-zero element in column k}
    for i from k + 1 to s do          {loop through all rows up to s}
        for j from k + 1 to L[i] do  {loop across row to last element}
            if tran[k] and tran[i] and (d <> 1) then
                {okay to divide as you transform row i}
                r[i,j] := iquo(r[k,k] * r[i,j] - r[i,k] * r[k,j],d)
                {iquo(m,n) computes the integer quotient of m divided by n}
            else
                r[i,j] := r[k,k] * r[i,j] - r[i,k] * r[k,j]
            fi
        od;
        r[i,k] := 0;                 {need to zero column k below row k}
        tran[i] := true              {row i has been transformed}
    od
end

```

printmatrix

{this is dependent on the language used; print each row and column}

```
for i from 1 to n do
  for j from 1 to n do
    write r[i,j]    {on one line}
  od;
  advance a line
od
end
```

The variables

1. $r[i,j]$ is a two dimensional matrix (array).
2. $n_1 = \deg(p_1(x))$.
3. $n = 2*n_1$ is the length and width of the resultant (matrix).
4. $L[i]$ is the location of the last element in row i ; this is important because it is used so that we do not update the zero elements of a row.
5. $\text{tran}[i]$ is a one dimensional boolean (or logical) array; it is true when row i was transformed during the last transformation; this is important since only transformed rows may be divided by d .
6. d is the value which a transformed row may be divided by if all other factors allow for division. In the Bareiss transform d is $r[k-1,k-1]$.
7. k is the current transformation number and $r[k,k]$ is the corner element where the next transformation will begin.
8. tempint , tempbool and temprow are temporary variables used for pivoting.
9. ck4gcd is a boolean (logical) variable which will be true when row k is all zeros. This means a greatest common divisor (gcd) has been found and further transformations are not necessary.

Below we present an incomplete example where bubble pivoting is needed [3]; note that there is a difference of 3 in the degrees of the members of the prs, as opposed to a difference of 2 in Knuth's "classic" incomplete example.

Example. Let us find the polynomial remainder sequence of the polynomials $p_1(x) = 3x^9 + 5x^8 + 7x^7 - 3x^6 - 5x^5 - 7x^4 + 3x^3 + 5x^2 + 7x - 2$ and $p_2(x) = x^8 - x^5 - x^2 - x - 1$. This incomplete prs example presents a variation of three in the degrees of its members (from 7 to 4) and it requires a bubble pivot in the matrix-triangularization method; that is, a pivot will take place between rows that are not adjacent.

The matrix-triangularization subresultant prs method

row		degree
1>	3 5 7 -3 -5 -7 3 5 7 -2 0 0 0 0 0 0 0 0	(9)
2>	0 1 0 0 -1 0 0 -1 -1 -1 0 0 0 0 0 0 0 0	(8)
3)	0 0 5 7 0 -5 -7 6 8 10 -2 0 0 0 0 0 0 0	(8)
4>	0 0 0 -7 0 0 7 -6 -13 -15 -3 0 0 0 0 0 0 0	(7)
5)	0 0 0 0 -49 0 0 79 23 19 -55 14 0 0 0 0 0 0	(7)
#6)	0 0 0 0 0 -343 0 -24 501 73 93 -413 98 0 0 0 0 0	(7)
#7)	0 0 0 0 0 0 -2401 -510 -1273 1637 -339 56 -2891 686 0 0 0 0	(7)
8>	0 0 0 0 0 0 0 2058 4459 7546 3430 2401 0 0 0 0 0 0	(4)
9)	0 0 0 0 0 0 0 0 -1764 -3822 -6468 -2940 -2058 0 0 0 0 0	(4)
10)	0 0 0 0 0 0 0 0 0 1512 3276 5544 2520 1764 0 0 0 0	(4)
11)	0 0 0 0 0 0 0 0 0 0 25811 -18982 4520 -811 -3024 0 0 0	(4)
12>	0 0 0 0 0 0 0 0 0 0 0 -64205 -77246 -37568 -28403 0 0 0	(3)
13)	0 0 0 0 0 0 0 0 0 0 0 2124693 449379 519299 128410 0 0	(3)
14>	0 0 0 0 0 0 0 0 0 0 0 0 10853 -1800739 -2018639 0 0	(2)
15)	0 0 0 0 0 0 0 0 0 0 0 0 0 -22909248 -24412716 10481706 0	(2)
16>	0 0 0 0 0 0 0 0 0 0 0 0 0 0 -40801132 47620330 0	(1)
17)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -398219984 81602264	(1)
18>	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 682427564	(0)

Largest integer generated is 27843817119202448 [17 digits].

Pivoted row 6 during transformation 6. Stored row is:

6> 0 0 0 0 0 0 0 42 91 154 70 49 0 0 0 0 0 0 (4)

Pivoted row 7 during transformation 7. Stored row is:

7) 0 0 0 0 0 0 0 294 637 1078 490 343 0 0 0 0 0 0 (4)

Bibliography

- [1] Akritas, A.G.: *A simple validity proof of the reduced prs algorithm*. Computing 38, 369-372, 1987.
- [2] Akritas, A.G.: *A new method for computing greatest common divisors and polynomial remainder sequences*. Numerische Mathematik 52, 119-127, 1988.
- [3] Akritas, A.G.: *Elements of Computer Algebra with Applications*. John Wiley, New York, in press.
- [4] Bareiss, E.H.: *Sylvester's identity and multistep integer-preserving Gaussian elimination*. Mathematics of Computation 22, 565-578, 1968.
- [5] Brown, W.S.: *On Euclid's algorithm and the computation of polynomial greatest common*

divisors. JACM 18, 476-504, 1971.

- [6] Brown, W.S.: *The subresultant prs algorithm*. ACM Transactions On Mathematical Software 4, 237-249, 1978.
- [7] Collins, G.E.: *Subresultants and reduced polynomial remainder sequences*. JACM 14, 128-142, 1967.
- [8] Habicht, W.: *Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens*. Commentarii Mathematici Helvetici 21, 99-116, 1948.
- [9] Laidacker, M.A.: *Another theorem relating Sylvester's matrix and the greatest common divisor*. Mathematics Magazine 42, 126-128, 1969.
- [10] Loos, R.: *Generalized polynomial remainder sequences*. In: Computer Algebra Symbolic and Algebraic Computations. Ed. by B. Buchberger, G.E. Collins and R. Loos, Springer Verlag, Wien, New York, 1982, Computing Supplement 4, 115-137.
- [11] Sylvester, J.J.: *On a theory of the syzygetic relations of two rational integral functions, comprising an application to the theory of Sturm's functions, and that of the greatest algebraical common measure*. Philosophical Transactions 143, 407-548, 1853.
- [12] Van Vleck, E. B.: *On the determination of a series of Sturm's functions by the calculation of a single determinant*. Annals of Mathematics, Second Series, Vol. 1, 1-13, 1899-1900.

Computation of Fourier Transforms on the Symmetric Group

Daniel Rockmore¹
Harvard University
Department of Mathematics

Abstract Let G be a finite group and f any complex-valued function defined on G . If ρ is a matrix representation of G then the Fourier transform of f at ρ is defined as the matrix $\sum_{s \in G} f(s)\rho(s)$. Various applications demand the computation of the Fourier transforms of f at all irreducible representations of G . Direct computation of all such Fourier transforms requires on the order of $|G|^2$ arithmetic operations.

In earlier work with Diaconis ([DR]) ideas have been presented for more efficient methods of computing Fourier transforms. In particular, for S_n several algorithms were sketched. This paper describes in detail a running implementation of one of these algorithms which has been used effectively on a VAX11/750 and a SUN4.

1. Introduction

Let G be a finite group, f a complex-valued function defined on G , and let ρ be a matrix representation of G . Then the Fourier transform of f with respect to ρ is defined to be the matrix

$$\hat{f}(\rho) = \sum_{s \in G} f(s)\rho(s).$$

Direct computation of $\hat{f}(\rho)$ for all irreducible representations ρ of G requires on the order of $|G|^2$ arithmetic operations. In [DR] Diaconis and Rockmore take advantage of the group structure to develop more efficient algorithms for this computation.

In brief, the basic idea is as follows. If ρ is an irreducible representation of G , consider ρ restricted to some fixed subgroup H . As a representation of H , ρ will split into irreducible representations. Thus, with respect to an appropriate basis, ρ (restricted to H) can be written in block diagonal form with irreducible representations of H forming the blocks. The Fourier transform at ρ can thus be built up as a direct sum of transforms over the subgroup. This process can be iterated. It yields a family of algorithms, all of which take fewer operations than direct computation. When specialized to abelian groups this idea gives the well-known Cooley-Tukey algorithm.

The ability to carry out this plan depends mainly on the existence of matrix representations of G that split (ie. become block diagonal) as they are restricted down a tower of subgroups of G . Such is the case for the symmetric group S_n and its natural tower of subgroups,

$$S_n \supseteq S_{n-1} \supseteq \dots \supseteq S_1 = \{\text{identity}\}$$

¹Supported by IBM and NSF Graduate Fellowships

where

$$S_k = \{\pi \in S_n \mid \pi(j) = j, k < j \leq n\}.$$

Here, at least two classes of representations discovered by Alfred Young, Young's semi-normal and orthogonal representations, have this "splitting property". In addition, the way in which irreducible representations of S_n decompose when restricted down this tower (the branching theorem) is also well-understood.

In the context of analyzing ranked data in an election, computation of all Fourier transforms of an appropriately defined function on S_n is required ([D1]). This is an example of "spectral analysis" for data on groups, one of a host of applications of noncommutative Fourier analysis developed by Diaconis ([D2]).

Thus, interest in actually computing Fourier transforms over S_n has led to the implementation of algorithms on the computer. In [DR] several algorithms for computing Fourier transforms over S_n are sketched. This paper discusses a running implementation of the algorithm described in [DR] as using "complete branching and partial storage". Listings of the working C code may be obtained upon request.

In the interest of being as self-contained as possible, in section 2 an extremely brief introduction to the representation theory of the symmetric group is given. All of this and much more may be found in James and Kerber's encyclopedic study ([JK]). Section 3 contains the main ideas that have shaped the program, describing the scheduling of the computation and copying of the restricted transforms. In less detail this may also be found in [DR]. Section 4 contains a detailed sketch of the algorithm and a brief discussion of the main data structures. Section 5 closes with some final remarks.

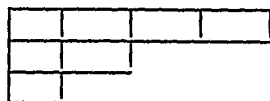
2. Background

A. Representations of the Symmetric Group

Let G be a finite group. Recall that a representation ρ of G is a map assigning matrices to group elements in such a way that $\rho(st) = \rho(s)\rho(t)$ for all s and t in G . Thus, ρ is a homomorphism from G to $GL(V)$ with V a vector space of dimension d_ρ , the *dimension* or *degree* of ρ . If H is a subgroup of G then the restriction of ρ to H defines a representation of H . This is denoted as $\rho \downarrow H$. The representation is *irreducible* if and only if for any subspace $W \subseteq V$, if $\rho(s)W \subseteq W$ for all $s \in G$ then either $W = \{0\}$ or $W = V$. Serre ([S]) is an accessible introduction to basic representation theory.

The representation theory of the symmetric group S_n has been studied extensively. James and Kerber ([JK]) provide a thorough treatment of this subject. It is a fundamental fact that the irreducible representations of S_n are in a natural one-one correspondence with integer partitions of n . Let λ be a partition of n . This is usually written as $\lambda \vdash n$. Explicitly one writes $\lambda = (\lambda_1, \dots, \lambda_k)$ where $\lambda_1 + \dots + \lambda_k = n$. The λ_i are the *parts* of λ . It is assumed that $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_k > 0$.

To each partition of n is associated its (*Ferrers*) *diagram*. Recall that this is a left-justified arrangement of boxes with λ_i boxes in the i^{th} row. The diagram for the partition λ is said to be of *shape* λ . For example, the diagram for $\lambda = (4, 2, 1)$ is



Much of the representation theory of S_n involves the combinatorial properties of Ferrers diagrams and their generalizations. In particular, a neat formulation of the manner in which an irreducible representation ρ of S_n splits when restricted to S_{n-1} may be given in terms of the associated diagrams.

Theorem (Branching theorem). Let λ be a partition of n and ρ_λ the corresponding irreducible representation. Then ρ_λ restricted to S_{n-1} splits into the direct sum of irreducible representations ρ_ν where ν runs over all partitions of $n-1$ whose diagrams can be obtained from the diagram of λ by removal of a single box.

Example: Let $\lambda = (4, 2, 1)$. Then ρ_λ restricted to S_8 splits into the direct sum

$$\rho_{(3,2,1)} \oplus \rho_{(4,1,1)} \oplus \rho_{(4,2)}.$$

The splitting means that for $\pi \in S_{n-1}$, $\rho(\pi)$ can be written, perhaps after a change of basis, in block diagonal form. It turns out that in two well-known bases, the orthogonal and seminormal forms discovered by Alfred Young, this block structure is automatic. Specifically,

Theorem (Proposition 4, [DR]) Let λ be a partition of n and ρ_λ be the corresponding irreducible matrix representation of S_n in Young's seminormal (resp. orthogonal) form. Let $\pi \in S_{n-1} \subseteq S_n$. Then

$$\rho_\lambda(\pi) = \begin{pmatrix} \rho_{\nu^{(1)}}(\pi) & 0 & \dots & 0 \\ 0 & \rho_{\nu^{(2)}}(\pi) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \rho_{\nu^{(l)}}(\pi) \end{pmatrix}$$

where the $\nu^{(i)}$ are partitions of $n-1$ determined by the branching theorem and the $\rho_{\nu^{(i)}}(\pi)$ are again given in Young's seminormal (resp. orthogonal) form as defined for S_{n-1} .

Applying this inductively, the analogous result holds true for $\pi \in S_k$ for any k , $1 \leq k \leq n$.

Kerber ([K]) gives a very readable account of the matrix constructions. There, formulas, written in terms of functions defined on *standard Young tableaux* give closed form expressions for the matrix representations for all the pairwise adjacent transpositions $(j, j+1)$. The problem of building up all needed representations is addressed in section 4.

B. Fourier Transforms

Let f be a complex-valued function on a group G . The *Fourier transform* of f at a representation ρ of G is defined as

$$\hat{f}(\rho) = \sum_{s \in G} f(s) \rho(s) \quad (2.1)$$

Let $H \subseteq G$ be any subgroup of index k with coset representatives $\{s_1, \dots, s_k\}$ for G/H . Then (2.1) may be rewritten as

$$\hat{f}(\rho) = \sum_{i=1}^k \rho(s_i) \sum_{h \in H} f_i(h) \rho(h) = \sum_{i=1}^k \rho(s_i) \hat{f}_i(\rho \downarrow H) \quad (2.2)$$

where $f_i(h) = f(s_i h)$.

Apply this to S_n . The n transpositions $\{(n, n), (n-1, n), \dots, (1, n)\}$ ((n, n) is the identity) are a set of coset representatives for S_n/S_{n-1} . Thus, for ρ a representation of S_n (2.2) specializes to

$$\hat{f}(\rho) = \sum_{i=1}^n \rho(i, n) \sum_{\pi \in S_{n-1}} f_i(\pi) \rho(\pi).$$

Iterating this procedure yields (for any $k, 1 < k < n$)

$$\hat{f}(\rho) = \sum_{i_n=1}^n \rho(i_n, n) \sum_{i_{n-1}=1}^{n-1} \rho(i_{n-1}, n-1) \dots \sum_{i_k=1}^k \hat{f}_{i_n \dots i_{k+1}}(\rho \downarrow S_k)$$

where $f_{i_n \dots i_{k+1}}(\pi) = f((i_n, n) \dots (i_{k+1}, k+1)\pi)$ for all $\pi \in S_k$.

3. The Main Idea

As described in [DR] the major source of savings in any implementation of an "efficient" computation of Fourier transforms is the ability at all possible times to retrieve from memory previously computed restricted transforms rather than recompute them. That is, let ρ be an irreducible representation of G and let H be a subgroup of G . In general, $\rho \downarrow H$ will no longer be irreducible. Suppose that

$$\rho \downarrow H = \eta_1 + \dots + \eta_r$$

where the η_i are irreducible, although not necessarily distinct, representations of H . Thus, in a suitable basis the matrix for $\rho(h)$ for any $h \in H$ will have the form

$$\begin{pmatrix} \eta_1(h) & 0 & \dots & 0 \\ 0 & \eta_2(h) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \eta_r(h) \end{pmatrix}.$$

Consequently, (2.2) may be rewritten explicitly as

$$\sum_{i=1}^k \rho(s_i) \begin{pmatrix} \hat{f}_i(\eta_1) & 0 & \dots & 0 \\ 0 & \hat{f}_i(\eta_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \hat{f}_i(\eta_r) \end{pmatrix}.$$

As the irreducible representations ρ vary, the blocks $\hat{f}_i(\eta_j)$ will occur repeatedly, both for fixed ρ and different ρ 's. They need only be calculated once and then stored, subsequent computations simply retrieving them from memory. Iterating this idea through the most "refined" tower (that is a tower of subgroups for which the sum of the indices is maximal) yields impressive speedups. The Fourier transforms at the base of the tower are computed directly.

Practical considerations however, may make it impossible to store all restricted transforms. For example, consider the problem of computing Fourier transforms on S_{10} . To store all restricted transforms through S_5 would require approximately 80 megabytes of memory (assuming four bytes per floating point number). Clearly some compromises must be made.

To compute Fourier transforms on S_n two modifications are made. First, only restrictions as far as S_5 are considered. Any Fourier transforms needed on S_5 are computed directly. Second, only the restrictions to S_{n-1} will be stored throughout. In the case of $n = 10$ this brings the storage requirements down to a more manageable 13 megabytes. It is worth noting that at the cost of an additional 13 megabytes all restricted transforms to S_5 could be precomputed and stored for quick retrieval at the last level of computation. Even without storing all the restrictions one may still take advantage "locally" of the existence of identical blocks in the restrictions. This is perhaps best illustrated by an example.

Let f be a function defined on S_{10} and consider the problem of computing the Fourier transform of f at the representation $\rho_{(7,3)}$. The following "branching tree" represents the way

in which $\rho_{(7,3)}$ splits when restricted (recall the branching theorem). The k^{th} level shows the splitting for $\rho_{(7,3)}$ restricted to S_{10-k} . The restrictions through S_8 are shown in Figure 1.

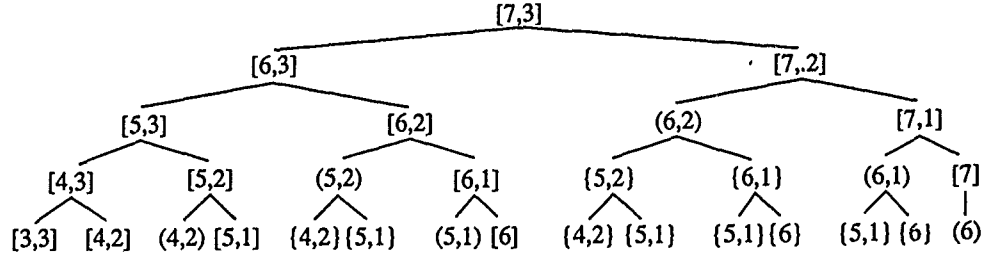


Figure 1: Branching Tree for (7,3)

In the example at hand assume that both $\hat{f}_i(\rho_{(6,3)})$ and $\hat{f}_i(\rho_{(7,2)})$ must be computed for all i , $1 \leq i \leq 10$. Following the algorithm consider the restriction to S_8 where $\rho_{(6,2)}$ occurs as a direct summand (ie. block) in both restrictions (matrices). This need only be computed once, and then copied immediately to the other block of the matrix. Consequently, no additional work need be done to compute the block corresponding to this second occurrence of $\rho_{(6,2)}$.

With this in mind, the notation in Figure 1 may now be explained. The partitions enclosed by square brackets are representations which must be computed (the first occurrences). Those within parentheses will have the restricted transforms copied into them, while those inside curly brackets will be ignored entirely. These are simply the restrictions which occur beneath representations which are going to be copied at a higher level.

As the calculation of the Fourier transforms proceeds, for each partition of n such a "branching tree" is generated. These provide "road maps" for the computation in the sense described above. The above scenario requires that computations be scheduled in such a way that all restrictions to a given subgroup be computed on the same pass. Thus, this tree which guides the computation requires a little more connectivity than is usually given. The branching trees are data structures of the following form:

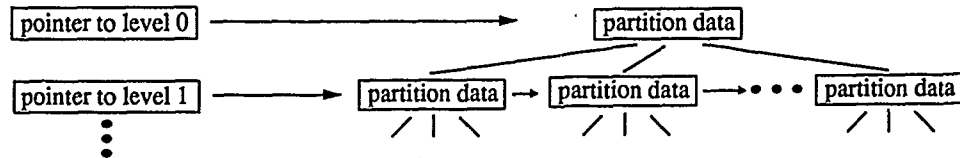


Figure 2: Branching Tree Layout

That is, the tree is created as shown previously, but with additional pointers between nodes on a given level, as well outside pointers to the first partition on each level. The structure of the nodes of the tree is shown in Figure 3. There, "Partition" is an integer array with zeroth element equal to the number of parts and the successive elements equal to the parts, in decreasing order. "Dimension" is the dimension of the associated representation and "compute" and "spread" are boolean flags indicating if this restriction is to be computed (the partitions enclosed in brackets in figure 1), or spread to (those enclosed in parentheses). If both are false then nothing is done at the node (the partitions enclosed in curly brackets). "ptr 1" through "ptr j" point to the restrictions of "partition".

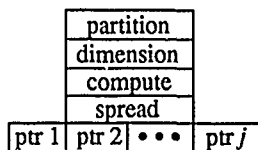


Figure 3: Branching Tree Node Structure

4. The Implementation

A. The Algorithm

Before giving a detailed sketch of the algorithm a few remarks explaining some of its structure and notation are necessary.

Remark 1. The Fourier transforms are computed in lexicographically decreasing order with respect to the associated partitions. Such scheduling has the property that at each computation, at most one new restriction must be computed. Formally, a simple exercise in manipulating diagrams gives the following proposition:

Proposition 1 *Let $\lambda = (\lambda_1, \dots, \lambda_r)$, be a partition of n and ρ_λ be the corresponding representation of S_n . Let R_λ be the set of all partitions of $n-1$ that correspond to "restrictions" of partitions that are lexicographically greater than λ . (ie. R_λ is the set of all partitions of $n-1$ that may be obtained by removing a single block from a Ferrers diagram of a partition strictly greater than λ .) Then,*

- i) *If $\lambda_1 = \lambda_2 > 0$ then all restrictions of λ are in R_λ .*
- ii) *If $\lambda_1 > \lambda_2$ then exactly one new partition of $n-1$ occurs in the restriction and it is the lexicographically less than all those in R_λ .*

Thus, while looping through the partitions in lexicographically decreasing order it is first checked if the first two parts of the partition are equal. If so, all the restricted transforms have been computed and they need only be retrieved in order to perform the calculation. Otherwise, compute (in the manner sketched above) the "first" of the restrictions and retrieve the others.

Remark 2. Restricted transforms $\{\hat{f}_i(\eta_\lambda)\}_{i=1}^{10}$ are stored in separate files indexed by the partitions of $n-1$. This seems to permit easy access and management. Upon computing a given Fourier transform it is initially determined which representations occur in the restriction to S_{n-1} . Pointers are then directed to the corresponding files of precomputed restricted transforms. These pointers are collected in a single array "branch-file-pointers[k]". All of this is performed by the subroutine "open-restrictions(p1, p2, b-array)". Here, "b-array" is an array of partitions of $n-1$ given by restricting the current representation of S_n . The parameters p1 and p2 indicate that partitions p1 through p2 in this array have had their corresponding restricted transforms computed so that these files should be opened. After opening these files, retrieval of the restrictions is done by simply moving through them sequentially. This is carried out by the subroutine "Retrieve-Transform(i, j, m)" which returns $\hat{f}_i(\text{b-array}[j])$ in the matrix m .

The algorithm which follows is in two parts, a "main" body and the subroutine "Compute-Restriction". In the main body some variables are matrices, while others are integers. In any particular case the context should make it clear which is meant. "transp-array" is an array

of integers such that $\text{transp-array}[j] = i_j$ for $k \leq j \leq n$ denotes that as the recursion reaches S_{k-1} the Fourier transforms of $f_{i_n \dots i_k}$ are being computed.

Subroutines other than those mentioned above are

(a) $\text{Branch}(\lambda, \text{number-of-branches}, \text{branches})$ – This takes a partition of k and returns in the variable $\text{number-of-branches}$ the number of partitions of $k-1$ to which it restricts, as well as these subpartitions, in increasing order, in branches , a variable array of partitions.

(b) $\text{Make-branch-tree}(\text{root}, \lambda)$ – returns in root a pointer to the branching tree for the partition λ as described in section 2.

(c) $\text{Build-matrix}(\text{matrix1}, \text{matrix2})$ – inserts matrix2 as a block on the diagonal of matrix1 in some specified position.

(d) $\text{Store-transform}(i, \lambda, \text{restriction})$ – creates a file containing the restricted Fourier transforms at the representation ρ_λ .

(e) $\text{Get-representation}(\lambda, i, \text{matrix})$ – for λ a partition of k returns in matrix the representation $\rho_\lambda(i k)$.

$\text{Compute-Restriction}$ is called from the main body with the following parameters:

k : Indicates computation of the restriction to S_k .

result: A matrix variable returning the block matrix $\widehat{f}_k(\rho \downarrow S_k)$. (Note that this matrix may only be partially filled.)

Additional subroutines called by $\text{Compute-restriction}$ are

(a) $\text{Spread-the-blocks}(\text{matrix})$ – Using the branching tree, this subroutine takes as input a block diagonal matrix and performs the appropriate copying of blocks as explained in section 2.

(b) $\text{Block-matrix-mult}(\text{matrix3}, \text{matrix1}, \text{matrix2})$ – matrix1 is block diagonal and matrix2 can be block diagonal of the same size, or full, of size equal to the size of some block in matrix1 . In either case, the appropriate multiplication is performed and put in matrix3 .

Note that the actual function f is not a parameter here. It is stored in a file in such a way that as the computation proceeds, the values may be read out sequentially. Figure 4 gives a “recursive picture” of the data storage in the sense that the functions f_i are stored in consecutive blocks and within the block for each f_i , the functions f_{ij} are stored similarly, etc.

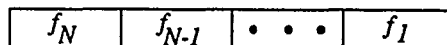


Figure 4: Data Storage

Algorithm for Computing Fourier Transforms on S_N

main body:

```

FOR  $\rho =$  first representation of  $S_N$  to the last DO
   $\text{fourier-transform} = 0$ ;
   $\text{branch}(\rho, \text{number-of-branches}, \text{branches})$ ;
  IF first part of  $\rho =$  second part THEN
     $\text{open-restrictions}(1, \text{number-of-branches}, \text{branches})$ ;
     $\text{first-repeat} = 1$ ;
  ELSE /* first part is different from second*/
     $\text{open-restrictions}(2, \text{number of branches}, \text{branches})$ ;
     $\text{first-repeat} = 2$ ;
   $\text{make-branch-tree}(\text{root}, \text{branches}[1])$ ;

```



```

FOR  $i = N$  DOWNTO 1 DO
     $transp[N] = i$ ;
    compute-transform( $n - 1$ ,  $branches[1]$ ,  $restriction$ );
    store-transform( $i$ ,  $branches[1]$ ,  $restriction$ );
    build-matrix(  $temp$ ,  $restriction$  );
    FOR  $j = first-repeat$  TO  $number-of-branches$  DO
        retrieve( $\hat{f}_i(branches[j])$ )
        build-matrix(  $temp$ ,  $\hat{f}_i(branches[j])$ )
    IF ( $i = N$ ) THEN  $\rho(i, N) = identity$ ;
    ELSE get-representation( $\rho, i, \rho(i, N)$ );
     $fourier-transform = fourier-transform + \rho(i, N) * temp$ ;
store-ft( $fourier-transform$ );

Compute-Restriction( $k$ ,  $result$ )
    IF  $k = 5$  THEN
        FOR  $\rho =$  the first partition at this level in the tree TO the last DO
            IF the restriction at this partition is to be computed (ie. if
                this is a first occurrence of the partition) THEN compute directly;
        spread-the-blocks( $result$ );
    ELSE
         $result = 0$ ;
        FOR  $i = k$  DOWNTO 1 DO
             $transp-array[k] = i$ ;
            Compute-Restriction(  $k - 1$ ,  $restriction$  );
            FOR  $\rho =$  first partition in the restriction to  $S_k$  TO the last DO
                get-representation( $\rho, i, \rho(i, k)$ );
                block-matrix-mult(  $temp$ ,  $\rho(i, k)$ ,  $restriction$  );
                 $result = result + temp$ ;
        spread-the-blocks(  $result$  );

```

B. Generating the Representations

In order to compute the Fourier transforms for S_n the algorithm requires that for each k with $5 \leq k \leq n$, the irreducible representations of the transpositions (j, k) , $1 \leq j \leq k$, considered as elements of S_k , (as opposed to considering them as elements of S_n) be available. In the case of Young's orthogonal and seminormal forms, the representations for pairwise adjacent transpositions (those of the form $(k, k+1)$) may be easily computed in terms of simple functions defined on standard Young tableaux (see [K]). Unfortunately, no such expression is known for the other transpositions. Thus, these must be built up from the pairwise adjacents by successive conjugations. For each k , the necessary representations of elements of S_k are stored in separate files whose exact form is explained in the following section. In the algorithm which follows "make-pairwise-adj" is a subroutine which takes as a parameter an integer k and consequently creates a file containing the irreducible representations of all the pairwise adjacent transpositions (these are simply stored in blocks in lexicographically decreasing order - with respect to partitions - and within each block in the order $\rho(j-1, j), \dots, \rho(1, 2)$), and returns in "n-of-reps" the number of representations (ie. partitions of k) generated. "p-a-t" is an array of matrices such that each call to "get-p-a-t" (one occurring for each representation ρ of S_k) fills p-a-t[j] with $\rho(jj+1)$ for all j between 1 and $k-1$. "write-to-file" writes each representation to a file in the format explained in section B. "conjugate(dest, m1, m2)" returns in the matrix variable $dest$, the product $m1 * m2 * m1$ for matrix variables $m1$ and $m2$. If $m1$

is a representation of a transposition then $(m1)^{-1} = m1$, so this is conjugation.

```

FOR n = 6 to N DO
  make-pairwise-adj(n, n-of-reps);
  FOR j = 1 TO n-of-reps DO
    get-p-a-t(p-a-t);
    matrix1 = p - a - t[n - 1];
    FOR k = n - 2 DOWNT0 1 DO
      write-to-file(matrix1);
      conjugate(matrix1, p-a-t[k], matrix1);
      write-to-file(matrix1);

```

Algorithm for Generating Pairwise Adjacent Representations

It is worth pointing out that only the irreducible representations for $(k-1, k)$ in S_k for $1 \leq k \leq n$ need be computed. Using the branching property of these representations, the other pairwise adjacents may be built up successively. This may be important when space is at a premium. On the SUN4 and VAX11/750 this never became an issue. Also, note that if λ is a partition of k and λ' the conjugate partition (the partition obtained by considering the transpose of the Ferrers diagram of λ) then the pairwise adjacent representations for $\rho_{\lambda'}$ may be obtained directly from those for ρ_{λ} .

C. Storing The Representations

As the program executes, various restricted transforms are computed. Each of these computations may require retrieving many distinct representations of the transpositions. To rewind the file of representations for each retrieval would clearly be extremely wasteful and inefficient. Thus, a coherent scheme for keeping track of position in the file at any given moment, as well as "markers" to indicate relative positions of the other representations is useful. By keeping a "bookmark" for each file and storing the representations between distinguishing "bookends" it is possible to move directly from one representation to the next.

The bookmarks data structures consist of four objects. There is the actual physical marker, a file pointer, pointing to the current position in a given file of representations. Also, the corresponding partition of the current representation is stored as an integer array. Finally, there are two additional integers, denoting the dimension of the representation and the current transposition.



Figure 5: Bookmark Structure

To compute Fourier transforms on S_n the representations for elements in $S_n, \dots, S_5$ (recall that restrictions are computed until S_5 is reached where the computation is performed directly) must be stored. The representations for each of the different symmetric groups are stored in distinct files. Consequently, an array of bookmarks is kept, one for each group.

The structure of the files at which they point is dictated by the need for relative movement within a file. Figure 6 shows the file structures for the representations of $S_6, \dots, S_n$. The object "reversed partition" is simply the original partition stored in reverse order. If ρ is the representation corresponding to p then the data block for each $\rho(i, k)$ is some d_p^2 block of

floating point entries. The only difference in the file for S_5 is that the representations of all the group elements are stored.

• • •	partition	dimension	($n-1$ n)	• • •	(1 n)	dimension	reversed partition	• • •

Figure 6: Representation File Structure

These blocks are stored in lexicographically decreasing order with respect to the partitions.

In brief, the bookmarks contain sufficient information to move the file pointer in either direction directly to the end of a block of representations. The bookends then contain all the necessary information for the new updating of the bookmark for the next block.

5. Concluding Remarks

Perhaps the single most important factor in the speed of the algorithm described here is the exponent for matrix multiplication. That is, the value a such that two m by m matrices can be multiplied in $O(m^a)$ steps. Currently, the lowest theoretical bound, due to Winograd ([W]), is $a = 2.38$. In practice $a = 3$ when the naive algorithm is used.

It is also worth pointing out that large numbers of matrix calculations for matrices with floating point entries can result in the accumulation of serious rounding errors. Young's semi-normal form gives a representation of S_n defined over the rational numbers (the orthogonal form is real, but not rational). Consequently, if the spectral analysis is to be carried out on a rational-valued function rounding errors may be avoided entirely by performing rational arithmetic. Indeed, even if f is not rational such methods may substantially lessen the cumulative error.

Depending on the storage capacity of the computing environment these ideas can be amplified by computing and storing the restrictions to smaller subgroups.

Acknowledgements

Thanks to David Fry for his useful suggestions and help with the diagrams.

References

- [D1] Diaconis, P. (1987). Spectral analysis for ranked data. To appear, Ann. Statist
- [D2] Diaconis, P. (1988). Group Representations in Probability and Statistics. Institute of Mathematical Statistics, Hayward, CA.
- [DR] Diaconis, P. and Rockmore, D. (1988). Efficient computation of Fourier transforms on finite groups, To appear.
- [JK] James, G. D. and Kerber, A. (1981). The Representation Theory of the Symmetric Groups. Addison-Wesley, Reading, Mass.
- [K] Kerber, A. (1971). Representations of Permutation Groups I. Lecture Notes in Math. 240. Springer-Verlag, Berlin.
- [S] Serre, J.P. (1977). Linear Representations of Finite Groups. Springer-Verlag, New York.
- [W] Winograd, S. (1978). On computing the discrete Fourier transform. Math. Comp. 32, 175-199.

Integration in Finite Terms and Simplification with Dilogarithms: A Progress Report ¹

Jamil Baddoura

Massachusetts Institute of Technology, Cambridge, MA 02139

Abstract. *In this extended abstract, we report on a new theorem that generalizes Liouville's theorem on integration in finite terms. The new theorem allows dilogarithms to occur in the integral in addition to elementary functions. The proof is based on two identities, for the dilogarithm, that characterize all the possible algebraic relations among dilogarithms of functions that are built up from the rational functions by taking transcendental exponentials, dilogarithms, and logarithms. We report also on a generalization of Risch's decision procedure for integrating elementary transcendental functions to include dilogarithms and elementary functions in the integral.*

1. Introduction

In 1967, M. Rosenlicht [9] published an algebraic proof of Liouville's theorem on the problem of integration in finite terms with elementary functions, based on the notions of differential algebra. R. Risch [8] was able to sharpen it and obtain an algorithm for calculating the integral of an elementary transcendental function. In 1972, J. Moses [6] started discussing the problem of extending Liouville's and Risch's result to include non-elementary functions in the integrand as well as in the integral. He asked whether a given expression has an integral within a class of expressions of the form $(F(V_i))$, where F is a given special function and (V_i) is a finite set of functions lying in the ground field. Singer, Saunders, and Caviness [11] proved an extension of Liouville's theorem allowing logarithmic integrals and error functions to occur in the integral. Their result allowed Cherry [2] to obtain two decision procedures for expressing integrals of transcendental elementary functions in terms of those special functions. The dilogarithm is, however, more complex than logarithmic integrals and error functions, in the sense that if an integrand has an integral which can be expressed using dilogarithms, these can have derivatives which contain logarithms transcendental over the field of the integrand. Moreover, the logarithms in the derivatives of dilogarithms may be algebraically independent even though the dilogarithms and these logarithms are algebraically dependent.

R. Coleman [3] produced an analytic characterization of the identities of the dilogarithm for rational functions. We show that two identities of the dilogarithm, in addition to the

¹This research has been supported, in part, by the Air Force Office of Scientific Research, Grant No. AFOSR-85-0264.

identities among primitives and the identities among exponentials, are required to generate all algebraic relations among dilogarithms and logarithms of functions built up from the rational functions by taking transcendental exponentials, logarithms, and dilogarithms. Our proof uses Ostrowski's theorem [7] in several places. Given these two identities, we generalize Liouville's theorem to include dilogarithms in the integral, in addition to elementary functions. As expected, dilogarithms have close relations with the logarithms in the way they appear in the integral. In addition, we generalize Risch's decision procedure for integrating transcendental elementary functions allowing dilogarithms to occur in the integral.

2. Dilogarithmic-Elementary Extensions

Let k be a differential field of characteristic zero. The derivation operator of k into itself will be denoted by $'$, that is, the derivative of an element x in k is x' . The subfield of constants of k will be denoted by C . A differential field extension F of k is said to be dilogarithmic-elementary over k if F can be resolved into a tower:

$$F = F_n \supseteq F_{n-1} \supseteq \cdots \supseteq F_1 \supseteq F_0 = k$$

such that $F_i = F_{i-1}(\theta'_i, \theta_i)$, where, for each $i, 1 \leq i \leq n$, one of the following holds,

- (i) $\theta'_i = \phi'/\phi$ for some nonzero ϕ in F_{i-1} , which we write $\theta_i = \log \phi$. We say that ϕ_i is logarithmic over F_{i-1} .
- (ii) $\theta'_i = \phi'\theta_i$ for some ϕ in F_{i-1} , which we write $\theta_i = \exp \phi$. We call θ_i exponential over F_{i-1} .
- (iii) $\theta'_i = -(\phi'/\phi)u$, where $\phi \in F_{i-1} - \{0, 1\}$, and u is such that $u' = (1 - \phi)'/(1 - \phi)$. In this case, we write $\theta_i = \ell_2(\phi)$ and call θ_i dilogarithmic over F_{i-1} . We note, in this case, that θ_i is defined up to the addition of a constant multiple of a logarithm over F_{i-1} since u is defined up to a constant. We don't assume, however, that u lies in F_{i-1} .
- (iv) θ_i is algebraic over F_{i-1} .

Roughly speaking, condition (iii) means that θ_i is the composition of the function ϕ with the dilogarithmic function $\ell_2(x)$ defined as:

$$\ell_2(x) = - \int_0^x \frac{\log(1-t)}{t} dt$$

If f is an element of k and F is some dilogarithmic-elementary extension of k , we say that f has an integral in F if there exists an element g in F whose derivative is f . We say then that f has a dilogarithmic-elementary integral.

A differential field extension F of k is said to be transcendental-dilogarithmic-elementary over k if F can be resolved into a tower:

$$F = F_n \supseteq F_{n-1} \supseteq \cdots \supseteq F_1 \supseteq F_0 = k$$

such that $F_i = F_{i-1}(\theta'_i, \theta_i)$, where, for each $i, 1 \leq i \leq n$, one of the following holds,

- (i) $\theta'_i = \phi'/\phi$ for some nonzero ϕ in F_{i-1} , which we write $\theta_i = \log \phi$. We say that ϕ_i is logarithmic over F_{i-1} .

- (ii) $\theta'_i = \phi' \theta_i$ for some $\phi \in F_{i-1}$, and ϕ_i transcendental over F_{i-1} . We write $\theta_i = \exp \phi$.
- (iii) $\theta'_i = -(\phi'/\phi)u$, where $\phi \in F_{i-1} - \{0, 1\}$, and u is such that $u' = (1 - \phi)'/(1 - \phi)$. In this case, we write $\theta_i = \ell_2(\phi)$.

Let f be an element of k . We say that f has a transcendental-dilogarithmic-element integral if there exists a transcendental-dilogarithmic-elementary extension F of k and an element g in F such that the derivative of g is equal to f .

Our objective is to determine the structure of the integral of a function f when it has a dilogarithmic-elementary integral.

3. An Extension of Liouville's Theorem

Let k be a differential of characteristic zero and K be a differential field extension of k such that $K = k(t, u, v)$. We say that $t = D(\phi)$ if ϕ is an element of $k - \{0, 1\}$ and:

$$t' = -\frac{1}{2} \frac{\phi'}{\phi} u + \frac{1}{2} \frac{(1 - \phi)'}{(1 - \phi)} v$$

where $u' = (1 - \phi)'/(1 - \phi)$ and $v' = \phi'/\phi$. From this definition, it follows that t is defined up to the addition of a linear combination of $\log \phi$ and $\log(1 - \phi)$ with constant coefficients. Informally, t is equal to:

$$\ell_2(\phi) + \frac{1}{2} \log \phi \log(1 - \phi)$$

This motivates considering the dilogarithm ℓ_2 and the associated function D as defined mod the vector space generated by constant multiples of logarithms over k . We denote from now this vector space by M_k for any differential field k . So, if $W \in M_k$, then there exist constants $c_1, \dots, c_n$ and $u_1, \dots, u_n$ such that $u_i, 1 \leq i \leq n$, is logarithmic over k for all i , and:

$$W = \sum_{i=1}^n c_i u_i$$

One of the key foundations for this work is the following proposition (see [1]), which produces a differential algebraic characterization of the functional identities of the dilogarithm.

Proposition 3.1:

- (a) If k is a differential field of characteristic zero and f an element in $k - \{0, 1\}$, then:

$$D\left(\frac{1}{f}\right) \equiv -D(f) \pmod{M_k}$$

- (b) Let k be as above and let θ be transcendental over k with $k(\theta)$ being a differential field. Let $f(\theta) \in k(\theta)$ and K be the splitting field of $f(\theta)$ and $1 - f(\theta)$. We define, if α is a zero or a pole of $f(\theta)$, $\text{ord}_\alpha f(\theta)$ to be the multiplicity of $(\theta - \alpha)$. Then, there exists $f_0 \in k$ such that:

$$D(f(\theta)) \equiv D(f_0) + \sum_{a,b} \text{ord}_b(1 - f(\theta)) \text{ord}_a(f(\theta)) D\left(\frac{\theta - b}{\theta - a}\right) \pmod{M_{K(\theta)}}$$

where a and b are the zeros and poles of $f(\theta)$ and $(1 - f(\theta))$, respectively.

(c) Any identity for the dilogarithm can be reduced to "instances" of (a) and (b).

Example: Let $k = C(z)$, where z is transcendental over C and $z' = 1$, and C is the field of complex numbers. Applying the previous proposition to $f(z) = z^2$, $f(z) = z$, and $f(z) = -z$, respectively, yields

$$D(z^2) \equiv 2D\left(\frac{z-1}{z}\right) + 2D\left(\frac{z+1}{z}\right) \pmod{M_{C(z)}}$$

$$D(z) \equiv D\left(\frac{z-1}{z}\right) \pmod{M_{C(z)}}$$

and:

$$D(-z) \equiv D\left(\frac{z+1}{z}\right) \pmod{M_{C(z)}}$$

So, $D(z^2) \equiv 2D(z) + 2D(-z) \pmod{M_{C(z)}}$, which implies that:

$$\begin{aligned} \ell(z^2) &\equiv \frac{1}{2} \log z^2 \log(1-z^2) \\ &\equiv 2(\ell_2(z) + \ell_2(-z) + \frac{1}{2} \log z \log(1-z) + \frac{1}{2} \log(-z) \log(1+z)) \pmod{M_{C(z)}} \end{aligned}$$

and we obtain $\ell(z^2) \equiv 2\ell_2(z) + 2\ell_2(-z) \pmod{M_{C(z)}}$. This is a well known identity of the dilogarithm.

Our main theorem is a generalization of Liouville's theorem allowing dilogarithms to appear in the integral in addition to elementary functions. We proved the following theorem (see [1]).

Theorem 3.1: Let k be a differential field of characteristic zero, which is a Liouville extension of the subfield of constants assumed algebraically closed. Let f be an element in k and suppose that f has a transcendental-dilogarithmic-elementary integral. Then:

$$\int f = g + \sum_{i=1}^m s_i w_i + \sum_{j=1}^n d_j v_j \quad (1)$$

where n and m are positive integers, $g \in k$, $s_i \in k$ for all i , $1 \leq i \leq m$, w_i is logarithmic for all i , $1 \leq i \leq m$, d_j is a constant for all j , $1 \leq j \leq n$, and $v_j = D(\phi_j)$, where $\phi_j \in k - \{0, 1\}$ for all j , $1 \leq j \leq n$. We observed that, although v_j does not in general belong to k , it can even be transcendental over k , as is illustrated in the following example.

Example: Let k be any differential field of characteristic zero. Assume that θ is primitive and transcendental over k . Let $p(\theta)$ and $q(\theta)$ be two irreducible polynomials over k such that $\deg p > \deg q \neq 0$.

We consider the differential field $K = k(\theta)(\phi_1, \phi_2)$, where ϕ_1 and ϕ_2 are such that:

$$\phi_1' = \frac{p'(\theta)}{p(\theta)} \text{ and } \phi_2' = \frac{q'(\theta)}{q(\theta)}$$

It is immediate that ϕ_1 and ϕ_2 are algebraically independent over $k(\theta)$. It is also clear that, if ϕ_3 is such that:

$$\phi_3' = \frac{(p(\theta) + q(\theta))'}{p(\theta) + q(\theta)}$$

then ϕ_3 is transcendental over K . Consider the function:

$$f = \frac{1}{2} \left(\frac{q'}{q} - \frac{(p+q)'}{(p+q)} \right) \phi_1 - \frac{1}{2} \left(\frac{(p+q)'}{(p+q)} - \frac{p'}{p} \right) \phi_2 + \frac{1}{2} (\phi_1 + \phi_2) \frac{(p+q)'}{p+q}$$

$f \in K$, and we can check that:

$$\left[D\left(\frac{-p}{q}\right) + \frac{1}{2}(\phi_1 + \phi_2)\phi_3 \right] \equiv \int f \pmod{M_K}$$

but $(D(-p/q))'$ is transcendental over K since ϕ_3 is.

4. The Decision Procedure

In addition to the previous theorem and the identities of the dilogarithm described in Proposition 3.1, the decision procedure for dilogarithms makes use of the notions of a factored transcendental elementary field and its rank as defined in [2]. We have the following theorem (see [1]).

Theorem 4.1: Let $C(z)$ be a differential field of characteristic zero, where z is transcendental over C , a solution to $z' = 1$, and C is an algebraically closed subfield of constants.

Let $k = C(z_1, \theta_1, \dots, \theta_n)$, $n \geq 0$, be a transcendental elementary extension of $C(z)$ that is factored. Given f in k , one can decide in a finite number of steps if f has a transcendental-dilogarithmic-elementary integral and if so determine g, w_i, s_i, d_j, ϕ_j , and v_j satisfying (3.1).

Example: Let $k = C(z, \log z, e^z)$. Let $\theta = e^z$, and:

$$f(z) = \frac{1}{2} \left(\frac{1 - 2e^{2z}}{z - e^{2z}} - \frac{1}{z} \right) \log\left(\frac{e^{2z}}{z}\right) \epsilon k$$

(since $\log(e^{2z}/z) \epsilon k$). Let us try to evaluate $\int f(z) dz$. In the course of the algorithm, we need to determine if:

$$g(z) = \frac{1}{2} \left[(1 - e^{2z}) - \frac{1}{z} \right]$$

has finite decomposition of the form:

$$g \sum_{p \in \Delta} C_p \frac{\left[\frac{\theta^p - \lambda_p}{\lambda_p} \right]'}{\frac{\theta^p - \lambda_p}{\lambda_p}}$$

where Δ is a finite set of positive integers, C_p is a constant, and $\lambda_p \in C(z)$. In our case:

$$g(z) = \frac{1}{2} \frac{\left(\frac{z - \theta^2}{z} \right)'}{\frac{z - \theta^2}{z}}$$

So:

$$\begin{aligned} & \int \frac{1}{2} \left(\frac{1 - 2e^{2z}}{z - e^{2z}} - \frac{1}{z} \right) \log\left(\frac{e^{2z}}{z}\right) dz \\ &= \frac{1}{2} \log\left(\frac{e^{2z}}{z}\right) \log\left(\frac{z - e^{2z}}{z}\right) + D\left(\frac{e^{2z}}{z}\right) \end{aligned}$$

5. Conclusion

We believe that the results presented here can be extended to other special functions, such as the trilogarithm, if we can characterize its functional identities as we have done with the dilogarithm. However, it appears that the identities of the trilogarithm are fundamentally different from those of the dilogarithm and much harder to find. We hope that the results obtained here will spur further research for higher polylogarithms.

Acknowledgements

We would like to thank Joel Moses, Harold Stark, and Michael Singer for helpful discussions.

Bibliography

- [1] J. Baddoura: Ph.D. Thesis, M.I.T., Cambridge, MA in progress.
- [2] G. Cherry: Ph.D. Thesis, University of Delaware, 1983.
- [3] R. Coleman: Dilogarithms, Regulators and p-adic L-functions, *Inventiones Mathematicae*, **69**, 1982, 171-208.
- [4] E. Kolchin: Algebraic Groups and Algebraic Independence, *Amer. J. of Math.*, **90**, 1968, 1151-1164.
- [5] J. Moses: Symbolic Integration, the Stormy Decade, *Comm. of the ACM*, **14**, 1971, 548-560.
- [6] J. Moses: Towards a General Theory of Special Functions, *Comm. of the ACM*, **15**, 1972, 550-554.
- [7] A. Ostrowski: Sur les Relations Algébriques Entre les Integrales Indéfinies, *Acta Mathematica*, **78**, 1946, 315-318.
- [8] R. Risch: The Problem of Integration in Finite Terms, *Trans. Amer. Math. Soc.*, **139**, 1969, 167-189.
- [9] M. Rosenlicht: On Liouville's Theorem of Elementary Functions, *Pacific J. Math.*, **65**, 1976, 485-492.
- [10] M. Rosenlicht and M. F. Singer: On Elementary, Generalized Elementary and Liouvillian Extension Fields, in *Contribution to Algebra*, Bass, Cassidy, and Kovacic (eds.), Academic Press, New York, 1977, 329-342.
- [11] M. Singer, B. Saunders, and B. F. Caviness: An Extension of Liouville's Theorem on Integration in Finite Terms, *SIAM J. on Computing*, 1985, 966-990.

Why Integration is Hard

*H. James Hoover
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada, T6G 2H1*

Abstract. This paper is a brief introduction to how the techniques of computational complexity can be applied to real analysis—integration in particular. We investigate how the difficulty of computing a function relates to the difficulty of computing its integral. Our comments are directed to an audience that is more familiar with traditional analysis and numerical methods than it is with complexity theory.

0 Introduction

Here are two “obvious facts” about symbolic and numerical integration: (1) all polynomials are easy to evaluate and to integrate; and (2) there are many real functions that are easy to compute but that are hard to integrate. Why is this so?

Our use of the word *integrate* incorporates problem solving of the following kinds: You are given a specific continuous real function f and asked to compute the *real number* $a = \int_0^1 f(x)dx$. Or, you are given a specific continuous real function f and asked to compute the *function* $g(y) = \int_0^y f(x)dx$. Or, finally, you are given an arbitrary continuous real function f and asked to compute the *operator* $I(f) = \int_0^1 f(x)dx$.

Intuitively, the more we know about the structure of a real function f the easier it is to integrate. For example, the polynomial $a_n x^n + \cdots + a_1 x + a_0$ is fully described by its coefficients, and so it can be both computed and integrated by simple term-by-term methods. In contrast, consider the function f that is zero everywhere except for a unit height and width peak centered at some point a_0 . Knowing the structure of f , it is certainly easy to compute both f and its indefinite integral—and with about the same amount of effort. But suppose that we feed f into a numerical integration algorithm that is ignorant of f , and can only take samples of it. Since the algorithm cannot know where the peak of f is located, integrating to an accuracy of 1 over an interval of length 2^n will require about 2^n samples—substantially more work than just evaluating f .

Since merely hiding structural information can make even a simple function hard to integrate, it seems fairer to ask the following question: *Suppose that we have access to all of the structural information used to evaluate function f . How does the difficulty of computing f relate to the difficulty of integrating f ?*

We will approach this problem from the perspective of a complexity theorist and ask what can be computed by machine, and how efficiently. To do this we must ask some very basic questions:

- What does it mean to compute a real number?
- What does it mean to compute a real function?
- What does it mean to compute an operator on a real function?
- What does it mean to present a real function to an operator?

• What do easy and hard mean for the above?

This somewhat unorthodox approach, at least by numerical and functional analysis standards, leads to a very simple characterization of the easy to compute real functions, and to the surprising conclusion that even when f is simply a polynomial the above three integration problems are still hard.

1 What is a real number?

For computer scientists, computation is almost always symbolic, and by its very nature must deal with objects that can be described by collections of symbols from some finite alphabet. But a real number is not in general a finite object. For example, unlike an arbitrary integer, we cannot simply write down an arbitrary real number as a string of decimal digits. So what approach can we take to represent and compute with real numbers?

We could write down an equation using a finite number of symbols, such as $x^2 - 2 = 0$, and say that this represents the real number x that is the positive solution to the equation.

Or, we could write down a program P such as

```
P:  $\langle e \rangle_0 \leftarrow 1$ 
    for  $i \leftarrow 1$  to  $\infty$  do
         $\langle e \rangle_i \leftarrow \langle e \rangle_{i-1} + 1/i!$ 
    end for
```

which “computes” e in the sense that each $\langle e \rangle_n$ is closer to e than the one before. (In fact, for $n \geq 8$, $|e - \langle e \rangle_n| \leq 2^{-n}$.)

Although both of these “represent” a real number in some sense, it is rather difficult to do an actual computation with them. For example, how do you compute $x + e$ given the representations above? We cannot just “add” the equation $x^2 - 2 = 0$ to the program P . Some additional structure is required of the representations if we hope to describe even basic arithmetic operations on reals. Without some structure, a procedure that added two reals would be required to accept and interpret almost any set of symbols that could describe a real number. Yet we must also ensure that our structural constraints do not overly limit the kinds of reals that we can compute.

One compromise solution is to distinguish between the manner in which the real is defined or computed, and the manner in which it is presented to the outside world, and then to standardize the interface between the two. The main constraint on the standard interface is that it must accept a finite string of symbols as input, and deliver a finite string of symbols as output. This is required so that we can manipulate the real in a computational context. Other than this, the interface can be arbitrary, keeping in mind that a poor choice of interface can cause massive technical problems. Behind this interface we allow any kind of computation that we can imagine, including uncomputable computations.

The typical kind of interface that we use takes a natural number n , and delivers an approximation $\langle x \rangle_n$ to the real x that is being represented. In other words, the interface delivers a sequence of n -th approximations to the real being represented. (This notion appears in Turing [Tu36, Tu37], in Grzegorzczuk [Gr57], and in Bishop [Bi67].)

Definition 1.1 *Let x be a real number. The notation $\langle x \rangle_n$, for $n \geq 0$, stands for any rational number such that $|x - \langle x \rangle_n| \leq 2^{-n}$. We say that $\langle x \rangle_n$ is an n -th approximation to x , and that a sequence $\{\langle x \rangle_n\}$ of such approximations represents x .*

So if we choose to define the real x as the positive solution to $x^2 - 2 = 0$, then a representation of x could be computed by a procedure that solves such equations to arbitrary accuracy using rational arithmetic, and the interface to x would be a function $\langle x \rangle_n$ that on input n delivers a rational value such that $|\sqrt{2} - \langle x \rangle_n| \leq 2^{-n}$.

So much for representing reals. What about computation? The manner in which we obtain each representation dictates whether we say that the real is easy to compute, hard to compute, or not even computable at all. What do we mean by "compute", and by "easy" and "hard"?

Computation is always done in the context of some given model of abstract machine. Each computation is described as a program, for the abstract machine, that takes inputs and produces outputs. The model of computation also specifies what kinds of resources are consumed by a computation so that we can assign a cost to each particular computation on each specific input. These costs are typically *time*, in terms of number of basic steps, or *space*, in terms of number of atomic memory cells used by the machine during the course of the computation. The usual model of computation is the Turing machine, and the cost measure that interests us is time.

We specify the *complexity* of a computation by describing how the cost of the computation varies as a function of the size of the input being fed into the computation. In general, the cost will increase with increasing input length, and we are interested in the asymptotic rate at which the cost function grows. A computation is considered *easy*, or *feasible*, if its cost C as a function of input size n is bounded by some polynomial in n . That is, if there is a k such that for $n \geq 1$, $C(n) = O(n^k)$. This is commonly denoted by $C(n) = n^{O(1)}$. Thus the natural notion of feasibility for real numbers is that computing n bits of an approximation to real number x should only require time $n^{O(1)}$.

Definition 1.2 *Real number x is a feasible real if there is a Turing machine that, on input of natural number n , outputs $\langle x \rangle_n$ in time $n^{O(1)}$.*

Note that there are non-feasible reals whose n -bit approximations take $O(2^n)$ time to compute, and there are even uncomputable reals. However most kinds of reals that we use daily, such as π , e , and the roots of polynomials with feasible real coefficients, are feasible.

2 What is a real function?

If real numbers are considered to be convergent sequences of rationals, then a real function can be considered to be something that takes a sequence of rationals as input, and produces a sequence of rationals as output.

Consider any continuous real function f defined on $(-\infty, +\infty)$. One possible way of computing f on input x would be by a procedure that takes two inputs: a natural number n , and a representation of x . The procedure would read in n , and establish the interface to the representation of x . It would then make subroutine calls to obtain approximations $\langle x \rangle_i$ to the input, do some computation, possibly make more subroutine calls, and then finally output $\langle f(x) \rangle_n$.

Note how we let the function look at its argument via only the representation interface, and require the function to produce its output in the form of a real representation, thus separating the task of computing the function itself from the problem of computing the particular input. This separation is necessary if we want to talk about the intrinsic difficulty of computing the function f . We could not do this if the function were allowed to cross the input interface and examine how the input was computed, for then the complexity

of the function would be sensitive to the manner in which its input was obtained, not just to the value of the input. (There are theories of analysis that do allow this kind of activity and they can lead to very strange results—for example, continuous functions that are not bounded on closed intervals. [Ab80].)

What sort of computation model should we allow for functions? There are two main possibilities. One is to use an *unstructured* model, such as the Turing machine, that is permitted to manipulate the actual symbols of its input in any way whatsoever. In such a model, the computation has access to the actual bits of the rational numbers representing the input, and can manipulate them as it sees fit to produce the bits of the function's output. (This is the method chosen by Grzegorzczuk [Gr57] and by Ko and Friedman [KF82].)

The other model is more *structured*, and requires the inputs to be handled as atomic units, and permits only restricted operations on the input such as addition, subtraction, multiplication, and division. This is the model implicit in the study of approximation theory.

Historically the unstructured computational model came first, and if we were only interested in computing functions, then either alternative would be acceptable, although it is not clear that the two models have equivalent power. But we are ultimately also interested in computing operators such as integration, and in describing the computation of an operator we face the same problem that we did for functions with respect to their inputs—the need for standardization in order to avoid arbitrarily general descriptions of the functions being operated on. What form can this standardization take?

Recall that we want our operators to know as much about the structure of the function they are operating on as is used in computing the function itself. That is, we want to take a standard description of a function, feed it to an operator, and then let the operator examine the function. In this case, the unstructured model seems to be too complex to deal with. For example, suppose that one wishes to compute $\int_0^y f(x)dx$ for arbitrary functions f , where f is presented as a Turing machine program that potentially does some kind of obscure bit manipulations on its input in order to obtain an output value. We can imagine writing the integration operator so that it examines the Turing machine program directly. It deduces information about the function being computed, which it then uses to compute the integral in a cleverer way than by just taking samples of f and integrating numerically. But this is not too likely, and for arbitrary f is actually impossible. However, if f is presented as a polynomial represented in some standard form, then the structure in the description of f can be exploited to efficiently symbolically integrate f .

One structured approach to computing a real function is to use *arithmetic circuits*. These circuits do nothing but arithmetic operations, and are the natural extension to the traditional notions of approximation by polynomials and rational functions [BB86]. Each arithmetic circuit over R is an acyclic network of gates where the edges carry real numbers and the gates perform the operations $+$, $-$, $\times$, $\cdot^{-1}$ (inverse) or deliver rational constants. A computation by such a circuit is the obvious one, with the circuit computing a rational function over R , and with the proviso that the computation is undefined when any inverse gate has a zero input. These circuits and their extensions to general fields have been extensively investigated ([vzG86], [vzGS86]) and are one of the main models of parallel algebraic complexity.

Figure 2.1 is an example of an arithmetic circuit α that computes the polynomial $\alpha(x) = x - 1$ in a rather stupid way. Note how the output of the circuit is undefined when the input is -1 .

How do we use arithmetic circuits to compute arbitrary continuous real functions? Just as we use sequences of rationals to approximate a real number, we can use a sequence, or

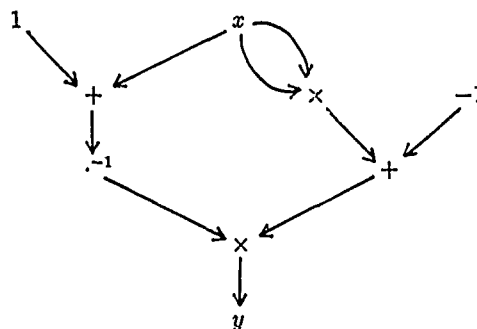


Figure 2.1: Arithmetic circuit $\alpha(x) = x - 1$.

family of arithmetic circuits to approximate a real function. For example, each member α_n of the family $\{\alpha_n\}$ could approximate f within 2^{-n} . But for many f no single rational function can approximate f within 2^{-n} over the entire interval $(-\infty, +\infty)$. For example, the function $\sin(x)$ has an infinite number of zeroes, so any rational function $P(x)/Q(x)$ that is within 2^{-n} , $n \geq 2$, of $\sin(x)$ must also have an infinite number of zeroes, which implies that $P(x)$ is either constant or has infinite degree. So any approximating family of rational functions will in general require an index that specifies the range over which the approximation works.

For notational simplicity, we use one index to indicate both the accuracy of approximation and the range over which it works. Each circuit α_n of the approximating family $\{\alpha_n\}$ for f takes as input a real $x \in [-2^n, 2^n]$, and computes a real output, denoted by $\alpha_n(x)$, which approximates $f(x)$. Pictorially we have the following situation, where we slightly abuse the notation $\langle f(x) \rangle_n$ and allow it to denote real values, not just rational ones.

$$x \rightarrow \boxed{\alpha_n} \rightarrow \langle f(x) \rangle_n$$

Note that if x is a rational number, then the arithmetic nature of the gates in α_n ensure that $\alpha_n(x)$ will also be a rational number.

We can now define what it means to compute a real function with arithmetic circuits.

Definition 2.1 Let $\{\alpha_n\}$ be a family of arithmetic circuits over R , and let f be a real function. Suppose that for all $n \geq 0$, circuit α_n satisfies the relation that if $x \in [-2^n, 2^n]$ then $|f(x) - \alpha_n(x)| \leq 2^{-n}$. Then we say that the family $\{\alpha_n\}$ of arithmetic circuits sup-approximates real function f .

This partly addresses the structure issue for functions. What remains is to develop an appropriate notion of cost for the computations performed by arithmetic circuits.

3 Feasible real functions

Arithmetic circuits are in one sense a model of an idealized analog computer, and we could confine our study to those computations which are feasible on analog computers—whatever the notion of feasible means for such machines. But in practice, we must perform our computations on digital computers, and so we want any notion of a feasible arithmetic circuit computation to correspond to our usual notion of a feasible computation on a Turing machine. Knowing this, we can work primarily in the domain of feasible arithmetic circuits, being confident that our results remain feasible in the world of Turing machines. To establish this correspondence we must do two things.

First, we must be able to actually produce the description of each arithmetic circuit of the family in a reasonable time. In order to ensure this we require the circuits to satisfy a *uniformity condition*. There are many possible precise uniformity conditions that one can use, but for our purposes the following informal definition will suffice:

Definition 3.1 *An arithmetic circuit family $\{\alpha_n\}$ is log-space uniform if a description of the connection pattern, gate types, and values of the constant gates (encoded in binary) for circuit α_n can be produced in space $O(\log n)$ on a deterministic Turing machine.*

Note that we need uniformity only if we care about constructing the circuits, otherwise we can view them like reals, with α_n being a function that magically delivers the description of the n -th member of the family.

Secondly, once we have circuit α_n we actually want to use it to compute an approximation to f at some specific point x . Since we lack computing devices that actually manipulate real numbers we must view α_n as specifying a series of operations that can only be approximated using rational numbers. But since α_n is itself just an approximation to f , a sufficiently accurate simulation of α_n on input x will yield a good approximation to $f(x)$.

These rational computations must be feasible in the usual sense—a simulated computation of α_n on x must require at most time $n^{O(1)}$. This can be achieved with two constraints on the circuit family. One is that α_n must perform only $n^{O(1)}$ operations, that is, to have a polynomial number of gates, which we denote by $\text{size}(\alpha_n) = n^{O(1)}$.

The other condition is that the rational numbers involved in the simulation of α_n do not require more than a polynomial number of bits to represent. Otherwise, a simulation of α_n cannot possibly remain feasible. This amounts to bounding the magnitude of the numbers involved in the simulation by $2^{n^{O(1)}}$. For example, if no intermediate value ever has a magnitude bigger than 2^n , then no non-zero intermediate value will ever have a magnitude smaller than 2^{-n} , and no more than about $2n$ bits will ever be required to represent an intermediate value.

One way of keeping the numbers short is to restrict α_n to have degree $n^{O(1)}$. This implicitly limits the magnitude of the internal values to $2^{n^{O(1)}}$, and is the typical limit used in algebraic complexity theory. But there are many functions that are easy to compute, yet have high degree, so such a limit would overly restrict the kind of functions we could compute. Instead, we directly restrict the magnitude of the values involved in the circuit.

Definition 3.2 *Let $\{\alpha_n\}$ be an arithmetic circuit family over R , and let $\alpha_n^v(x)$ denote the output value of gate v of α_n on input x . The magnitude of circuit α_n , denoted $\text{mag}(\alpha_n)$ is the quantity*

$$\text{mag}(\alpha_n) \equiv \max_{v \in \alpha_n} \left\{ \max_{x \in [-2^n, 2^n]} |\alpha_n^v(x)| \right\}$$

That is, $\text{mag}(\alpha_n)$ is the absolute value of the largest output from any gate of α_n on any input $x \in [-2^n, 2^n]$.

Combining the consideration of polynomial size with feasible magnitude we get a class of circuits such that each member can be simulated by a polynomial time Turing machine.

Definition 3.3 A family $\{\alpha_n\}$ of arithmetic circuits over R is feasible-size-magnitude if $\text{size}(\alpha_n) = n^{O(1)}$ and $\text{mag}(\alpha_n) = 2^{n^{O(1)}}$.

Then by adding uniformity, we can define a sufficient condition for a circuit family to be feasible, and thus can define the notion of feasible real function in a way that captures our intuitions about feasibility.

Definition 3.4 A real function is feasible if it can be sup-approximated by a feasible-size-magnitude family of uniform arithmetic circuits.

The fact that a feasible-size-magnitude circuit family can be efficiently approximated by a Turing machine is expressed in the following proposition [Ho87].

Proposition 3.5 If a real function f is feasible, then there is a function μ from naturals to naturals, with $\mu(n) = n^{O(1)}$, and a Turing machine M , such that for all natural n , and all reals $x \in [-2^n, 2^n]$, if n and rational $\langle x \rangle_{\mu(n)}$ are input to M , then M outputs rational $\langle f(x) \rangle_n$ in time $n^{O(1)}$.

So to approximate the output of a feasible-size-magnitude arithmetic circuit $\alpha_n(x)$ to n bits of precision requires only $n^{O(1)}$ bits of precision in the input x .

This proposition has the following surprising, and non-trivial converse [Ho87, Ho88].

Proposition 3.6 Let f be a continuous real function. Suppose that there is a Turing machine M and a function μ from naturals to naturals, with $\mu(n) = n^{O(1)}$, such that for all natural n , and all reals $x \in [-2^n, 2^n]$, if n and rational $\langle x \rangle_{\mu(n)}$ are input to M , then M outputs rational $\langle f(x) \rangle_n$ in time $n^{O(1)}$. Then f is a feasible real function.

Thus there is an equivalence between the unstructured computation of f and its structured computation—if a Turing machine can compute the function f feasibly then we can find a feasible-size-magnitude arithmetic circuit family that computes the function f .

This equivalence can be exploited to obtain a feasible version of the Weierstrass approximation theorem that every continuous function can, over a closed interval, be approximated arbitrarily closely by polynomials.

Theorem 3.7 A real function f is feasible iff f can be sup-approximated by a uniform family of feasible-size-magnitude arithmetic circuits that do not contain any inverse gates.

In other words, every real function that is feasible under the conventional Turing machine complexity notions is in fact computable by a family of feasible polynomials described by arithmetic circuits that have no inverse gates but only contain $+$, $-$, $\times$, and constant gates. Thus we have a very simple structural characterization of the easy to compute real functions, and we can now ask if these functions can be integrated easily. (It is worth remarking that these results have analogues in the domain of space-bounded computation.)

4 Integration is Hard

Returning at last to our original motivation, we can now ask: *Suppose that f is a feasible real function. Is the indefinite integral $g(y) = \int_0^y f(x)dx$ also a feasible real function?*

Note that we do not ask if there is in general an efficient way to compute the integration operator, but merely if an easy to compute function has an easy to compute integral.

Although there is no definitive answer to this question, results by Ko [KF82] and Friedman [Fr84] indicate that it is probably as hard as other classical hard problems in complexity theory. How does one go about classifying such a problem as easy or hard?

In the usual practice of complexity theory, problems are classified by placing them into complexity classes according to their mutual difficulty on various models of computation. The most familiar classes are P , the class of problems with polynomial time solutions on Turing machines; NP , the class of search problems that have polynomial time solutions on nondeterministic Turing machines; and $\#P$ (pronounced "sharp p ") the class of enumeration problems whose number of solutions can be counted in polynomial time by nondeterministic counting Turing machines.

A problem X is considered to characterize a class if it is *complete*, that is, if the solution of any problem in the class can be reduced to the solution of problem X . The problems in P are considered to be easy, and the complete problems in NP and $\#P$ are considered to be hard. The interested reader can refer to [GJ79] for an excellent introduction to this material, but the basic issue is that we do not know how to solve NP or $\#P$ problems without resorting to an exhaustive search of an exponential size space.

In order to use the tools of complexity theory to classify the integration problem we must somehow move from the domain of combinatorial problems to the domain of real analysis. The standard difficult combinatorial problem is called SAT, and almost any attempt at showing that a problem is hard begins with it. Given a Boolean formula F of length n , SAT asks the question of whether there is an assignment of true or false to each variable of F that makes F true. In general, for a formula F of length n and containing at most n variables, we know no better way of finding a satisfying assignment for F than testing all 2^n possible truth assignments. Our suspicion is that it is not possible to do better.

Since SAT is NP -complete, any problem in NP can be converted into an equivalent problem that involves finding satisfying assignments of Boolean formulas. This conversion is such that if the satisfying assignments could be found quickly, then the original problem could be solved quickly. So efforts to find efficient algorithms for NP problems need only concentrate on finding efficient algorithms for SAT. The flip side of this relationship is that whenever we find a problem X such that being able to solve X helps us solve SAT, we suspect that there will be no efficient solution for X .

To show that integration is hard, we need only show how it can be used to solve some aspect of SAT, or of the corresponding $\#P$ -complete problem, $\#SAT$, which asks how many satisfying assignments are there for formula F ? The key idea is that integration can be used to count. All we require is a feasible real function S that maps Boolean formulas onto intervals of the real line in such a way that the integral of S , over the interval associated with formula F , corresponds to the number of its satisfying assignments. Being able to integrate S feasibly implies being able to feasibly count satisfying assignments, which implies $P = \#P$. We call such a function as S a *satisfiability function*.

How do we convert a discontinuous combinatorial problem into one involving continuous functions? We briefly describe the construction of S from [Ho87].

Let x be a positive integer. The function S considers each interval $[x, x+1]$ to encode some fixed boolean formula F . This interval is further divided by S into a number

of subintervals of fixed width δ , one for each possible assignment A of true or false to each of the variables in F . The value of S is defined on each subinterval as follows: If the assignment A satisfies F , then S is a triangular peak of height δ centered in the subinterval, and with height 0 at the end points of the subinterval. If the assignment A does not satisfy F , then S is the constant function 0 over the subinterval. Thus, S is a piecewise-linear continuous function with a peak at each satisfying assignment, and zero elsewhere. (Actually a version of SAT can be constructed that is infinitely differentiable and is still feasible, so smoothness is not an issue.)

The S function is best implemented using the unstructured Turing machine model, in which it is easy to see that a polynomial time bounded Turing machine can do the necessary decoding and testing for satisfiability. Then the equivalence theorem of the previous section can be applied to get an inverse-free feasible-size-magnitude circuit family $\{\alpha_n\}$ that computes S .

Now, for integer x corresponding to formula F , integrating S over the interval $[x, x+1]$ will compute the area of all the peaks at satisfying assignments, and so dividing by the area of one peak counts the number of satisfying assignments for F . Thus, if the indefinite integral of a feasible real function were itself a feasible real function, then it would require only $n^{O(1)}$ time to count the number of satisfying assignments of an arbitrary, length n , boolean formula, and thus #SAT would have an efficient solution. As this would imply $P = NP = \#P$, this is considered most unlikely.

The previous integration need not be done exactly, and for $|x| \leq 2^n$ it is sufficient to approximately integrate the polynomial α_n rather than the function S . So it's even unlikely that there is a family of feasible-size-magnitude circuits whose members approximate the integrals of the members of $\{\alpha_n\}$.

The following result [Ho87] summarizes this discussion. Note that it says that *there is a specific family of polynomials that are easy to compute but whose individual integrals are hard to even approximate*—never mind the general problem of computing the integration operator.

Theorem 4.1 *There is a family $\{\alpha_n\}$ of inverse-free feasible-size-magnitude arithmetic circuits such that the following are equivalent:*

1. *There is a family $\{\beta_n\}$ of feasible-size-magnitude arithmetic circuits such that, for $x \in [-2^n, 2^n]$*

$$| \beta_n(x) - \int_0^x \alpha_n(y) dy | \leq 2^{-n}$$

2. $P = \#P$

Furthermore, observing that the maximum operator can be used to detect peaks, and thus detect satisfying assignments leads to a similar result for the class NP .

Theorem 4.2 *There is a family $\{\alpha_n\}$ of inverse-free feasible-size-magnitude arithmetic circuits such that the following are equivalent:*

1. *There is a family $\{\beta_n\}$ of feasible-size-magnitude arithmetic circuits such that, for $x \in [-2^n, 2^n]$*

$$| \beta_n(x) - \max_{y \in [0, x]} \alpha_n(y) | \leq 2^{-n}$$

2. $P = NP$.

So even for easy to compute polynomials, the problems of integration and finding the maximum are hard. Perhaps the question that we should really be asking is why is it so difficult to compute the norms of easy to compute real functions, and what further structure is required on the computation of such functions so that operators become easy.

Acknowledgement

This research was supported by the Natural Sciences and Engineering Research Council of Canada grant OGP 38937.

References

- [Ab80] O. Aberth. *Computable Analysis*. 1980. McGraw-Hill.
- [Bi67] E. Bishop. *Foundations of Constructive Analysis*. 1967. McGraw-Hill.
- [BB86] J. M. Borwein and P. B. Borwein. *On the complexity of familiar functions and numbers*. Draft. Private Communication, 1986.
- [Fr84] H. Friedman. *The computational complexity of maximization and integration*. Adv. in Math., 53, 1984, 80-98.
- [GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.
- [vzG86] J. von zur Gathen. *Feasible arithmetic computations: Valiant's hypothesis*. J. Symb. Comp., 4, 1987, 137-172.
- [vzGS86] J. von zur Gathen and G. Seroussi. *Boolean circuits versus arithmetic circuits*. Proc. 6th Int. Conf. in Computer Science, Santiago, Chile. 1986, 171-184.
- [Gr57] A. Grzegorzcyk. *On the definition of computable real continuous functions*. Fund. Math., 44, 1957, 61-71.
- [Ho87] H. J. Hoover. *Feasibly constructive analysis*. Ph.D. thesis and Technical Report 206/87, Department of Computer Science, University of Toronto, 1987.
- [Ho88] H. J. Hoover. *Feasible real functions and arithmetic circuits*. Technical Report TR 88-16, August 1988, Department of Computing Science, University of Alberta, Edmonton, Canada.
- [KF82] K. Ko and H. Friedman. *Computational complexity of real functions*. Theoret. Comput. Sci., 20, 1982, 323-352.
- [Tu36] A. M. Turing. *On computable numbers, with an application to the entscheidungsproblem*. Proc. London Math. Soc. (2), 42, 1936/37, 230-265.
- [Tu37] A. M. Turing. *A correction*. Proc. London Math. Soc. (2), 43, 1937, 544-546.

LIOUVILLIAN SOLUTIONS OF LINEAR DIFFERENTIAL EQUATIONS WITH LIOUVILLIAN COEFFICIENTS

Extended Abstract

Michael F. Singer¹
Department of Mathematics, Box 8205
North Carolina State University
Raleigh, N.C. 27695

ABSTRACT. Let $L(y) = b$ be a linear differential equation with coefficients in a differential field K . We discuss the problem of deciding if such an equation has a non-zero solution in K and give a decision procedure in case K is an elementary extension of the field of rational functions or is an algebraic extension of a transcendental liouvillian extension of the field of rational functions. We show how one can use this result to give a procedure to find a basis for the space of liouvillian solutions of $L(y) = 0$ where $L(y)$ has coefficients in such a field.

I will consider the following two questions: Let K be a differential field and let $a_{n-1}, \dots, a_0, b \in K$. Let $L(y) = y^{(n)} + a_{n-1}y^{(n-1)} + \dots + a_0y$.

Question 1. When does $L(y) = b$ have non-zero solutions in K and how can one find all such solutions?

Question 2. When does $L(y) = 0$ have a non-zero solution y such that $y'/y \in K$ and how does one find all such solutions?

In [SING88], I present an algorithm to answer these questions when K is an elementary extension of $\mathbb{C}(x)$ or K is an algebraic extension of a purely transcendental liouvillian extension of $\mathbb{C}(x)$, where $\mathbb{C}$ is a computable algebraically closed field of characteristic zero. Before I discuss this algorithm, I will discuss why these are important questions and how they are related to each other. First, let me recall some definitions. A field K is said to be a differential field with derivation $D: K \rightarrow K$ if D satisfies $D(a+b) = D(a) + D(b)$ and $D(ab) = (Da)b + a(Db)$ for all $a, b \in K$. The set $C(K) = \{c \mid Dc = 0\}$ is a subfield called the field of constants of K . We will usually denote the derivation by $'$, i.e. $a' = Da$. A good example to keep in mind is the field of rational functions $\mathbb{C}(x)$ with derivation d/dx ($\mathbb{C}$ denotes the complex numbers). All fields in this paper, without further mention, are of characteristic zero. We say K is a liouvillian extension of k if there is a tower of fields $k = K_0 \subset K_1 \subset \dots \subset K_n = K$ such that for each $i = 1, \dots, n$, $K_i = K_{i-1}(t_i)$ where either (a) $t_i' \in K_{i-1}$ or (b) $t_i'/t_i \in K_{i-1}$ or (c) t_i is algebraic over K_{i-1} . For example

$\mathbb{C}(x, e^{x^2}, e^{\int e^{x^2}})$ is a liouvillian extension of $\mathbb{C}(x)$. We say K is an elementary extension of k if there is a tower of fields $k = K_0 \subset K_1 \dots \subset K_n = K$ such that for each $i = 1, \dots, n$, $K_i = K_{i-1}(t_i)$ where either (a) for some $u_i \neq 0$ in K_{i-1} , $t_i' = u_i'/u_i$ or (b) for some u_i in K_{i-1} , $t_i'/t_i = u_i'$ or (c) t_i is algebraic over K_{i-1} . For example $\mathbb{C}(x, \log x, e^{(\log x)^2})$ is an elementary extension of $\mathbb{C}(x)$. The example following the definition of liouvillian extension is not an elementary extension of $\mathbb{C}(x)$ since $\int e^{x^2}$ lies in no elementary extension of $\mathbb{C}(x)$. We say that w is liouvillian (elementary) over k if w belongs to a liouvillian (elementary) extension of k .

Algorithms to answer questions 1 and 2 would be useful in solving two other problems. First of all, an answer to question 1 would have a bearing on the Risch Algorithm. In a series of papers [RISCH68], [RISCH69], [RISCH70], Risch gave a procedure to answer the following question: Given α in an elementary extension K of $\mathbb{C}(x)$ ($\mathbb{C}$ a finitely generated extension of the rational numbers $\mathbb{Q}$ and $\mathbb{C}(K) = \mathbb{C}$), decide if $\int \alpha$ lies in an elementary extension of K . Liouville's Theorem states that if α has an antiderivative in an elementary extension of K , then $\alpha = v_0' + \sum c_i v_i'/v_i$ where $v_0 \in K$, $v_1, \dots, v_n \in \bar{C}K$ and $c_i \in \bar{C}$, where $\bar{C}$ is the algebraic closure of $\mathbb{C}$. Risch's algorithm gives a procedure to decide if such elements exist. As a corollary of Liouville's Theorem, one can show that if α is of the form fe^g with f and g in K , then α has an elementary anti-derivative if and only if $y' + g'y = f$ has a solution y in K (i.e. if and only if there is a y in K such that $(ye^g)' = fe^g$). In general, Risch's Algorithm forces one to deal, again and again, with this same question: given f and g in an elementary extension K of $\mathbb{C}(x)$, decide if $y' + g'y = f$ has a solution in K . When K is a purely transcendental extension of $\mathbb{C}(x)$, one may write $K = E(t)$ with $t' \in E$ or $t'/t \in E$ and t transcendental over E . Letting

$$y = \sum_{i=1}^m \sum_{j=1}^{n_i} \frac{a_{ij}(t)}{(p_i(t))^j} + h(t)$$

be the partial fraction decomposition of y , one can plug this expression into $y' + g'y = f$. Equating powers and using the uniqueness of partial fraction decompositions, one can find a finite number of candidates for the p_i 's and bound the degree of h . This allows one to find all possible solutions y . (In fact there are now improvements on this idea. Rothstein [ROTH76] showed how one can use "Hermite Reduction" to postpone, as much as possible, the need to factor polynomials). When K is not a purely transcendental extension of $\mathbb{C}(x)$, but involves algebraics in the tower, things are more complicated. In the purely transcendental case, partial fractions gave us a global normal form that captured all necessary local information (e.g. the factors of the denominators and the powers to which they appear). When algebraics occur, one does not have this normal form. If $K = E(t, \gamma)$ with γ algebraic of degree n over $E(t)$, one may write $y = b_0 + b_1\gamma + \dots + b_{n-1}\gamma^{n-1}$ with the $b_i \in E(t)$. To find the b_i , one is forced to work with puiseux expansions (a local normal form) at each place of the function field $E(t, \gamma)$. Although Risch

showed that this approach does yield an algorithm, it is much more complex than the purely transcendental case (Bronstein [BRON87] has made significant improvements in the Risch algorithm and can avoid puioux expansions in many situations, but he is still forced to consider them in certain cases). One would like to reduce the question of deciding if $y' + g'y = f$ has a solution in $E(t, \gamma)$ to a similar question in $E(t)$, where one could apply partial fraction techniques and a suitable induction hypothesis. If we write $y = b_0 + b_1\gamma + \dots + b_{n-1}\gamma^{n-1}$, substitute into $y' + g'y = f$, and equate powers of γ , we get a system of first order linear differential equations for the b_i with coefficients in $E(t)$. This system is equivalent to an n^{th} order linear differential equation. If we could answer question 1 for $E(t)$, then we could solve this equation.

The second place these questions arise is in the general problem of finding liouvillian solutions of linear differential equations with liouvillian coefficients. In [SING81], I showed that given a homogeneous linear differential equation $L(y) = 0$ with coefficients in F , a finite algebraic extension of $\mathbb{Q}(x)$, one can find in a finite number of steps, a basis for the vector space of liouvillian solutions of $L(y) = 0$. I would like to extend this result to find, given a homogeneous linear differential equation with coefficients in a liouvillian extension K of $\mathbb{Q}(x)$, a basis for the liouvillian solutions of $L(y) = 0$. One can show that to solve this problem, it is sufficient to find one non-zero liouvillian solution. An inductive procedure would then allow one to find all such solutions. To see how problem 2 fits into this, I will outline the procedure to decide if a given $L(y) = 0$ with coefficients in K has a non-zero liouvillian solution. It is known [SING81] that if $L(y) = 0$ has a non-zero liouvillian solution, then there is a solution y such that $u = y'/y$ is algebraic over K of degree bounded by an integer N that depends only on the order of $L(y)$. Furthermore there are effective estimates for N . Therefore, for some $m \leq N$, u satisfies an irreducible equation of the form $f(u) = u^m + a_{m-1}u^{m-1} + \dots + a_0 = 0$ with the $a_i \in K$. We must now find the possible $a_i \in K$ and test to see if, for such a choice of a_i , $e^{\int u}$ satisfies $L(y) = 0$. For example, let us try to determine the possible a_{m-1} . If $u = u_1, \dots, u_m$ are the roots of $f(u) = 0$ and $y_1 = e^{\int u_1}$ satisfies $L(y) = 0$, then for $i = 2, \dots, m$, $y_i = e^{\int u_i}$ also satisfies $L(y) = 0$. We have

$$a_{m-1} = -(u_1 + \dots + u_m) = -\left(\frac{y_1'}{y_1} + \dots + \frac{y_m'}{y_m}\right) = -\left(\frac{(y_1 \cdots y_m)'}{y_1 \cdots y_m}\right).$$

One can show that the product $y_1 \cdots y_m$ satisfies a homogeneous linear differential equation $L^{\otimes m}(y) = 0$ and that $y'/y \in K$. Finding all such solutions is just problem 2 above. Theorem 2 below states that for certain liouvillian extensions K , we can fill in the details of the above argument and give a procedure to find a basis for the vector space of liouvillian solutions of $L(y) = 0$.

The first result of [SING88] states that we can reduce Question 2 to Question 1. We shall consider fields of the form $E(t)$, where either $t' \in E$, $t'/t \in E$ or t is algebraic over E and where E

satisfies certain hypotheses. We can show that for these fields, if we can answer question 1 algorithmically then we can answer question 2 algorithmically. To make this precise, we need some definitions. We call a differential field K a computable differential field if the field operations and the derivation are recursive functions and if we can effectively factor polynomials over K . We say that we can effectively solve homogeneous linear differential equations over K if for any homogeneous linear differential equation $L(y) = 0$ with coefficients in K , we can effectively find a basis for the vector space of all $y \in K$ such that $L(y) = 0$. We say that we can effectively find all exponential solutions of homogeneous linear differential equations over K if for any homogeneous linear differential equation $L(y) = 0$ with coefficients in K , we can effectively find $u_1, \dots, u_m$ in K such that if $L(e^{\int u}) = 0$ for some $u \in K$, then $e^{\int u} = e^{\int u_1} \dots e^{\int u_m}$ for some i . The precise result is:

Proposition 1. Let $E \subset E(t)$ be computable differential fields with $C(E) = C(E(t))$, an algebraically closed field, and assume that either $t' \in E$ or $t/t \in E$ or t is algebraic over E . Assume that we can effectively solve homogeneous linear differential equations over $E(t)$ and that we can effectively find all exponential solutions of homogeneous linear differential equations over E . Then we can effectively find all exponential solutions of homogeneous linear differential equations over $E(t)$.

The proof of this is contained in [SING88], but I will give an example of the algorithm below. First recall some facts about the Riccati equation. If u is a differential variable and $y = e^{\int u}$, formal differentiation yields $y^{(i)} = P_i(u, u', \dots, u^{(i-1)}) e^{\int u}$, where the P_i are polynomials with coefficients satisfying $P_0 = 1$ and $P_i = P_{i-1}' + u P_{i-1}$. If $L(y) = y^{(n)} + A_{n-1}y^{(n-1)} + \dots + A_0 = 0$ is a linear differential equation, then $y = e^{\int u}$ satisfies $L(y) = 0$ if and only if u satisfies $R(u) = (u, \dots, u^{(n-1)}) + A_{n-1}P_{n-1}(u, \dots, u^{(n-2)}) + \dots + A_0 = 0$. This latter equation is called the Riccati equation associated with $L(y) = 0$.

Example 1. Let $E = \mathbb{Q}(x)$ and $t = \log x$. We shall consider the differential equation

$$L(y) = y'' - \frac{1}{x(\log x + 1)} y' - (\log x + 1)^2 y = 0$$

and decide if it has solutions of the form $e^{\int u}$ with $u \in E(t)$. We shall assume that the hypotheses of the theorem are satisfied by E and proceed to find the partial fraction decomposition of u . The associated Riccati equation is

$$R(u) = (u' + u^2) + \frac{1}{x(\log x + 1)} u - (\log x + 1)^2 = 0.$$

Assume that u is a solution of $R(u) = 0$ in $E(t) = \mathbb{Q}(x, \log x)$. If $p(t) \neq t+1$ is irreducible in $E[t]$,

then the order of u at $p(t)$ is bigger than or equal to -1 . Furthermore, if the order is -1 , then the leading term is p'/p . At $\log x + 1$, we may write

$$u = \frac{u_\gamma}{(\log x + 1)^\gamma} + \frac{u_{\gamma-1}}{(\log x + 1)^{\gamma-1}} + \dots$$

Substituting this expression in $R(u)$ and comparing leading terms, one sees that if $\gamma > 1$, then the leading term in $R(u)$ is $u_\gamma (\log x + 1)^{2\gamma}$. If $\gamma = 1$, then the leading term (after some cancellation) is $u_1^2 (\log x + 1)^2$. This means that u cannot have a pole at $\log x + 1$. We therefore have that $u = \sum \frac{p'_i}{p_i} + s$ where the p_i are irreducible polynomials in $E[t]$, not equal to $t+1$ and s is a polynomial in $E[t]$. We now proceed to determine $s(t) = s_m t^m + \dots + s_0$. Plugging into $R(u)$ and comparing terms we see that $m = 1$ and $s_1 = \pm 1$ and so $s(t) = \pm t + s_0 = \pm \log x + s_0$. We therefore alter $L(y)$ in two ways. Let $L_1(y) = L(ye^{-\int \log x})/e^{-\int \log x} =$

$$y'' + \frac{2x \log^2 x + 2x \log x - 1}{x \log x + x} y' + \frac{-2x \log^2 x - 3x \log x - x + 1}{x \log x + x} y$$

Let $L_2(y) = L(ye^{\int \log x})/e^{\int \log x} =$

$$y'' + \frac{-2x \log^2 x - 2x \log x - 1}{x \log x + x} y' + \frac{-2x \log^2 x - 3x \log x - x - 1}{x \log x + x} y.$$

To determine the possible s_0 we consider L_1 and L_2 separately. In both cases we are looking for solutions of this equation of the form $y = e^{\int s_0 + (-\sum \frac{p'_i}{p_i})}$ with s_0 in E . For L_1 , if we expand the coefficients in decreasing powers of $\log x$, we get

$$L_1(y) = y'' + (2 \log x + \dots) y' + (-2 \log x + \dots) y = 0.$$

$e^{\int s_0}$ will satisfy $\hat{L}_1(y) = 2y' - 2y = 0$. By the hypotheses, we can find exponential solutions of this latter equation over $E = \mathbb{Q}(x)$. In fact, e^x is the only such solution i.e. the only possibility for s_0 is 1. We now modify $L_1(y)$ and form $\tilde{L}_1(y) = L_1(ye^x)/e^x =$

$$y'' + \frac{2x \log^2 x + 4x \log x + 2x - 1}{x \log x + x} y'.$$

We are looking for solutions of this latter equation of the form $r(t)\exp(-(\sum \frac{p_i'}{p_i}))$ with $r(t) \in E(t)$, that is solutions in $E(t)$. A partial fractions argument shows that the only such solutions are constants. This implies that our original equation has a solution of the form $e^{\int \log x + x} = e^{x \log x}$. Repeating this procedure for $L_2(y)$ would yield a solution of our original equation of the form $e^{\int -\log x - x} = e^{-x \log x}$.

I now turn to the problem of effectively answering question 1 for fields of the form $E(t)$ where E satisfies a suitable hypothesis and either $t'/t \in E$, $t' \in E$ or t is algebraic over E . I actually deal with a slightly more general question related to the following definition. Let K be a differential field. We say that we can effectively solve parameterized linear differential equations over K if given $a_{n-1}, \dots, a_0, b_m, \dots, b_0$ in K , one can effectively find $h_1, \dots, h_r$ in K and a system $\mathcal{L}$ in $m+r$ variables with coefficients in $C(K)$ such that $y^{(n)} + a_{n-1}y^{(n-1)} + \dots + a_0y = c_1b_1 + \dots + c_mb_m$ for $y \in K$ and c_i in $C(K)$ if and only if $y = y_1h_1 + \dots + y_rh_r$ where the $y_i \in C(K)$ and $c_1, \dots, c_m, y_1, \dots, y_r$ satisfy $\mathcal{L}$. Obviously, if K is computable and we can effectively solve parameterized linear differential equations over K , then we can effectively solve homogeneous linear differential equations over K . In [SING88], I show the following:

Proposition 2. Let $E \subset E(t)$ be computable differential fields with $C(E) = C(E(t))$. Assume that we can effectively solve parameterized linear differential equations over E .

a) If t is algebraic over E or if $t' \in E$, then we can effectively solve parameterized linear differential equations over $E(t)$.

b) If t is transcendental over E and $t'/t \in E$, assume that we can effectively find all exponential solutions of homogeneous linear differential equations over E and that for any $u \in E$, we can decide if $y' + uy = 0$ has a nonzero solution in $E(t)$ and find all such solutions if one exists. Then we can effectively solve parameterized linear differential equations over $E(t)$.

A few words need to be said about the assumption in the previous proposition that for $u \in E$ we can decide if $y' + uy = 0$ has a solution in $E(t)$. A priori, this is stronger than the assumption that we can decide effectively find all exponential solutions or all solutions of homogeneous linear differential equations over E . Since $t'/t \in E$, it is known ([ROS76], Theorem 2) that any solution in $E(t)$ of $y' + uy = 0$ must be of the form $y_n t^n$ for some integer n . y_n will then satisfy $y_n' + (u + n(t'/t)) y_n = 0$. We are therefore asking to decide if there is some integer n such that this latter equation has a non-zero solution in E . Similar problems come up in the Risch algorithm for integration in finite terms (we are asking if $\int u = \log y_n + n \log t$ for some y_n and integer n). We do not know how to reduce this question to the assumptions that we can effectively find all exponential solutions or effectively solve homogeneous linear differential equations. We are able

to show in [SING88] that this condition holds when E is an elementary extension of $C(x)$, $x'=1$ and C a computable field of constants or when E is a purely transcendental liouvillian extension of $C(x)$. I am not able to show that this condition holds for arbitrary liouvillian extensions of $C(x)$ and the difficulty is related to the problem of parameterized integration in finite terms mentioned in [DAS186]. I will illustrate part b of the above proposition with the following example.

Example 2. Let $E = \mathbb{Q}$ and $t = e^x$. Consider the linear differential equation

$$L(y) = y'' + \frac{-24e^x - 25}{4e^x + 5}y' + \frac{20e^x}{4e^x + 5}y = 0$$

We wish to find all solutions of this equation in $\mathbb{Q}(e^x)$. Using p -adic expansions for $p \neq t$, one can easily show that any solution must be of the form $y_\gamma t^\gamma + \dots + y_\delta t^\delta$. We therefore clear denominators in the above differential equation and consider

$$(1) \quad (4t + 5)y'' + (-24t - 25)y' + 20ty = 0$$

Comparing highest powers of t , we see that $y_\delta t^\delta$ satisfies $4y'' - 24y' + 20y = 0$. We use the hypotheses of the proposition to find solutions of this equation that are exponential over E and see that e^{5x} and e^x are such solutions. We use the other induction hypotheses to decide if $y' - 5y = 0$ and $y' - y = 0$ have solutions in $E(t)$, and see that both e^{5x} and e^x are in $\mathbb{Q}(e^x)$. Therefore $\delta \leq 5$. Comparing lowest powers of t , we see that $y_\gamma t^\gamma$ satisfies $5y'' - 25y' + 20y = 0$. This latter equation has solutions e^{4x} and e^x in $\mathbb{Q}(e^x)$. Since $\gamma \geq 0$, we conclude that either $\gamma = 0$ or $y_\gamma = 0$. Therefore $y = y_5 t^5 + \dots + y_0$ for some y_i constants. If we substitute this expression in (1) we get the following

$$-12y_4 t^5 + (-20y_4 - 16y_3)t^4 + (-30y_3 - 12y_2)t^3 + (-30y_2)t^2 + (-20y_0 - 20y_1)t = 0$$

Equating powers of t to 0 and solving gives us that $y_2 = y_3 = y_4 = 0$ and $y_0 = y_1$. Therefore, solutions of (4) in $E(t)$ are of the form $c_1 e^{5x} + c_2 (e^x + 1)$ where c_1 and c_2 are arbitrary constants.

Turning to part a of the above proposition, we note that when t is algebraic over E , methods similar to those described in ([SING81], Proposition 3.3) yield the desired result. When $t \in E$, the key to the proposition is the following result.

Lemma. Let $E \subset E(t)$ be computable differential fields with $C(E) = C(E(t))$, t transcendental over E and $t' \in E$. Assume that we can effectively solve parameterized linear differential equations over E . Let $A_n, \dots, A_0, B_m, \dots, B_1 \in E[t]$. Then we can effectively find an integer M such that if $Y = y_0 + \dots + y_\gamma t^\gamma$, $y_\gamma \neq 0$, is a solution of

$$(2) \quad A_n Y^{(n)} + \dots + A_0 Y = c_m B_m + \dots + c_1 B_1$$

for some $c_i \in C(E)$ then $\gamma < M$.

The idea behind this lemma is the following. Let $A_i = a_{i\alpha} t^\alpha + \dots + a_{i0}$, $B_i = b_{i\beta} t^\beta + \dots + b_{i0}$. If we formally substitute $Y = y_\gamma t^\gamma + \dots$ into (2) and equate coefficients, we see that y_γ satisfies $\sum_{i=0}^n a_{i\alpha} y_\gamma^{(i)} = 0$. Our induction hypothesis allows us to find $z_{\gamma 1}, \dots, z_{\gamma r_\gamma}$ such that $y_\gamma = \sum c_{\gamma i} z_{\gamma i}$ where $c_{\gamma i}$ are undetermined constants. Replacing y_γ by this expression we move on to attempt to find $y_{\gamma-1}$. $y_{\gamma-1}$ satisfies a linear differential equation whose coefficients depend on the $c_{\gamma i}$ and γ . The existence of a solution of this differential equation turns out to be equivalent to γ satisfying a polynomial equation $f_1(\gamma) = 0$. If f_1 is not identically zero, this places a restriction on γ and allows us to bound γ . If f_1 is identically zero, then we proceed to try and determine $y_{\gamma-2}$. $y_{\gamma-2}$ also satisfies a linear differential equation, whose solvability is equivalent to γ satisfying $f_2(\gamma) = 0$. If f_2 is not identically zero, we can bound γ , otherwise we must continue. In [SING88], we show that at some point we must have f_n being not identically zero. Therefore this procedure terminates. At present we are unable to give an apriori estimate of when this procedure does terminate; we do not even know if it is primitive recursive. Clearly, such a bound would be desirable. We illustrate this sketchy description with an example.

Example 3. Let $E = \mathbb{Q}(x)$ and $t = \log x$. Let

$$L(y) = (x^2 \log^2 x) y'' + (x \log^2 x - 3x \log x) y' + 3y = 0$$

We will look for solutions y of $L(y) = 0$ in $E(t) = \mathbb{Q}(x, \log x)$. Considering y as a rational function of t , we see that the only possible irreducible factor of the denominator is $t = \log x$. If we expand y in powers of $\log x$ and write $y = y_\alpha / (\log x)^\alpha + \dots$, we see that the leading coefficient in $L(y)$ is $y_\alpha [\alpha(\alpha+1)(1/x)^2 - 3(1/x)^2(-\alpha) + 3/x^2]$. Since this must equal zero, we have that $\alpha(\alpha+2) = 0$. Therefore $\alpha = 0$. This means that any solution of $L(y) = 0$ in $E(t)$ is actually in $E[t]$. We let $y =$

$y_\gamma t^\gamma + y_{\gamma-1} t^{\gamma-1} + \dots$ and substitute into $L(y) = 0$. Calculating the coefficients of powers of t , we get the following:

ℓ	Coefficient of t^ℓ
$\gamma+2$	$L_\gamma(y_\gamma) = x^2 y_\gamma'' + x y_\gamma'$
$\gamma+1$	$L_{\gamma-1}(y_{\gamma-1}) = x^2 y_{\gamma-1}'' + x y_{\gamma-1}' + (2\gamma x - 3x) y_\gamma'$
γ	$L_{\gamma-2}(y_{\gamma-2}) = x^2 y_{\gamma-2}'' + x y_{\gamma-2}' + (2\gamma x - 5x) y_{\gamma-1}' + (\gamma^2 - 4\gamma + 3) y_\gamma$

It is easy to see that $L_\gamma(y_\gamma) = 0$ has only constant solutions in E . Replacing y_γ by $c_{\gamma,1} \cdot 1$ in $L_{\gamma-1}(y_{\gamma-1})$ yields the equation $x^2 y_{\gamma-1}'' + x y_{\gamma-1}' = 0$ for $y_{\gamma-1}$. This new equation has only constant solutions in E and places no restrictions on γ . We let $y_{\gamma-1} = c_{\gamma-1,1} \cdot 1$ and substitute in the expression $L_{\gamma-2}(y_{\gamma-2})$. We obtain

$$x^2 y_{\gamma-2}'' + x y_{\gamma-2}' + (\gamma^2 - 4\gamma + 3) c_{\gamma,1} = 0.$$

Since $c_{\gamma,1} \neq 0$, this latter equation has a solution in E if and only if $\gamma^2 - 4\gamma + 3 = 0$. This implies that $\gamma \leq 3$. Therefore $y = y_3 t^3 + y_2 t^2 + y_1 t + y_0$. Substituting this expression into $L(y) = 0$ and calculating the coefficients of powers of t , we find:

ℓ	Coefficient of t^ℓ
5	$L_3(y_3) = x^2 y_3'' + x y_3'$
4	$L_2(y_2) = x^2 y_2'' + x y_2' + 3x y_3'$
3	$L_1(y_1) = x^2 y_1'' + x y_1' + x y_2'$
2	$L_0(y_0) = x^2 y_0'' + x y_0' - x y_1' - y_2$
1	$-3x y_1'$
0	$3 y_0$

Successively setting these expression equal to zero and finding solutions in E yields that y_3 and y_1 are arbitrary constants and y_2 and y_0 are 0. Therefore all solutions of $L(y) = 0$ in $Q(x, \log x)$ are of the form $c_1(\log x)^3 + c_2 \log x$.

Using the results of the above propositions, I can answer questions 1 and 2 for certain

classes of fields.

Theorem 1 Let C be a computable field and assume that either:

- (i) K is an elementary extension of $C(x)$ with $x' = 1$ and $C(K) = C$, or
- (ii) K is an algebraic extension of a purely transcendental liouvillian extension of C with $C(K) = C$.

Then one can effectively find exponential solutions of homogeneous linear differential equations over K and effectively solve parameterized linear differential equations over K .

Using these results and techniques similar to those in [SING81], I can also show

Theorem 2 Let C and K be as in Theorem 4.1 with C algebraically closed. If $L(y) = 0$ is a homogeneous linear differential equation with coefficients in K , then one can find a basis for the space of solutions of $L(y) = 0$ liouvillian over K .

REFERENCES

- [BRON87] Manuel Bronstein, Integration of Elementary Functions, Ph.D. Thesis, University of California, Berkeley, 1987
- [DASI86] J.H. Davenport, M.F. Singer, "Elementary and liouvillian solutions of linear differential equations," J. Symbolic Computation (1986), 2, 237-260.
- [RISCH68] Robert H. Risch, "On the integration of elementary functions which are built up using algebraic operations," SDC Report, SP-2801/002/00, 1968.
- [RISCH69] _____, "The problem of integration in finite terms," Trans AMS, 139, 167-189.
- [RISCH70] _____, "The solution of the problem of integration in finite terms," Bull. AMS, 76, 605-608.
- [ROS76] M. Rosenlicht, "On Liouville's theory of elementary functions," Pacific J. Math. 65, 485-492.
- [ROTH76] M. Rothstein, Aspects of Symbolic Integration and Simplification of Exponential and Primitive Functions, Ph.D. Thesis, Univ of Wisconsin, Madison, 1976.
- [SING81] M.F. Singer, "Liouvillian solutions of n^{th} order homogeneous linear differential equations," Amer. J. Math., 103, 661-681.
- [SING88] _____, "Liouvillian solutions of linear differential equations with liouvillian coefficients," Submitted to the Journal for Symbolic Computation

Recipes for Classes of Definite Integrals Involving Exponentials and Logarithms

K.O. Geddes* and T.C. Scott**

*Department of Computer Science and

**Department of Physics

University of Waterloo

Waterloo, Ontario

Canada N2L 3G1

Abstract. There are many classes of definite integrals for which the corresponding indefinite integral cannot be expressed in closed form whereas the definite integral can be expressed (often in terms of special functions). A computer algebra system should be capable of recognizing a wide variety of definite integrals and, in order to achieve a broad coverage, it is desirable to encode this knowledge in programs which are more general than simple table look-up. By exploiting integral definitions of the various special functions of mathematics and by generalization and differentiation, we are able to derive closed-form solutions for broad classes of definite integrals. In this paper we treat integrals involving exponentials and logarithms. The resulting programs, based on pattern matching and differentiation, are very efficient.

1. Introduction

The indefinite integration problem for elementary functions has been extensively studied in recent decades [Ris69, Tra84, Bro87]. There has also been some progress on algorithms to handle some non-elementary functions [Che85, Che86]. Nonetheless, in mathematical applications there arise many *definite* integrals which can be expressed in closed form (often in terms of special functions) while the corresponding indefinite integral cannot be so expressed.

In a computer algebra system, one approach is to store a table of "common" definite integrals with corresponding symbolic values. Such an approach is limited by the set of integrals stored in the table. It is desirable to have more general programs for dealing with classes of definite integrals.

Wang[Wan71] presented some algorithmic methods for computing definite integrals using, for example, contour integration techniques. Kolbig[Kol85] presents another approach for a specialized class of definite integrals. We propose a scheme which, starting with an integral definition of any particular special function, derives a wide class of definite integrals which can be expressed in closed form. This approach is more general than Kolbig's methods, and the resulting programs execute very quickly compared with Wang's methods. In this paper, we apply the technique for some integrals involving exponentials and logarithms.

This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and in part by grants from the Information Technology Research Centre of Ontario.

2. Integrals Related to the Gamma Function

Consider the standard integral definition of the Gamma function[Abr66]

$$\Gamma(z) = \int_0^{\infty} t^{z-1} \exp(-t) dt, \quad \operatorname{Re}(z) > 0.$$

Of course, if we have an integrand which matches the form of the integrand appearing in this integral definition, with interval of integration from 0 to ∞ , then we can immediately express the result in terms of $\Gamma(z)$. However, we can generalize this integral in several ways.

2.1. Generalizations of the Gamma integral

The first form of generalization is to apply differentiation with respect to z , yielding the formula

$$\frac{d^m}{dz^m} \Gamma(z) = \int_0^{\infty} t^{z-1} \ln(t)^m \exp(-t) dt.$$

This has introduced a logarithmic term into the class of integrals.

Next, let us generalize the exponential term by applying the transformation of variables $t = x^s$ where s is a nonzero real number. Taking into account the sign of s , this yields

$$\frac{d^m}{dz^m} \Gamma(z) = \operatorname{signum}(s) s^{m+1} \int_0^{\infty} x^{sz-1} \ln(x)^m \exp(-x^s) dx.$$

In other words, we now have a formula for a class of integrals where the integrand involves exponentials and logarithms:

$$\int_0^{\infty} t^w \ln(t)^m \exp(-t^s) dt = \operatorname{signum}(s) s^{-(m+1)} \left[\frac{d^m}{dz^m} \Gamma(z) \right]_{z=\frac{w+1}{s}}. \quad (1)$$

In formula (1), the parameters must satisfy:

m is a nonnegative integer;

s is a nonzero real number;

w is a complex number such that $\operatorname{Re}(\frac{w+1}{s}) > 0$.

It is possible to further generalize the "ln" and "exp" arguments appearing in formula (1). Suppose that we have an integral of the form (1) except that the "ln" term has the more general form $\ln(bt^d)^m$, where b is a positive real number and d is any real number. This term can be expressed in the form

$$(\ln(b) + d \ln(t))^m$$

and by expanding in a binomial series, the integral is reduced to a sequence of integrals of the form (1).

Next suppose that we have an integral of the form (1) except that the exp term has the more general form $\exp(-u t^s)$, where u is a positive real number. By applying the change of variables $t = u^{-1/s} x$, the integral reduces to the form (1) but with generalized ln term $\ln(u^{-1/s} x)^m$, which can be treated by the technique just mentioned.

In summary, we can write a program which will express in closed form any definite integral of the form

$$\int_0^{\infty} t^w \ln(b t^d)^m \exp(-u t^s) dt$$

where

m is a nonnegative integer;

b and u are positive real numbers;

d is any real number;

s is a nonzero real number;

w is a complex number such that $\operatorname{Re}(\frac{w+1}{s}) > 0$.

The value of such an integral will be expressed in terms of the Gamma function and its derivatives, which are usually expressed in terms of other special functions.

2.2. Examples

In the examples presented here, simplifications known to the Maple system for the Gamma function and its derivatives evaluated at particular points have already been applied. The examples are presented in the notation of Maple output. In particular, $\text{GAMMA}(z)$, $\text{Psi}(z)$, and $\text{Zeta}(z)$ denote the functions indicated by these names and $\text{Psi}(n, z)$ denotes the n^{th} derivative of $\text{Psi}(z)$. The constants appearing here are P1 (the constant π) and gamma (Euler's constant γ).

$$\int_0^{\infty} \frac{\exp(-t^2)}{t^{1/2}} dt = \frac{\text{P1}}{2 \text{ GAMMA}(3/4)}$$

$$\int_0^{\infty} t \ln(t) \exp(-t^{1/2}) dt = 44 - 24 \text{ gamma}$$

$$\int_0^{\infty} \frac{\ln(t) \exp(-\frac{1}{2t})}{t^3} dt = \frac{1}{48} \pi^2 + \frac{1}{8} \gamma^2$$

$$\int_0^{\infty} \frac{\ln(t) \exp(-\frac{1}{4t})}{t^{3/2}} dt =$$

$$-\frac{1}{64} \Psi(1, 1/8) \Gamma(1/8) + \frac{1}{64} \Psi(1/8) \Gamma(1/8)^2$$

$$\int_0^{\infty} \frac{\ln(3 \frac{1}{t^2}) \exp(-2t)}{t^{1/2}} dt =$$

$$\frac{9}{4} \ln(3^2) - \frac{3}{64} \ln(3^2) (450 - 216 \gamma)$$

$$+ \frac{3}{128} \ln(3^2) (1890 + 108 \pi^2 - 2700 \gamma + 648 \gamma^2)$$

$$+ \frac{243}{16} \zeta(3) - \frac{2025}{256} \pi^2 + \frac{8505}{128} \gamma + \frac{243}{64} \gamma \pi^2$$

$$- \frac{6075}{128} \gamma^2 + \frac{243}{32} \gamma^3 - \frac{1215}{64}$$

3. Integrals Related to the Polygamma Functions

Let $\psi^{(n)}(z)$ denote the n^{th} Polygamma function, defined as the n^{th} derivative of the Digamma (Psi) function:

$$\psi(z) = \frac{\Gamma'(z)}{\Gamma(z)} .$$

Consider the following integral definition for the n^{th} Polygamma function [Abr66]

$$\psi^{(n)}(z) = (-1)^{n+1} \int_0^\infty \frac{t^n \exp(-zt)}{1 - \exp(-t)} dt , \quad \text{Re}(z) > 0 ,$$

where n is a positive integer.

Of course, if we have an integrand which matches the form of the integrand appearing in this integral definition, with interval of integration from 0 to ∞ , then we can immediately express the result in terms of $\psi^{(n)}(z)$. However, we can generalize this integral as follows.

3.1. Generalizations of the Polygamma integral

The first type of generalization used in the previous section was differentiation, but in the present case we gain nothing new by differentiation with respect to z since our class this time is a sequence of derivatives of $\psi(z)$.

We can generalize the exponential terms as in the previous section. Applying the transformation of variables $t = vx$ where v is a positive real number yields

$$\psi^{(n)}(z) = (-1)^{n+1} v^{n+1} \int_0^\infty \frac{x^n \exp(-zvx)}{1 - \exp(-vx)} dx .$$

Evaluating this formula at $z = \frac{u}{v}$ where u is a positive real number, we have the following result for a class of integrals:

$$\int_0^\infty \frac{x^n \exp(-ux)}{1 - \exp(-vx)} dx = \frac{(-1)^{n+1}}{v^{n+1}} \psi^{(n)}\left(\frac{u}{v}\right) .$$

A further generalization of the exponential term is obtained via the transformation of variables $x = t^s$ where s is a nonzero real number. Writing $w = (n+1)s - 1$ and expressing the formula in terms of $m = n+1$ we get (taking into account the sign of s)

$$\int_0^\infty \frac{t^w \exp(-ut^s)}{1 - \exp(-vt^s)} dt = \frac{(-1)^m}{|s| v^m} \psi^{(m-1)}\left(\frac{u}{v}\right) \quad (2)$$

where $m = \frac{w+1}{s}$. In formula (2), the parameters must satisfy:

u, v are positive real numbers;

s is a nonzero real number;

w is a real number such that $m = \frac{w+1}{s}$ is an integer > 1 .

In comparison with the previous section, formula (2) is less general because we have not included a logarithmic term in the integrand. A logarithmic term would be generated by differentiation with respect to the variable w appearing in formula (2), but in this formulation the severe restrictions on w rule out differentiation. The desired generalization is achieved by considering the Riemann Zeta function.

4. Integrals Related to the Riemann Zeta Function

Consider the Zeta function defined by [Abr66]

$$\zeta(z) = \sum_{k=1}^{\infty} \frac{1}{k^z}, \quad \operatorname{Re}(z) > 1$$

and its corresponding integral definition which can be expressed in the form

$$\zeta(z) = \frac{1}{\Gamma(z)} \int_0^{\infty} \frac{t^{z-1} \exp(-t)}{1 - \exp(-t)} dt, \quad \operatorname{Re}(z) > 1.$$

Note that the integral appearing here would fit into the class specified by formula (2) of the preceding section, except that the power of t appearing in the integrand is no longer severely restricted. This allows us to achieve the desired generalizations.

4.1. Generalizations of the Zeta integral

The first form of generalization is to apply differentiation with respect to z to the product function $\Gamma(z)\zeta(z)$, yielding the formula

$$\frac{d^m}{dz^m} \left(\Gamma(z)\zeta(z) \right) = \int_0^{\infty} \frac{t^{z-1} \ln(t)^m \exp(-t)}{1 - \exp(-t)} dt.$$

This has introduced a logarithmic term into the class of integrals.

As before, let us generalize the exponential term by applying the transformation of variables $t = x^s$ where s is a nonzero real number. Taking into account the sign of s , this yields

$$\frac{d^m}{dz^m} \left(\Gamma(z)\zeta(z) \right) = \operatorname{signum}(s) s^{m+1} \int_0^{\infty} \frac{x^{sz-1} \ln(x)^m \exp(-x^s)}{1 - \exp(-x^s)} dx.$$

In other words, we now have a formula for a class of integrals where the integrand involves exponentials and logarithms:

$$\int_0^{\infty} \frac{t^w \ln(t)^m \exp(-t^s)}{1 - \exp(-t^s)} dt = \operatorname{signum}(s) s^{-(m+1)} \left[\frac{d^m}{dz^m} \left(\Gamma(z)\zeta(z) \right) \right]_{z=\frac{w+1}{s}}. \quad (3)$$

In formula (3), the parameters must satisfy:

m is a nonnegative integer;

s is a nonzero real number;

w is a complex number such that $\operatorname{Re}\left(\frac{w+1}{s}\right) > 1$.

It is possible to further generalize the $\ln$ and $\exp$ arguments appearing in formula (3) exactly as in the case of the Gamma integral.

Comparing with formula (2) of the preceding section, formula (3) is less general than desired. While we have managed to introduce a logarithmic term, we would like to allow the exponential terms in the numerator and denominator to be independent of each other in the sense of formula (2). The two-argument Zeta function provides precisely the desired generalization, as we see in the next section.

5. Integrals Related to the Two-argument Zeta Function

Consider the generalized Riemann Zeta function defined by [Gra65]

$$\zeta(z, q) = \sum_{k=0}^{\infty} \frac{1}{(q+k)^z}, \quad \operatorname{Re}(z) > 1$$

and its corresponding integral definition

$$\zeta(z, q) = \frac{1}{\Gamma(z)} \int_0^{\infty} \frac{t^{z-1} \exp(-qt)}{1 - \exp(-t)} dt, \quad q > 0, \quad \operatorname{Re}(z) > 1.$$

Note that $\zeta(z, 1) = \zeta(z)$.

5.1. Generalizations of the Two-argument Zeta integral

We can generalize the exponential terms just as we did for the Polygamma integrals. Applying the transformation of variables $t = vx$ where v is a positive real number yields

$$\zeta(z, q) = \frac{v^z}{\Gamma(z)} \int_0^{\infty} \frac{x^{z-1} \exp(-qv x)}{1 - \exp(-v x)} dx.$$

Evaluating this formula at $q = \frac{u}{v}$ where u is a positive real number, we have the following result for a class of integrals:

$$\int_0^{\infty} \frac{x^{z-1} \exp(-u x)}{1 - \exp(-v x)} dx = \frac{\Gamma(z) \zeta(z, \frac{u}{v})}{v^z}.$$

The next form of generalization is to apply differentiation m times with respect to z yielding the formula

$$\int_0^{\infty} \frac{x^{z-1} \ln(x)^m \exp(-u x)}{1 - \exp(-v x)} dx = \frac{d^m}{dz^m} \left(\frac{\Gamma(z) \zeta(z, \frac{u}{v})}{v^z} \right).$$

This has introduced a logarithmic term into the class of integrals.

A further generalization of the exponential term is obtained via the transformation of variables $x = t^s$ where s is a nonzero real number. Writing $w = sz - 1$ we get (taking into account the sign of s)

$$\int_0^{\infty} \frac{t^w \ln(t)^m \exp(-u t^s)}{1 - \exp(-v t^s)} dt = \operatorname{signum}(s) s^{-(m+1)} \left[\frac{d^m}{dz^m} \left(\frac{\Gamma(z) \zeta(z, \frac{u}{v})}{v^z} \right) \right]_{z=\frac{w+1}{s}}. \quad (4)$$

In formula (4), the parameters must satisfy:

m is a nonnegative integer;

u, v are positive real numbers;

s is a nonzero real number;

w is a complex number such that $\operatorname{Re}(\frac{w+1}{s}) > 1$.

As was the case for the Gamma integral in section 2, it is possible to further generalize the $\ln$ argument appearing in formula (4). Suppose that we have an integral of the form (4) except that the $\ln$ term has the more general form $\ln(b t^d)^m$, where b is a positive real number and d is any real number. This term can be expressed in the form

$$(\ln(b) + d \ln(t))^m$$

and by performing a binomial expansion, the integral is reduced to a sequence of integrals of the form (4).

In summary, we can write a program which will express in closed form any definite integral of the form

$$\int_0^{\infty} \frac{t^w \ln(b t^d)^m \exp(-u t^s)}{1 - \exp(-v t^s)} dt$$

where

m is a nonnegative integer;

u, v, b are positive real numbers;

d is any real number;

s is a nonzero real number;

w is a complex number such that $\operatorname{Re}(\frac{w+1}{s}) > 1$.

The value of such an integral will be expressed in terms of the Gamma function, the generalized Zeta function (which can be expressed, in some cases, in terms of Polygamma functions or the ordinary Zeta function), and their derivatives.

5.2. Examples

We present some examples of the above class of definite integrals. In the results presented, simplifications known to the Maple system for the generalized Zeta function and its derivatives evaluated at particular points have already been applied. Note that in the Maple notation, $\operatorname{Zeta}(n, z)$ denotes the n^{th} derivative of the ordinary Zeta function, while the generalized Zeta function (which doesn't appear explicitly in these examples due to simplifications) would be denoted by $\operatorname{Zeta}(n, z, q)$ where n denotes the order of differentiation.

$$\int_0^{\infty} \frac{t^2 \exp(-5t)}{1 - \exp(-2t)} dt = -\frac{56}{27} + \frac{7}{4} \text{Zeta}(3)$$

$$\int_0^{\infty} \frac{\exp(-1/2 \cdot 1/t)}{t^3 (1 - \exp(-1/t))} dt = \frac{1}{2} \text{Pi}^2$$

$$\int_0^{\infty} \frac{\exp(-t^{1/2})}{t^{1/4} (1 - \exp(-t^{1/2}))} dt = \text{Pi}^{1/2} \text{Zeta}(3/2)$$

$$\int_0^{\infty} \frac{t \ln(t) \exp(-t)}{1 - \exp(-1/2 t)} dt =$$

$$4 (1 - \gamma) (-1 + 1/6 \text{Pi}^2) + 4 \text{Zeta}(1, 2) - 4 (-1 + 1/6 \text{Pi}^2) \ln(1/2)$$

$$\int_0^{\infty} \frac{t^{1/2} \exp(-4t^{1/2})}{1 - \exp(-t^{1/2})} dt = 4 \text{Zeta}(3) - \frac{251}{54}$$

6. Conclusions

We have proposed a scheme for developing programs which are capable of expressing broad classes of definite integrals in terms of special functions and their derivatives. The technique is based on the integral definitions of various special functions, and generalizations obtained via differentiation and transformation of variables. Specifically in this paper, we have shown how to develop programs for the closed-form evaluation of two general classes of integrals involving exponentials and logarithms by exploiting the Gamma, Polygamma, Zeta, and Generalized Zeta functions. Similarly one can exploit other special functions to derive programs for other classes of definite integrals. The programs which result from this approach are based on pattern matching, differentiation, and substitution, and they rely on the computer algebra system's knowledge of special functions. These programs fit nicely into the integration package of a computer algebra system as a "first line of attack" because they execute very quickly.

7. References

Abr66.

M. Abramowitz and I.A. Stegun, *Handbook of Mathematical Functions*, Nat. Bur. Standards Appl. Math. Series 55, US Government Printing Office, Washington, D.C. (1966). (5th printing)

Bro87.

Manuel Bronstein, *Integration of Elementary Functions*, University of California, Berkeley, California (1987). (Ph.D. thesis)

Che85.

G. Cherry, Integration in Finite Terms with Special Functions: The Error Function, *Journal of Symbolic Computation* 1 pp. 283-302 (Sept. 1985).

Che86.

G. Cherry, Integration in Finite Terms with Special Functions: The Logarithmic Integral, *SIAM J. of Computing* 15 pp. 1-21 (Feb. 1986).

Gra65.

I.S. Gradshteyn and I.M. Ryzhik, *Tables of Integrals, Series, and Products*, Academic Press, New York (1965). (4th edition)

Kol85.

K.S. Kolbig, Explicit Evaluation of Certain Definite Integrals Involving Powers of Logarithms, *Journal of Symbolic Computation* 1 pp. 109-114 (Mar. 1985).

Ris69.

R.H. Risch, The problem of integration in finite terms, *Trans. Am. Math. Soc.* 139 pp. 167-189 (1969).

Tra84.

Barry M. Trager, *Integration of Algebraic Functions*, Massachusetts Institute of Technology, Cambridge, Mass. (1984). (Ph.D. thesis)

Wan71.

P.S. Wang, *Evaluation of Definite Integrals by Symbolic Manipulation*, Massachusetts Institute of Technology, Cambridge, Mass. (1971). (Ph.D thesis).

Logic and Computation in MATHPERT: An Expert System for Learning Mathematics

MICHAEL J. BEESON
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE
SAN JOSE STATE UNIVERSITY, SAN JOSE, CA 95192
beeson@ucsc.ucsc.edu

*Abstract*¹

MATHPERT (as in "Math Expert") is an expert system in mathematics explicitly designed to support the learning of algebra, trigonometry, and first semester calculus. This paper gives an overview of the design of MATHPERT and goes into detail about some connections it has with automated theorem proving. These connections arise at the borderline between logic and computation, which is to be found when computational "operators" have logical side conditions that must be satisfied before they are applicable. The paper also explains how MATHPERT maintains and uses an internal model of its user to produce individually tailored explanations, and how it dynamically generates individualized and helpful error messages by comparing user errors to its own internal solution of the problem.

How MATHPERT is to be used in education, and the implications of learning environments like MATHPERT for curriculum and pedagogy, are discussed in Beeson [1989a, 1989d, 1989e]. The details of the design of MATHPERT are discussed in Beeson [1989b].

Overview of MATHPERT

MATHPERT is an expert system in that it is capable of solving (almost) all problems in the stated subject domain internally and autonomously.² In this respect it is similar to existing computer algebra programs such as MACSYMA, MAPLE, or *Mathematica*. However, it differs from them in several respects, which are not only important but vital for education. The most basic of these differences are that MATHPERT is *glass box* and *cognitively faithful*; these terms are explained in detail below, but roughly speaking, they mean that MATHPERT produces not just "answers", but full step-by-step "solutions", with each step intelligible to the user and justified on-screen, and does so not by secret high-powered algorithms, but by the same methods students are supposed to use.

MATHPERT also incorporates a fairly sophisticated *user model*, and uses it to produce step-by-step solutions at a level of detail custom-tailored to the knowledge of the individual user.

MATHPERT is based on the analysis of the stated subject areas into several hundred operators which can be applied to mathematical expressions. When MATHPERT is operating in "menu mode", the user chooses which operator to apply next. The computer carries out the actual application of the operator. Operators which are "well-known" to the student, according to the student model, will be applied automatically, allowing the student to focus on

¹This work partially supported by NSF Grant Number IST-8511176.

²The range of MATHPERT's capabilities is extensive. For example, it can solve problems in simplification, including all kinds of exponents and radicals, factoring, equation solving including transcendental equations, trig identities, limits, differentiation, and integration. It also includes graphics and numerical facilities which will not be discussed at all in this paper.

the less well-known parts of the problem. At any time MATHPERT can be switched into "automatic mode", in which it will not only apply the operator but choose it, thus generating one (or more) steps of the solution automatically. The user can return to "menu mode" at will.

MATHPERT accepts arbitrary symbolic problems from the user; for example, a student might type in her homework. MATHPERT is designed for use with existing courses; whether or not the class is officially using MATHPERT, an individual student should be able to use MATHPERT beneficially. It is designed to be useful to students across the spectrum from those needing remedial work to the very brightest students.

If MATHPERT were supplemented by a few high-powered algorithms, such as Risch-Normann integration and routines for factoring arbitrary polynomials, which are not in the undergraduate curriculum, it could compete with *Mathematica*, *MACSYMA*, *MAPLE*, etc. It is explicitly designed for different purposes and as such should be considered complementary to such programs. MATHPERT cannot do integration or factorization beyond the undergraduate curriculum, which these other programs certainly can. On the other hand, it can give a six-line solution to the common-denominator problem $1/x + 1/y$, explaining every step (or a one-line solution for a more advanced student!), and it can do fairly complicated calculus problems, such as calculating $d/dx \sqrt{x}$ directly from the definition of derivative, showing and justifying every step on a separate line, and handling the logic correctly as well as the symbolic computation. It is these capabilities which set MATHPERT apart from *Mathematica*, *MAPLE*, etc.

Glass Box and Cognitively Faithful

An expert system is called *glass box* if you can see how it arrives at its answer. MATHPERT can print out the individual steps of its solution, with their justifications. (We use the word "solution" to mean such a sequence of intelligible steps, whose last line is the "answer".)

An expert system is *cognitively faithful* if its own internal solutions correspond to the solutions a human would produce. MATHPERT solves math problems in the way we teach students to do, rather than using high-powered algorithms. Cognitive fidelity must be designed for from the beginning, as the demand for cognitive fidelity complicates the construction of an expert system considerably.³

MATHPERT also incorporates an elaborate internal *user model*, or *student model* (but there may well be non-student users). This model contains (among other things), the information concerning which of several hundred pieces of knowledge are, for this user, *well known*, *known*, *learning*, or *unknown*. MATHPERT uses the model to tailor its output to the user. A naive user will receive more detailed explanations than a sophisticated one, and in particular ways tailored to the exact knowledge of that user; a generally sophisticated user with some gaps in her knowledge will still receive detailed explanations when her weak points are involved. This use of the student model to modify output results in MATHPERT's being "cognitively faithful" not just to some idealized student, but to the particular, individual user with whom it is dealing at the moment (provided, of course, that the internal user model is accurate).

The Operator View of Mathematics

³The term *glass box* is in the literature, e.g. in Anderson [1988], Burton and Brown [1982] (where it is credited to a 1977 memo of Goldstein and Papert). The term *cognitively faithful* has probably been used, too: certainly the concept appears in Anderson [1988] and in Wenger [1987] (there under the name "psychological plausibility").

MATHPERT depends on an analysis of its subject matter (algebra, trigonometry, and elementary one-variable calculus) into several hundred operators which can be applied to mathematical expressions. For example, one operator is called *collect powers*. It applies to an expression x^2x^0 and produces x^{11} . The key to the solution of a mathematical problem, according to this view, consists in a correctly-chosen sequence of operators which are to be applied to the input. The "solution" itself is the line-by-line record of the result of these operator applications, together with their "justifications". The justifications are simply the names (or formulas) describing the operators applied.

MATHPERT operates in two "modes": in *menu mode*, the user directs the course of the developing solution by choosing the next operator from a system of menus. Since there are several hundred operators, it would not be practical to require the student to remember and type the names of the operators.⁴ The menu system has been designed to show only those operators which might be relevant to the problem at hand, so that usually the student does not have to leaf through all four hundred operators looking for the right one. Moreover, even in menu mode, "well-known" operators will be applied automatically. Thus while doing integration by parts, for example, you need normally not search for the operator $-(-a) = a$, which should be well-known long before you are tackling integration by parts.

In *automatic mode*, MATHPERT will generate its own "ideal solution", step by step. The user can switch at will between automatic and menu mode. Thus you can start in menu mode, get stuck, switch to automatic mode for one or two steps for a hint; then, back on the track, you can continue in menu mode. When you choose "finished", MATHPERT will switch into automatic mode and see if you really are finished, according to its own internal algorithm. If not it will supply the last steps of the problem. If you switch into automatic mode and stay there, even at the beginning of the problem, MATHPERT will generate the complete solution for you. Thus in principle you could just type in your homework and have MATHPERT print out complete, step-by-step solutions to each problem.

Although automatic mode generates a single "ideal solution", menu mode permits "alternate solution paths": any correct sequence of operators will be accepted. At any point you can switch into automatic mode and MATHPERT will successfully complete the problem from that point, if possible.

The User Model in MATHPERT

The operators used by MATHPERT have been carefully chosen so as to correspond to cognitive skills, that is, to identifiable "chunks" of knowledge which can be taught and learned. Thus the "skills lattice" which is used in cognitive science to model learning can be directly correlated to a user model based on the executable operators of MATHPERT. Each operator is at the same time a procedure and a skill. Although this is important for the educational design of MATHPERT, it is not very important for the logical and design issues raised in this paper. From the present point of view, what is important is that we want the both the program's choice of operators to execute, and the output of the individual operators, to be different for different users (and appropriate to the user at hand).

MATHPERT's internal model of its user consists in recording, for each of some four hundred operators, one of the values *learning*, *known*, *well-known*, or *unknown*. This section will describe how the internal behavior and output of MATHPERT is supposed to depend on these values.

The values *learning* are used by individual operators. For example, we stated above that the operator *collect powers* will produce output x^{11} on the expression x^2x^0 . This is not

⁴Besides, students are generally poor typists. In MATHPERT, they have to type only to enter their problem, not to solve it.

strictly true: if the student is still on record as "learning" this operator, it will produce instead x^{2+9} , after which the operator arithmetic will be automatically applied (if it is well-known) to produce x^{11} on the next line. A substantial fraction of MATHPERT's operators are designed in this way, to produce more explicit versions of their output when certain operators are still recorded as "learning".

The values *well-known*, on the other hand, are used not by individual operators, but by the main MATHPERT system. Even in menu mode, well-known operators are applied automatically. This lets the user concentrate on the task at hand, relieving her of the necessity to search through the menus for an operator she learned two years ago and knows very well how to use. However, these operators will still be visibly applied on the screen. This provides the best of both worlds: the student does not have to think about applying $-(-a) = a$, but also can see explicitly that it was applied. Otherwise, the application of two or three well-known operators (invisibly) can result in confusion.

The values *unknown* are not explicitly used in MATHPERT, but are maintained to allow for the possibility that tutorial software (running on top of MATHPERT) may want to "grey out" unknown operators so that the student can't see or use them. This will be used particularly in the (not uncommon) case that one operator combines the skills represented by several simpler operators. For example, the operator *common denominator* is broken into five or six simpler operators intended for use while learning common denominators.

Evidently the user model is most useful if it is accurate. A program called *DIAGNOSER* will initialize the student model interactively at the student's first serious session. *DIAGNOSER* will generate problems dynamically, based on the student's previous responses; since there are about four hundred operators, dynamic generation rather than a pre-stored test is necessary. Closely related to *DIAGNOSER* will be a program called *EVALUATOR* which will analyze the student's performance with MATHPERT and decide on the correct updating of the user model. At present *DIAGNOSER* and *EVALUATOR* are still in the design stage; they will be implemented in summer 1989.

MATHPERT can be used (and was used in Spring 1989), without *DIAGNOSER* and *EVALUATOR*, if a human tutor (or the user herself) adjusts the student model appropriately to the level of the student in question. It need not be absolutely accurate to be useful.

Symbolic Manipulation in MATHPERT : The Simplifier

For a detailed discussion of the design of MATHPERT, see Beeson [1989b]. Here we give a only a sketch of the design of the symbolic computation engine.

Conceptually the symbolic manipulation part of MATHPERT can be regarded as having two main parts: the individual operators, and the control structure. The individual operators are the repository of knowledge of specific mathematical facts and simple techniques. Each operator is meant to generate one line of a solution. The choice of what operators to apply, and in what order to apply them, is made by the control structure. In menu mode, the control structure is simple: the menu choices directly link to the operators, and there is no automatic control. (But even in menu mode, MATHPERT shifts to auto mode after each menu choice, in order to automatically apply the well-known operators.)

In automatic mode, however, a sophisticated control structure is required. For want of a better name, we call the control structure the "simplifier".

Form of the Operators.

Many of the operators might be expressed as rewrite rules, that is, they can be applied matching the left-hand side of the rule to an expression with free (meta)variables and replacing the expression with the right-hand side of the rule (under the same bindings of the

metavariables). On the other hand, many operators cannot be so construed. A simple example is the rule `collect powers` mentioned above; it has to collect powers of the same variable x even if (a) there are many factors with powers of x , and (b) they are separated by other factors. Moreover, the output involves adding the exponents, not just placing a plus sign between them. (Unless the user is just learning this operator, in which case it does just place a plus sign between the exponents.) MATHPERT allows operators to be defined by arbitrary programs.

The output of an operator is a triple [Next, Yellow, Reason]. The expression Next is the mathematical output, the new expression. The expression Yellow is a copy of Next with some subterms (often the entire term Next) labelled "yellow", so that they can be displayed in a different color to emphasize where the action took place. The third part, Reason, is a "justification" that appears on the right-hand portion of the screen to justify this step in the computation. During ordinary operation of the simplifier, each application of an operator produces output on the screen consisting of Yellow at the left (in white with yellow highlighting) and Reason at the right (in green).

For the benefit of any readers who may not have seen MATHPERT live on the screen, let me repeat that the output is a sequence of lines. Each line consists of a mathematical expression together with a "justification". Each line is obtained from the line above by the application of an operator to some part of the line above.⁵ The use of Yellow is to highlight the part of the expression that has changed.

Control Parameters.

The name "simplifier" is slightly misleading, however, because the behavior of the simplifier itself can be radically different, depending on the type of problem being solved, as well as on the user model. It is well-known, for example, that polynomials have two distinct "normal forms": factored and expanded. For example, $(x+1)^3$ and $x^3 + 3x^2 + 3x + 1$. We thus obtain four distinct possible "normal forms" for rational functions, according as whether the numerator and denominator are to be expanded or factored. In addition there is a fifth normal form for rational functions as a sum of rational functions (partial fraction form). Simplification in mathematics is not as simple as in logic! we can't just specify the normal form and the reduction rules.

This problem is attacked in MATHPERT by maintaining certain "control parameters" that affect the operation of the simplifier. For example, there is one parameter that causes numerators to be expanded (or factored or left alone); one parameter that affects denominators similarly; one parameter that determines whether negative exponents should be converted to positive exponents in the denominator; one parameter that determines when floating-point numbers should be converted to rational fractions, etc. The total number of such parameters is in the vicinity of twenty. One can speak of the "behavior of the simplifier" on a given problem only relative to the settings of these control parameters. Note that *the user does not ever have to adjust or even know about these parameters.*⁶

When the user enters MATHPERT, she must choose what type of problem she intends to solve. The internal effect of this choice is to set certain control parameters. For example, choosing `Factor` will set the simplifier to factor numerators and denominators (or polynomials standing alone), and to expand only products and powers that are part of sums (in hope of additive cancellation). A more subtle example concerns the meaning of variables: each free

⁵A few operators, however, are "jumpers" that actually retrieve previous parts of the computation and use them; for example, when solving several simultaneous equations, you can work on one for a while and then select another.

⁶There is a "control panel" through which the user can, if desired, adjust the control parameters, although this is not normally necessary. For example, you might want to fine-tune the parameters controlling when decimals will be converted to rational numbers and vice-versa.

variable is internally considered either existentially or universally quantified. If the problem type is "solve equations", then the variable or variables to be solved for are considered existentially quantified. This will prevent, for example, differentiating both sides of an equation with respect to such a variable. (Otherwise you can differentiate $x = 1$ with respect to x and derive $1 = 0$; note that if x is a universally quantified variable, it is perfectly legitimate to differentiate $x = 1$ with respect to t , or even x .)

Setting the control parameters is the *only* internal effect of the choice of problem type. As soon as the control parameters are set, the problem is passed to the simplifier, no matter what the initial choice of problem type may be.

Data Types in MATHPERT.

The principal data type in MATHPERT is the *term* or *expression*, which has the syntax

```
term ::= functor(term {,term})
term ::= atom
term ::= number
functor ::= .atom
functor ::= symbol
```

These are in some sense "meta-types". MATHPERT internally keeps track of the types of its "object variables", which may be integer, real, or complex. This is done by using ordinary expressions of the form n : integer. These expressions are rarely seen by the user, but sometimes they appear in the "assumption window", e.g. when solving the equation $\sin x = 0$ we get $x = n\pi$ with the assumption n :integer.

Numbers can be (at the meta-level, i.e. as seen by the programmer), real or complex integers, rationals, or floating-point numbers. Internally a distinction is also made between "ordinary" integers and "bignums" (requiring more than one digit base 2^{16}). Similarly, there are ordinary rationals and "bigrats". (At present there are no "bigfloats": floating point precision is limited to fifteen decimal places.)

An expression can be regarded as a tree, with the functor at the root and the arguments at the daughter nodes. At the leaves of the tree are numbers or symbolic atoms. The tree so constructed is known as the expression tree. Subexpressions of the an expression can be uniquely specified by giving the node of the expression tree at which they occur.⁷

Expression Tree Traversal.

The simplifier works by traversing the expression tree in depth-first fashion, applying operators to nodes as it goes. This design decision only begins to specify the simplifier, however. It leaves several important issues undecided.

Foremost among these issues is the following: When we first visit a node of the expression tree, we have not yet simplified the arguments of the term. Should we then apply what operators we can, or should we wait until we have been down the tree (i.e., have simplified the arguments) and only then apply the operators that apply at this node?

A similar issue arises in logic, specifically in λ -calculus. Choosing to simplify the arguments first (an *inside-first* strategy) corresponds to call-by-value reduction in λ -calculus, while call-by-name results from simplifying arguments only later.

In the case of mathematics, there is no clearcut preference. Examples can be found where inside-first reduction is wrong: consider $A - A$, where A is a complicated expression requiring a lot of simplification. It looks quite silly to simplify A at length only to cancel out the result. On the other hand, often inside-first is correct: generally speaking, we want to simplify the arguments of trig functions before attempting to apply trig identities. Reflection reveals that

⁷Mathematica uses the same natural device. In MATHPERT, however, the specification of expressions by nodes is never done by the user, since MATHPERT was designed to the specification that it require nothing more of the user than a traditional textbook.

in most cases, there are certain simple operators associated with each functor, such that we want to apply these operators *before* simplifying the arguments, if possible, but that most of the operators applicable at a given node should be applied only *after* the arguments are simplified. For example, both additive and multiplicative *cancel* operators should be used (if possible) without simplifying the arguments first; and it is convenient to cancel a double minus sign as soon as you see it, etc. The issue of the order of application of operators came to the fore in the design of MATHPERT because of the demand for cognitive fidelity. If all we needed to see were the answer, it would not matter much whether we simplify A at length before cancelling; efficiency would be the only concern. As it is, since we demand cognitive fidelity of MATHPERT, it is important to get the order of application of operators right, in the sense that solutions produced by MATHPERT "look natural".

Technically, this is accomplished by considering each operator as associated to one (or in some cases more than one) functor. For example, *cancel* is associated to the operator $/$. When the expression tree is traversed, only operators associated with the functor at a given node will be considered.⁸ Each operator is either "pre-associated" or "post-associated" with the functor. Those operators that are pre-associated are applied (if possible) on the way down the expression tree. The post-associated operators are applied on the way back up, i.e. after the arguments have been simplified. Of course the tree changes as it is traversed, thanks to the applications of operators.

This description of the simplifier is conceptually correct but is seriously incomplete. Space limitations here force us to refer the reader to Beeson [1989b] for a discussion of important issues affecting the design of the simplifier, and a discussion of issues of efficiency in the simplifier. Here we have space for only one such issue:

Contextual Dependence of Operators.

In certain cases, it seems that the choice of which operator to apply (or whether to apply a given operator) depends not only upon the expression to which it applies, but also on the context in which that expression occurs. A simple example is the binomial theorem: one feels that by default (i.e. unless the control parameters are set to expand everything), one should not expand $(x + y)^7$. On the other hand, if $(x + y)^7$ is a term in a summand, such as $(x + y)^7 - x^7$, the choice is less clear. For another example, there are three operators which can be applied to $\cos(2x)$, yielding results $1 - 2\sin^2 x$, $\cos^2 x - \sin^2 x$, and $2\cos^2 x - 1$, respectively. The choice of which one to use depends heavily on the context of the occurrence of $\cos 2x$; generally speaking there is one and only one correct choice.

The contextual dependence of some operators seems to speak against the basic design of traversing the expression tree and applying operators associated with certain nodes. However, most examples of apparently context-dependent operators depend only on the expression tree *one level up*, and can simply be associated to the functor at that level, e.g. expanding powers can be associated to the '+' one level above the term to be expanded. The remaining very few truly context-dependent operators simply fetch and examine the context in which their operand occurs. This leaves the *control mechanism* free to pursue a simple tree traversal. To put it in slightly different words: the deviations from pure expression tree traversal are not considered part of the control structure, but as actions performed by the operators being controlled.

Still, not all the work can be pushed off on the operators, because the control structure still must settle the competing claims of several applicable operators, only one of which is the correct one. (Consider the example of the three rules for $\cos 2x$.) This is handled in MATHPERT by allowing an operator to be passed to the simplifier with a "condition" attached.

⁸In reality MATHPERT examines the functors at the daughter node as well when selecting operators to try. The selection process is made efficient by the use of a b-tree.

Instead of a symbolic name OpName, the simplifier gets a pair (OpName,Condition). In this case it is supposed to determine whether the Context (the current line of the computation) satisfies Condition, and only apply OpName in this case. This is the control mechanism by which the major work of context-checking (the verification of Condition) is pushed off on the individual operators.

Indirect Uses of the Simplifier in MATHPERT

The obvious use of the simplifier in MATHPERT is to generate ideal solutions of problems when MATHPERT is in automatic mode. However, there are several other uses as well.

Automatic Application of Well-Known Operators.

When MATHPERT is operating in menu mode, the user directs the computation by choosing the next operator to be applied. After that operator is applied, MATHPERT shifts momentarily to automatic mode, and the simplifier is run on the new expression, but with only the "well-known" operators allowed. This is accomplished by temporarily "inhibiting" all the other operators; the simplifier will not choose an inhibited operator. The result of this is that zero, one, or even several lines of the computation may be generated automatically, if they represent steps well-known to the user. This makes MATHPERT rather pleasant to use: you can focus on the steps that are new to you. While doing integration by parts, if your algebra is well-known, you won't have to search through the menus for operators to accomplish simple algebraic manipulations. As remarked above, the user model affects MATHPERT's operation in two ways: the well-known operators are applied automatically, and the "learning" operators affect the specific lines of output produced (both by themselves and by other operators). The latter effect has to be "hard coded" in the operators themselves; but the former is accomplished systematically by using auto mode with non-well-known operators inhibited.

Dynamic Generation of Error Messages.

A third use of the simplifier is for the dynamic generation of appropriate and helpful error messages. With about 500 operators in MATHPERT, there are about 250,000 pairs of operators. This makes the impossibility of providing pre-canned error messages patently obvious. Experience with MATHPERT shows, however, that there are three main categories of errors:

- (1) Errors in which the wrong key was pressed by mistake.
- (2) Errors in which the user's plan is basically correct, but she has skipped a step.
- (3) Errors in which the user did not know the correct thing to do.

The user of MATHPERT who makes an error of types (1) or (3) has a chance of getting one of a couple hundred canned error messages, if the author has seen that error before and canned a message for it. But most of the time she will get *Sorry, I can't do anything with that operator.*

The user who makes an error of type (2), however, will get an appropriate and helpful message. Let me give an example. Suppose we are working the common-denominator problem $1/x + 1/y$ and we have got as far as

$$\frac{1}{x} \frac{y}{y} + \frac{1}{y} \frac{x}{x}.$$

A natural error is to choose *Add Fractions* $a/c + b/c = (a+b)/c$ at that point. MATHPERT will then generate the error message:

That operator almost applies here, but you must first prepare the way by using 'multiply fractions' $(a/b)(c/d) = ac/bd$ and 'order factors'.

This message gives the user direct advice on how to correct the error. MATHPERT creates such messages by the following mechanism: when an inapplicable operator is chosen, it generates the next four steps of its own "ideal" internal solution. It then looks to see if the student's choice of operator is one of the four operators involved. If so, it assumes it is dealing with an error of type (2), and generates an appropriate message as illustrated above, using the information from the internal solution.

Logic and Computation

The author has long been interested in the interplay between logic and computation in mathematics. Beeson [1988] presents some general ideas which would make good background reading.

Operators with Side Conditions.

A *side condition* is a condition that must be true before an operator can be applied. For example, the operator $(\sqrt{x})^2 = x$ is only valid if $x \geq 0$. Of course, in simple cases when the expression matched to x is just a number, we can simply compute whether the condition is or isn't satisfied. But what should we do when x is not a number?

MATHPERT's answer, which we claim is cognitively faithful, is as follows: When an operator is applicable, but it has a side condition, we proceed as follows:

(1) First attempt to infer the side condition. This attempt at inference will include numerical computation but will also include symbolic computation.

(2) If this fails, then attempt to refute the condition, for example by numerical computation, but again any relevant operators may be applied.

(3) If this too fails, then assume the condition.

Let's see how this works in practice. Say we want to simplify $(\sqrt{x+h})^2 - (\sqrt{x})^2$. The conditions $x+h \geq 0$ and $x \geq 0$ can't be inferred or refuted, so they are assumed, and the operator is applied twice to yield $x+h-x$. Fine, but what has become of the assumptions? A record has been kept of them, and they are on display in a special "assumption window", where the user can peruse the current assumptions if desired.

Here we have crossed the boundary line from computation to logic: now we have a hypothetical result, an implication, not just the result of a computation.

When we have finished the main computation, we may select one of the assumptions, and try to simplify or even to infer it. There turns out to be no way to fence computation off from logic:

- We need operators with side conditions to do calculus.
- We can't handle operators with side conditions properly without a full-blown logical system.

Partial Terms in MATHPERT.

In mathematics we have to deal with partial functions, i.e. functions whose domain is not all of the reals. Similarly, there are other expressions in calculus which do not always denote values, notably terms involving limits. The correct automatic manipulation of such expressions necessarily involves some sort of logical apparatus.⁹

The author has dealt in the abstract with a "Logic of Partial Terms" (see Beeson [1986] or [1985], p. 97). This abstract theory LPT has been built into the logical machinery of MATHPERT. Thus one has expressions of the form $\text{defined}(\text{Exp})$ for every expression

⁹Mathematica lacks such a logical apparatus, which is the fundamental reason why it doesn't handle operators with side conditions properly (see Wolfram [1988], p. 417).

F. These expressions first arise in calculus, when limit problems are considered. Consider a problem like

$$\lim_{h \rightarrow 0} \frac{\sqrt{x+h} - \sqrt{x}}{\sqrt{x-h} - \sqrt{x}}$$

There is an operator `lim_quotient`, as follows:

$$\lim_{h \rightarrow a} \frac{u}{v} = \frac{\lim_{h \rightarrow a} u}{\lim_{h \rightarrow a} v}$$

However, this operator has a side condition, namely that two limits on the right side are defined, and the one in the denominator is different from zero. In **LPT**, this can be expressed using the propositions just discussed. Explicitly, in the internal form used by **MATHPERT**, the side condition in the above example takes the form of the conjunction of the three propositions:

```
defined(lim(h->0,sqrt(x+h) + (-sqrt(x))))
defined(lim(h->0,sqrt(x+(-h)) + (-sqrt(x))))
lim(h->0,sqrt(x+(-h)) + (-sqrt(x))) != 0
```

However, the beauty of the logic **LPT** is that we do not have to explicitly consider the first two of these, as an equality $t = s$ means that t and s are both defined and equal. The atomic proposition $t \neq s$ means that t and s are both defined and unequal. With these ideas built into the logical system, the only remaining side condition is the third one, that the limit of the denominator is not zero.

When **MATHPERT**'s simplifier is given the problem, it generates this side condition, and then attempts to "check" the side condition according to the algorithm given above, i.e. "check" means try to infer, refute, or assume, in that order. But the attempted inference is only allowed limited means: logical and inequality operators. In particular limit operations are not allowed, so the attempt to check the side condition does not result in evaluation of the limits. Since the side condition can be neither inferred nor refuted, it is assumed. The operator is then applied, and the computation can proceed.

In this example, however, the various limit rules (after generating more assumptions) lead to an expression $0/0$, which **MATHPERT** recognizes as undefined. This stops the simplifier from continuing to apply operators to the expression, and sends it back to check the assumptions still outstanding. Previously, when it tried to check the assumption that the limit of the denominator was zero, it failed, because the check algorithm doesn't include evaluating limits. But now, the limit has been evaluated. **MATHPERT**'s simplifier has taken care to keep track of the results of evaluation of subterms, so that check now succeeds in refuting this side condition. The computation therefore backtracks to the place where the rule for the limit of a quotient was applied, and another **MATHPERT** searches for another applicable operator. It so happens there is one: L'Hôpital's rule. Moreover, the side condition for L'Hôpital's rule can now be inferred, since the evaluation of the subterms has already taken place. After this the computation proceeds smoothly.

The most difficult part of the above to achieve mechanically is the correct recording and retention of results about the evaluation of subterms. In what we feel is a quite cognitively faithful mechanism, **MATHPERT** actually does not record this information as it is computed, but extracts it from the record of the computation when it is needed, by tracing subterms backwards to their predecessors in the computation.

Bound Variables in **MATHPERT**.

In the logico-mathematical system underlying **MATHPERT**, there are several ways of binding variables. There are, for example, integrals, derivatives, limits, and indexed sums.

There are also terms for n -th derivatives, and for definite integrals. These terms are the direct analogue of terms that appear in every calculus textbook. We also make use of some terms (internally) that do not appear in books. For example, `defined_in_nbhd(Exp,X,A)` expresses that the expression `Exp`, considered as a function of the variable `X` is defined in a neighborhood of `A`. To take a specific example: `defined_in_nbhd(sqrt(X),X,1)` is true but `defined_in_nbhd(sqrt(X),X,0)` is false. In this expression, the variable `X` is bound. By this (and other similar) devices, MATHPERT avoids the necessity of using the λ -calculus or some equivalent device for constructing names of functions.¹⁰ As a matter of fact, MATHPERT knows (internally) about λ -calculus and can be used to study it, but this has been done only for the author's amusement, and not because it is needed. The quantifiers `all` and `exists` are the other functors which might bind variables, but MATHPERT does not use quantifiers: its logic is quantifier-free.

Interaction of Bound Variables and Operators.

At the boundary between logic and computation we find interesting interactions between bound variables and operators with side conditions. An interesting example where these interactions arise is the following problem:

Calculate $\frac{d}{dx}\sqrt{x}$ directly from the definition of the derivative.

A few steps into the computation, we have an expression whose numerator is

$$\lim_{h \rightarrow 0} ((\sqrt{x+h})^2 - (\sqrt{x})^2).$$

We want to apply the operator $(\sqrt{x})^2 = x$ if $x \geq 0$. If it weren't for the fact that h is bound by the functor `lim`, we would generate the assumption $x+h \geq 0$. It would be incorrect, however, to generate an assumption containing h free. Hence something in the mechanisms described above requires modification.

In fact two modifications are necessary. One, the side condition of the operator in question is not really $x \geq 0$: it is instead `defined(sqrt(X))` (in MATHPERT's notation) (or $\sqrt{X} \downarrow$ for those familiar with LPT's notation). Normally this condition reduces immediately (by the MATHPERT operator called internally `domain_sqrt`) to the proposition $X \geq 0$. However, in case the expression X contains bound variables, the reduction is not so simple. The second modification to check concerns the nature of the reductions applicable to such propositions.

It seems that each functor that can bind variables has to be treated individually. Suppose we have a term `defined(sqrt(X))` in which the expression X contains a variable h bound by a limit operator `lim(h->a,...)`. Then the proposition `defined(sqrt(X))` should reduce to `defined_in_nbhd(sqrt(X),h,a)`. This reduction is accomplished by a certain MATHPERT operator. Incidentally, this operator is another example of a context-dependent operator.

The mechanical apparatus described so far is sufficient to carry out the solution of the problem stated above, computing the derivative of $\sqrt{x}$ directly from the definition of derivative. It comes out correctly to be differentiable only when $x > 0$, since the proposition `defined_in_nbhd(sqrt(x+h),h,0)` reduces to $x+0 > 0$. This last reduction is accomplished by the MATHPERT operator called internally `open_domain_sqrt`.

Implicit in the above remarks is a point that deserves to be made explicit: knowledge of the proposition defining the domains of each of the traditional mathematical functions, and of the proposition defining the interior of these domains, is expected of serious calculus students, and is built into MATHPERT. Although the twentieth-century tradition has wrapped the

¹⁰MATHPERT is cognitively faithful in this respect: you don't see λ -calculus in math books. In essence, this is achieved by keeping all names of functions in normal form.

central role of the defining propositions for the domain of functions in a smokescreen of set-theoretic notation, in fact sets have nothing to do with it; they are just a detour on the direct route from the question whether a function is defined to the proposition whose evaluation will answer that question. The decline of set theory will be one of the concomitants of the more concrete view of mathematics fostered by the widespread availability of unprecedented computational power; particularly in places like this one where set theory was making no real contribution.

Gentzen's Proof Theory.

In Gentzen's proof theory, assumptions are represented as the antecedent of a "sequent", or expression of the form *Assumption-List* $\Rightarrow$ *Proposition*. Formal proofs are described by trees labelled with sequents, such that each node corresponds to an application of one of the rules of inference given by Gentzen. (The conclusions are written below the premises.) It is an interesting question (for a logician turned programmer, at least) exactly what logical formalism is required to support trigonometry, algebra, and calculus.

To support MATHPERT, we need only a quantifier-free formalism, although we do require several variable-binding functors as described above. We need sequents of the form *Assumption-List* $\Rightarrow$ *Proposition*, where *Proposition* is an expression (including an equation or inequality as a special case of expression), and the members of *Assumption-List* are expressions too.

Note that, following Church (and Prolog), the propositions are treated just like any other expressions. This is in contrast to traditional logical systems where an expression $a = b$ is considered a "formula" and is treated differently from an expression $a + b$.

We can view the process of solving a problem as a process of constructing a proof in a Gentzen-like system. For example, we have the proof fragment

$$\frac{\Gamma \Rightarrow E[(\sqrt{x})^2]}{\frac{\sqrt{x} \downarrow, \Gamma \Rightarrow E[x]}{x \geq 0, \Gamma \Rightarrow E[x]}}$$

Note that applications of operators take us *down* the proof tree, i.e. they are a form of *forward* inference. The elaborate control apparatus described above for selecting appropriate operators can thus be thought of as a control structure for guiding forward inference in an otherwise far-too-vast search space.

Connections with Automated Deduction

It has been a long-standing problem in automated deduction to find good ways of combining symbolic computation with logical deduction. MATHPERT had to do so (albeit at an elementary level) just to support automatic problem-solving in calculus.

The author has also written a theorem-proving program GENTZEN based on Gentzen's proof theory. This program embodies an algorithm which constructs proof tree "backwards" from the concluding sequent. An extremely important point is that, although the proof is constructed by proceeding from the conclusions to the hypotheses ("upwards" in the proof tree), part of the conclusion is "left blank" in the beginning and "filled in" as we go up the tree. The technical device used to accomplish this is unification, using Prolog variables (metavariables) to range over terms of the formal language. Beeson [1989c] describes the program in detail. Although it constructs the proof tree from the bottom up, the algorithm is a combination of what intuitively one would call "forward" and "backward" inferences.

The reason for bringing up GENTZEN here is that the simplification algorithm employed in MATHPERT meshes nicely with the theorem-proving algorithm employed in

GENTZEN. The two programs use compatible language and concepts. The theoretical framework can be briefly described: Gentzen sequents and rules, extended so as to incorporate the logic of partial terms, and new variable binding operators, and extended (following Church) to treat propositions as ordinary terms. The author plans to merge the two programs at some point in the future. The algorithm of GENTZEN will be seen to be composed of certain "logical" operators on sequents. The combined algorithm will contain both GENTZEN and MATHPERT, as well as some rules that go beyond either, such as the rules for proof by mathematical induction.

References

- Anderson, John R. [1988], The Expert Module, in: Polson, M. C., and Richardson, J. J. (eds.), *Foundations of Intelligent Tutoring Systems*, pp. 21-54, Erlbaum, Hillsdale, N. J. (1988).
- Beeson, M. [1985], *Foundations of Constructive Mathematics: Metamathematical Studies*, Springer-Verlag, Berlin/ Heidelberg/ New York (1985).
- Beeson, M. [1986] Proving Programs and Programming Proofs, in: Marcus, Dorn, and Weingartner (eds.), *Logic, Methodology, and Philosophy of Science VII*, pp. 51-82, North-Holland, Amsterdam (1986).
- Beeson, M. [1988] Computerizing Mathematics: Logic and Computation, in: Herken, R. (ed.), *The Universal Turing Machine: A Half-Century Survey*, pp. 191-226, Oxford University Press, Oxford/ New York (1988).
- Beeson, M. [1989a], Learning Mathematics with MATHPERT, to appear.
- Beeson, M. [1989b], The Design of MATHPERT: An Expert System for Learning Mathematics, to appear.
- Beeson, M. [1989c], Some Applications of Gentzen's Proof Theory in Automated Deduction, submitted to *Journal of Automated Reasoning*.
- Beeson, M. [1989d], The User Model in MATHPERT: An Expert System for Learning Mathematics, to appear in *Proceedings of the Conference on Artificial Intelligence and Education, Amsterdam, May 1989*.
- Beeson, M. [1989e], MATHPERT: An Expert System for Learning Mathematics, in: *Proceedings of the Conference on Technology in Collegiate Mathematics Education, Columbus, Ohio, November 1988*, Addison-Wesley (to appear).
- Burton, R. R., and Brown, J. S. [1982], An investigation of computer coaching for informal learning activities, in Sleeman and Brown [1982], pp. 79-98.
- Sleeman, D., and Brown, J. S. [1982], (eds.), *Intelligent Tutoring Systems*, Academic Press, London/ Orlando, Fla. (1982).
- Wenger, E. [1987], *Artificial Intelligence and Tutoring Systems*, Kaufmann, Los Altos, Calif. (1987).
- Wolfram, S. [1988], *Mathematica: A System for Doing Mathematics by Computer*, Addison-Wesley, Redwood City, Calif. (1988).

Representation of Inference in Computer Algebra Systems with Applications to Intelligent Tutoring*

Tryg A. Ager

R. A. Ravaglia

Institute for Mathematical Studies in the Social Sciences
Stanford University

Sam Dooley†

University of California, Berkeley

Abstract. *Presently computer algebra systems share with calculators the property that a sequence of computations is not a unified computational sequence, thereby allowing fallacies to occur. We argue that if computer algebra systems operate in a framework of strict mathematical proof, fallacies are eliminated. We show that this is possible in a working interactive system, REQD. We explain why computational algebra, done under the strict constraints of proof, is relevant to uses of computer algebra systems in instruction.*

1 Introduction

In their recent book [2] Davenport, Siret and Tournier state that computer algebra systems should meet two requirements:

1. Provide pre-programmed commands to perform wearisome calculations.
2. Provide a programming language to define extensions or enlargements of the original set of pre-programmed commands.

Recently much attention has been given to the use of computer algebra systems in instruction. We are involved in such a project ourselves [8]. In this paper we wish to suggest that, especially in instructional situations, there should be a third requirement on computer algebra systems:

3. Represent mathematical inferences.

The essence of mathematics is proof. All advanced mathematics instruction depends on proof to present, justify, and apply mathematical concepts. Proof is the structure of mathematical discourse, and the bottom line of acceptability in mathematical problem solving. In an instructional context, it would be desirable to constrain student interactions to the straight and narrow road of proof and detect fallacious reasoning in a pedagogically effective and timely way.

*Research supported by the National Science Foundation Grants MDR-85-50596 and MDR-87-51523 at Stanford University and the Defense Advanced Research Projects Agency (DoD), monitored by the Space and Naval Warfare Systems Command under Contract N00039-88-C-0292 at the University of California at Berkeley.

†Supported under a National Science Foundation Graduate Fellowship.

In contrast, at this point in their development, computer algebra systems are designed as tools, and usually are supplied *caveat emptor*. This means that just as with more concrete tools like hammers—which can drive a nail or crush your thumb—computer algebra tools can perform elegant computations, or lead to contradictions.

Consider the following “proof”—an epitome of the divide-by-zero fallacies:

$$\begin{array}{lll} \text{Assume} & a = 0 & (1) \\ \text{Divide by } a & a/a = 0/a & (2) \\ \text{Simplify} & 1 = 0 & (3) \end{array}$$

This is a trivial example of a nontrivial problem. We know from [5] that zero equivalence is in general unsolvable. So we cannot detect division by zero in a general and algorithmic way.

But there is a different approach: we can reliably audit the operations or steps of inference that lead from (1) to the contradiction (3). Since a requirement of the division operation is that the divisor is not zero, we can add to an audit trail when division is performed. Thus

$$\begin{array}{lll} \text{Assume} & a = 0 & (1) \\ \text{Divide by } a & a/a = 0/a \text{ provided } a \neq 0 & (2') \\ \text{Simplify} & 1 = 0 \text{ provided } a \neq 0 & (3') \end{array}$$

Now a summary of this audited proof in the if... then idiom would be: If $a = 0$ then $1 = 0$, provided $a \neq 0$. So some term b for which zero equivalence is undetermined, either for practical or theoretical reasons, could be used in an audited proof as follows:

$$\begin{array}{lll} \text{Assume} & a = 0 & (1) \\ \text{Divide by } b & a/b = 0/b \text{ provided } b \neq 0 & (2'') \\ \text{Simplify} & a/b = 0 \text{ provided } b \neq 0 & (3'') \end{array}$$

For this proof, where we don't know if b is zero, auditing is the only protection we have. Reversing the auditing metaphor, the conclusion is mortgaged to the proposition that $b \neq 0$; a subsequent audit might show it was a bad loan. Fateman [4] discusses the auditing of symbolic computation along with several other options for dealing with computer algebra fallacies.

We call the provisos on divisors and other entries in such audit trails *restrictions*. The idea we want to develop in this paper is that keeping track of restrictions is a proper strategy for satisfying the third requirement on computer algebra systems. We will look at one other example of inference, then turn to a precise definition of restrictions, and conclude by showing how keeping track of restrictions leads to some nice results in justifying reasoning involving limit computations.

To broaden the intuitive idea of restrictions, consider

$$\text{Define a function } f(x) = (-2)^x \quad (4)$$

$$\text{Differentiate } f'(x) = (-2)^x \log(-2) \quad (5)$$

Both Reduce and Macsyma (*caveat emptor!*) let this happen. As an inference it would be expressed

$$\text{If } f \text{ is } (-2)^x \text{ then } f'(x) \text{ is } (-2)^x \log(-2). \quad (6)$$

As an audited inference it would be expressed

$$\text{If } f \text{ is } (-2)^x \text{ then } f'(x) \text{ is } (-2)^x \log(-2), \text{ provided } f \text{ is differentiable.} \quad (7)$$

Note that statement (6) is false, whereas the restricted statement (7) is vacuously true because in this case f is nowhere differentiable.

With others at Stanford [8] we have developed a prototype system that satisfies the requirement to represent proofs for a subset of the formulas and operations involved in the elementary differential calculus of univariate closed-form functions. Admittedly, not a large fragment of mathematics, but its complexity is sufficient to bring out interesting issues in representing inference in computer algebra systems. This system is called Restricted Equational Derivations (REQD). REQD is part of a larger project to construct a complete teaching system for elementary calculus with an embedded computer algebra system. The entire project involves a major user interface effort and more relevant to this paper, an effort to find abstract formulations of the differential and integral calculus of univariate functions by reference to which the inferences permitted by REQD can be proven to be consistent.

To illustrate the idea of interactive REQD proofs we show actual REQDs for the two motivating examples. One succeeds; the other fails. Annotated user commands are on lines preceded by the `calc>` prompt.

Division by zero example:

```
calc> intro[duce] a = 0
1.    a = 0
calc> 1 D[ivide] E[quals by equals] a
2.    a/a = 0/a
        provided that a is not zero.
calc> 2 [simplify]
2.    1 = 0
        provided that a is not zero.
```

Differentiability example:

```
calc> define f(x) = (-2) ** x
                        x
1.    F(x) = (-2)
        provided that x is rational with odd denominator.
calc> 1 diff[erentiate with respect to] x
                        x
The function F(x) = (-2) is not differentiable because
we cannot find an open interval on which it is defined,
and therefore none on which it is continuous.
```

For the purposes of this paper, we will describe a fragment of REQD that consists of the rational functions, addition, subtraction, multiplication, and division. We omit components of REQD related to differentiation, the use of hypotheses, and algebraic operations that introduce new terms into the derivation. Because we will not include contingent equalities (relations such as $x + y = c$) we can avoid, in this brief paper, complications associated with the treatment of implicit existential quantification.

We will allow exponentiation and trigonometric functions. Proofs will be built up from function definitions, transformations authorized by algebraic or trigonometric identities, and inferences authorized by definitions or theorems from the theory of limits of univariate functions. Think of this system as employed in an instructional setting in a first-year calculus course. Think of it as illustrating an approach to satisfying requirement (3) on computer algebra systems. Remember, however, that it is a severely diminished system, developed in detail to bring out issues related to the representation of proof in computer algebra systems.

2 Definitions

Here are some definitions that describe the structure of proofs in REQD:

1. REQD Derivation:

An REQD *derivation* is a series of steps each of which is either an equation introduction step or a consequence of one or more previous steps by an REQD inference rule.

2. REQD Step:

An REQD *step* is a pair $\langle E, R \rangle$ consisting of an equation E and a (possibly empty) set of restrictions R . We will refer to the i th step in an REQD derivation by the pair $\langle E_i, R_i \rangle$. In an actual implementation, an REQD step ordinarily would contain additional pedagogical and nonlogical information, such as the print representation, line citation, the justification of the step, etc. In this presentation, however, we concentrate on the minimal logical content of steps.

3. Equation Introduction Step:

An *equation introduction step* is a step that is not a consequence of any other step in an REQD. There are several kinds of introduction steps including axioms, theorems, definitions, and assumptions. Logically, introduction steps stand on their own and are self-justifying; although the rationale for assumptions is special. In this brief development, introduction steps will be restricted to definitions of univariate functions.

4. Transformation Step (Inference Rule):

A *transformation* creates a new step S_n from one or more previous steps $\{S_i, \dots, S_k\}$, called the *premises* of the rule, each of which is an equation-restriction pair as defined above. The equation E_n of the new step is generated by a rewrite rule from the previous equations $\{E_i, \dots, E_k\}$. The restriction set R_n of the new step is generated by a restriction transformation from the previous restriction sets $\{R_i, \dots, R_k\}$. Thus an REQD inference rule is specified by an equational rewrite rule and a restriction transformation.

The mathematical content of any REQD system is determined by its set of introduction and transformation rules. As will become apparent, the equational rewrite rules are purely syntactic, but the restriction transformations are subtle and mathematically substantive. The correctness of any REQD rule depends on completely and correctly implementing the substantive content of the underlying mathematical theorems. This always turns out to be mainly a problem in the transformation of restrictions.

5. Categorical REQD:

An REQD is *categorical* if its introduction rules include only axioms, theorems, or definitions. Categorical REQDs do not have hypotheses or assumptions, and thus are quite weak in terms of applicability, but are useful for isolating certain mathematical phenomena such as taking limits. The abbreviated system we use in this paper to illustrate the behavior of REQDs, is categorical because its only introduction rule is univariate function definition.

6. Restrictions:

The *restrictions* R_n on the n th REQD step are boolean predicates $r_1(t), \dots, r_j(t)$ which are applied to atomic terms that occur in the equation of step n or in the equations of

Operation	Example	Restriction
Division	$\frac{(expr_1)}{(expr_2)}$	$expr_2 \neq 0$
Square Roots	$\sqrt{(expr_1)}$	$expr_1 \geq 0$
Natural Logarithms	$\log(expr_1)$	$expr_1 > 0$
General Logarithms	$\log_{(expr_1)}(expr_2)$	$expr_1 > 0 \wedge$ $expr_1 \neq 1 \wedge$ $expr_2 > 0$
Exponentiation	$(expr_1)^{(expr_2)}$	$expr_1 > 0 \vee$ $expr_1 = 0 \wedge expr_2 > 0 \vee$ $expr_1 < 0 \wedge expr_2 \in \mathcal{O}$
Trigonometric Functions	$\tan(expr_1)$ $\cot(expr_1)$ $\sec(expr_1)$ $\csc(expr_1)$	$\cos(expr_1) \neq 0$ $\sin(expr_1) \neq 0$ $\cos(expr_1) \neq 0$ $\sin(expr_1) \neq 0$

Table 1: Restrictions generated for operations that appear in expressions.

steps from which step n was inferred. In the REQD system discussed here, and in the full system being built for calculus instruction, the restrictions constrain steps to the real numbers in the usual ways. Dooley [3] discusses the motivation for these constraints and surveys alternatives to the constraints presented here.

R_n expresses the domain of the n th step in the sense that it specifies a subset $D_n = \{x \mid \forall i r_i(x)\}$ of $\mathbb{R}$. Logically, R_n acts as an antecedent or presupposition of the truth of the equation, viz., conditions involving the variables and expressions arising in the derivation that must be true in order for the reasoning so far to be valid.

3 A Simple Categorical REQD System

This particular REQD system, which is intentionally left nameless, allows us to define univariate functions, manipulate them according to universally true algebraic identities, and take two-sided and one-sided limits. Its interest lies in (a) the interaction of algebraic manipulations with restrictions, and (b) the essential role restrictions play in checking the correctness of operations that take limits of univariate functions.

To develop a proof interactively in the REQD fragment described here, it is necessary first to define a function. This definition becomes the first step in the proof. Since the intended interpretation of this REQD system is univariate functions on $\mathbb{R}$, any atomic terms in the function definition other than the function parameter and the independent variable will be constants. The step created by function definition will also be restricted as determined by applying the restriction generator to the body of the function definition. Thus the specification sketch of the only introduction rule in this REQD system is (a) E_n is a canonical univariate function definition of the form $f(x) = tm(x)$ (the only free variable in $tm(x)$ is x), and (b) R_n is the set of $r_i(x)$ produced by using Table 1 at each level of the expression tree that represents $tm(x)$.¹

¹In the table, as in [3], the set $\mathcal{O}$ is the set of all rational numbers whose denominator, when reduced to lowest terms, is an odd integer.

Once under way, a proof in the present system may be continued by applying algebraic identities, replacement of identicals by identicals, defining additional functions, or taking limits. These particular rules are specified to operate in the usual ways, so we do not detail them here. In the full REQD system, many additional operations are possible.

In order to prove that REQDs represent valid inferences, we need to show the system is consistent. In general, a system of inference is consistent if and only if (a) every basis step is true under the intended interpretation, and (b) every inference step preserves truth under the intended interpretation. For the present REQD system for calculus, the intended interpretation is onto $\mathfrak{R}$. Consistency proofs are nontrivial, and not the subject of this paper, but one strategy is to prove isomorphism with a consistent set-theoretic formulation of the calculus.² A consequence of consistency is that any inconsistent equation, $1 = 0$ for example, will have a null domain, as is illustrated by the divide-by-zero examples in this paper.

4 Restriction Management

The restriction manager that has been implemented for REQD provides facilities for generating the restrictions that should apply to a newly introduced function, recording the restrictions that are added by applying REQD rules of inference, and simplifying the restrictions that apply to a given step. The restrictions are used in certifying or rejecting a proposed inference step, as is shown in the following frequently discussed example:

```
calc> define f(x) = (x**2)**(1 / 2)
                1/2
                2
1.   F(x) = (x )
    with no restrictions.
calc> 1 subst[itute] -1; [for] x [then simplify]
2.   F(-1) = 1
    with no restrictions.
calc> 1 exprod [products of exponents]
        (2*1/2)
3.   F(x) = x
    provided that x >= 0.
calc> 3 [simplify]
4.   F(x) = x
    provided that x >= 0.
calc> 4 subst[itute] -1; [for] x
-1 does not satisfy the restrictions on step 4
calc> 1 exabs [exponents with absolute value]
        (2*1/2)
5.   F(x) = |x
    with no restrictions.
calc> 5 [simplify]
6.   F(x) = |x|
    with no restrictions.
calc> 6 subst[itute] -1; [for] x [then simplify]
```

²A more sustained look at the issue of consistency of such derivation systems has since been done in [1].

7. $F(-1) = 1$
with no restrictions.

This example, which involves restrictions on exponentiation, illustrates how the laws of exponents are implemented so that they introduce or check restrictions. We remark that this approach offers a theoretically complete way for dealing with the subtleties of exponentiation. Reduce 3.3, for example has a *Precise* flag which controls this behavior for a few preprogrammed cases, but not in general. We believe that handling subtle restrictions is doomed to a case-by-case approach unless the unit of representation is the proof as whole rather than an individual computation on an isolated term.

4.1 Properties of Restriction Generation

In our Categorical REQD fragment, restrictions arise at two well-defined points: when functions are defined and when rules of inference are applied. When a function is defined, the restrictions, which express the domain of the function, can be obtained recursively from the syntax of the defining expression. The restrictions on a composite expression are always the restrictions on its subexpressions combined with the restrictions on the composing operation.

The restrictions thus generated describe the largest possible domain for each symbol that occurs in the expression, taking into account only commitments that are implicit in the expression itself. For example, when defining $f(x) = \sqrt{x}$, the largest possible domain for f is the interval $x \geq 0$, since the domain of the square root function is the set of nonnegative numbers.

At each level in an expression tree, at most a constant number of restrictions is needed to express the domain of the function at that level (cf. Table 1). Therefore, the number of restrictions generated directly from the expression syntax is proportional to the complexity of the expression. In practice many expressions generate no restrictions, and many others can be simplified immediately. For example, it is possible to determine that for all x , $x^2 + 1 > 0$, so no restrictions are needed on $\log(x^2 + 1)$.

In an REQD the rules of inference may introduce new restrictions that express the domain under which the transformation is valid. For example, it is only correct to change $\sqrt{x^2}$ to x under the restriction that $x \geq 0$, as in the previous example. So if you want to simplify the exponent, it will cost you a restriction. A tougher approach would be to say that the exponent simplification does not apply since its restrictions aren't satisfied. The example illustrates still a third option: keep the same domain, apply exponent reduction, but pay the price of an equation acquiring absolute value terms.

Another way rules of inference can generate new restrictions is by introducing new terms. For example, when multiplying both sides of an equation by $\frac{1}{x}$, the restriction must be added that $x \neq 0$.

4.2 Strategies for Restriction Simplification

Even though the number of restrictions generated from expressions that appear in an equation is proportional to the size of the expressions, many trivial and unnecessary restrictions will be generated. Furthermore, even restrictions that cannot be immediately discharged should be simplified for the sake of clarity. Consequently, restriction simplification mechanisms are needed just as normal algebraic simplification is needed for terms in REQD equations.

Also, a proposed inference step may form an inconsistent restriction set. That means the domain of discourse becomes the null set. Logically, the price of inconsistent restrictions on a step is vacuity, not invalidity, since anything follows from false premises. Nevertheless, it is

especially important in an instructional setting to detect inconsistent restriction sets. Therefore, the restriction simplifier also has components to attempt to determine the consistency of a restriction set. If it determines that the restrictions are inconsistent, the system can allow the student to continue or refuse to accept the operation for pedagogical reasons.

Three techniques are currently used to simplify restrictions: (1) algebraic simplification on the terms involved in the restrictions, (2) boolean simplification to combine restrictions on similar terms, both discussed in detail in [3], and (3) bounds propagation, based on [6], to determine upper and lower bounds on the expressions involved in the restrictions using information culled from the other restrictions that are being simplified. Since many restrictions are inequalities, any mechanism for simplifying a set of inequalities can also be used to simplify restrictions. The bounds propagation technique implemented for REQD is relatively efficient to execute, but gives somewhat loose bounds on certain expressions because different instances of the same symbol are analyzed as if they are completely independent.

5 Rules for Limits in REQD

In REQD, limit computations are done using standard limit rules from elementary calculus, such as the limit of a sum is the sum of the limits. These rules compute a limit by decomposing an expression into components for which the limits can be found. However, these rules presuppose that the expression is defined in a deleted neighborhood about the point at which the limit is to be taken. If such a neighborhood does not exist, then there is no guarantee that the limit computed by the rules will be valid. REQD takes this presupposition into consideration by attempting to compute a set of restrictions that guarantee the existence of the deleted neighborhood from the restrictions that guarantee the existence of the expression. The method that REQD uses to compute these restrictions will be discussed in the following sections. The goal here is not to prevent the introduction of all limit statements that do not exist, but to make sure that the deleted neighborhood assumption is satisfied, so that the successful computation of the limit constitutes an existence proof.

5.1 Two-sided Limits

Suppose that in REQD we want to introduce $\lim_{x \rightarrow a} f(x)$, where $f(x)$ is defined at step S_z . Then $f(x)$ will have associated with it a set of restrictions $R_z = \{r_1(x), \dots, r_q(x)\}$ and each of these restrictions will have a corresponding domain set $d_i = \{x \mid r_i(x)\}$. For the limit expression to be valid, we will need to show that $f(x)$ is defined in a deleted neighborhood of a . That is to say we need to show that for some c and d with $c < a < d$, $(c, a) \cup (a, d) \subseteq \bigcap_{i=1}^q d_i$. Testing for this explicitly is difficult. It suffices, however, if we test that none of the restrictions exclude such a neighborhood. The method that REQD uses for this test involves the following steps:

1. Discard any restriction of the form $H_i(x) \neq 0$. The existence of a deleted neighborhood will not be affected by the inclusion or exclusion of a finite number of points. However, some of these restrictions may delete more than a finite number of points; these restrictions will be characterized below.
2. Make all inequalities strict. Using the same motivation as in the first step, a restriction of the form $H_i(x) \geq 0$ can be replaced with $H_i(x) > 0$ without affecting the existence of a deleted neighborhood when only a finite number of points are specified by the equation $H_i(x) = 0$. Other inequalities are handled in a similar way.

3. Evaluate the restrictions at a . After substituting a for x in the modified set of restrictions and simplifying, if the restrictions are all satisfied then the deleted neighborhood exists, subject to the caveat mentioned above. If they are inconsistent, then no such deleted neighborhood exists. If some restrictions cannot be completely simplified then these restrictions form the restrictions on a that guarantee that a deleted neighborhood can be found. These restrictions are added to the new step.

5.1.1 An Example of the Rule

The following is an example of a two-sided limit computation in REQD. Here REQD rejects the first limit since the function is not defined on open intervals on both sides of 3. In the second case, REQD forms the limit but adds the restriction that $a > 3$, guaranteeing that the deleted neighborhood exists.

```
calc> define f(x) = sqrt(x - 3)
1.   F(x) = Sqrt(x - 3)
      provided that x - 3 >= 0.
calc> 1 limit [As] x [Approaches] 3
There are not open intervals of X on both sides of 3
calc> 1 limit [As] x [Approaches] a
2.   lim F(x) = lim sqrt(x - 3)
      x->a      x->a
      provided that a - 3 > 0.
```

5.1.2 Discussion of the Rule

Three problems arise from using the procedure outlined above:

1. A deleted neighborhood may exist outside the domain of the modified restrictions. This situation can occur when a restriction that only excludes a countable number of points is disguised as a restriction that seems to delete an entire interval, and so is not discarded by step 1. Examples are $a^2 > 0$ and $\cos(x) + 1 > 0$. When this type of restriction arises, some limit computations may not be allowed (on the basis of not being able to find a deleted neighborhood) when a deleted neighborhood actually exists, but no extra limit introductions will be allowed.
2. The converse of the first problem also arises. In steps 1 and 2, restrictions are modified under the assumption that only a finite number of points are affected. However, if $g(x) = k$ on some interval, say $[a, b]$, then a restriction of the form $g(x) \neq k$ excludes more than a finite number of points. If such a restriction is mistakenly discarded, a limit could be allowed at a point in (a, b) that would not otherwise be acceptable. An example of this type of restriction is given by $[x] \neq 0$, which deletes the interval $(-1, 0]$. Similarly, when a restriction of the form $g(x) \geq k$ is modified, the entire interval (a, b) will be removed from the available points where the limit could be taken, which may cause the limit mechanism to fail to recognize an existing deleted neighborhood on that interval.
3. A restriction may delete a countably infinite number of points. If a restriction takes the form $g(x) \neq 0$, where the solution set of $g(x) = 0$ is an entire sequence of points (as for the function $\sin \frac{1}{x}$), then discarding the restriction may allow a limit to be taken

at any of the accumulation points of the set $g(x) = 0$, where no deleted neighborhood exists. For $g(x) = \sin \frac{1}{x}$, this problem allows the introduction of $\lim_{x \rightarrow 0} \frac{1}{g(x)}$ even though there is no deleted neighborhood around 0 contained in the domain set specified by the restriction $g(x) \neq 0$.

Note that almost all of the limit exercises that arise in elementary calculus do not exhibit these problems. When one of these problems causes REQD to fail to allow a valid limit introduction, REQD still maintains a consistent proof. Further work is being done on ways to prevent REQD from allowing limit introductions where a corresponding deleted neighborhood fails to exist.

5.2 One-sided Limits

One-sided limits require not that the function in question is defined on a deleted neighborhood about some point but that it is defined only on the appropriate half of the deleted neighborhood. This loss of symmetry creates problems for the test used in the two-sided case. We cannot in general determine by evaluating an inequality $r_i(x)$ at the point a whether it is defined on intervals (b, a) or (a, c) , when it is not defined on both. As such, a different approach is needed. We will discuss the problem of one-sided limits only for limits from the right. The case of limits from the left is similar.

Suppose that an expression $f(x)$ has restrictions $r_1(x), \dots, r_n(x)$. Then $f(x)$ will have a semi-deleted neighborhood on the right of a if and only if $\exists c_0$ s.t. $f(x)$ is defined on $(a, a + c_0)$. The existential quantifier makes this formulation difficult to use, as it does for two-sided limits. One way to get around the existential statement is to introduce a positive constant c which can take on any value in $(0, c_0)$, and where c_0 is taken to be arbitrarily small. The semi-deleted neighborhood will exist if and only if

$$a + \epsilon \in \bigcap_{i=1}^m d_i \iff a + \epsilon \in d_i \forall i \iff \bigwedge_{i=1}^m r_i(a + \epsilon) \text{ is not false.}$$

The object is then to simplify each $r_i(a + \epsilon)$ to a form that does not depend on ϵ , but that expresses the restrictions that $r_i(x)$ places on where a one-sided open interval may be found. By examining the argument of each restriction and using the properties of c , this goal can be accomplished for at least two special cases:

1. The argument of the restriction is a polynomial. Assume that $r_i(x)$ is of the form $P(x) > 0$ (cases for other types of inequalities are similar). Then $r_i(a + \epsilon)$ can be put into the form $f + P'(\epsilon) * c > 0$ where f is some expression not containing c and $P'(\epsilon)$ is a polynomial in ϵ . These inequalities are resolved in the following way:

$$f + P'(\epsilon) * c > 0 \iff (f > 0) \vee (f = 0 \wedge P'(\epsilon) > 0)$$

Since $P'(\epsilon)$ has degree one less than $P(x)$, repeated application of this reduction will give us a set of restrictions not containing c .

2. The argument of the restriction is a rational function. In the case of restrictions which involve rational functions we can use the following procedure to resolve a particular restriction $r_i(a + \epsilon)$: assume that $r_i(x)$ is of the form $H(x) > 0$, where $H(x) = R(x)/Q(x)$, and $R(x)$ and $Q(x)$ are polynomials. (Again, other inequalities are similar.) Now since

$$H(a + \epsilon) > 0 \iff R(a + \epsilon) * Q(a + \epsilon) > 0$$

and $P(a + \epsilon) = R(a + \epsilon) * Q(a + \epsilon)$ is a polynomial, the technique used for restrictions involving polynomials will complete the process of simplifying $r_i(a + \epsilon)$.

These cases allow a complete justification for almost all of the exercises required of first-year calculus students. If each of the restrictions falls in one of these categories, we can compute restrictions which describe exactly where a semi-deleted neighborhood can be found, just as the evaluation of the restrictions in the two-sided case told us when the deleted neighborhood existed. Otherwise, we do not allow the introduction of the limit expression.

5.2.1 An Example

Example of $\lim_{x \rightarrow a+} \sqrt{x^2 - a^2}$. The following represents the internal reasoning that the limit machinery makes when it is given a restriction set.

```
calc> limit [as] x [approaches] a [of] sqrt(x**2 - a**2) [from] +
```

Reasoning internally ...

$$\begin{aligned} r(x) : x^2 - a^2 &\geq 0 \\ r(a + \epsilon) : (a + \epsilon)^2 - a^2 &\geq 0 \Leftrightarrow 2a\epsilon + \epsilon^2 \geq 0 \\ &\Leftrightarrow 2a + \epsilon \geq 0 \\ &\Leftrightarrow 2a > 0 \vee (2a = 0 \wedge 1 \geq 0) \\ &\Leftrightarrow 2a \geq 0 \\ &\Leftrightarrow a \geq 0 \end{aligned}$$

... and so REQD will respond:

1. $\lim_{x \rightarrow a+} \sqrt{x^2 - a^2}$, provided that $a \geq 0$.

5.2.2 Discussion of this Approach

These restrictions become increasingly more difficult to simplify when they are other than polynomial in character. In such cases we must look for adequate simplifications. For differentiable restrictions it may be possible to use some method of approximation, e.g. Taylor series. In cases in which we cannot simplify the restrictions, if the user wants to proceed anyway, REQD will add the audit assumption that the deleted neighborhood exists. These audits can also be simplified, since a later restriction of the form $x < 0$ will rule out the existence of a deleted neighborhood of 3, for example. We prefer to avoid these audit conditions given the difficulty of doing anything useful with them.

5.2.3 An Example involving an Approximation

Example of $\lim_{x \rightarrow a+} \sqrt{\cos(x) - \cos(a)}$. The following represents the internal reasoning that the limit machinery could make given a restriction set that involves non-rational functions.

calc> limit [as] x [approaches] a [of] sqrt(cos(x) - cos(a)) [from] +

Reasoning internally ...

Restrictions: $\cos(x) - \cos(a) \geq 0$.

Evaluating the restriction at $x = a + \epsilon$ and using the first 3 terms of the Taylor Expansion to make the simplifications $\cos(\epsilon) = 1 - \epsilon^2$ and $\sin(\epsilon) = \epsilon$ gives:

$$\begin{aligned} \cos(a + \epsilon) - \cos(a) \geq 0 &\Leftrightarrow \cos(a)\cos(\epsilon) - \sin(a)\sin(\epsilon) - \cos(a) \geq 0 \\ &\Leftrightarrow \cos(a)(1 - \epsilon^2) - \sin(a)\epsilon - \cos(a) \geq 0 \\ &\Leftrightarrow -\sin(a) + \epsilon(-\cos(a)) \geq 0 \\ &\Leftrightarrow \sin(a) < 0 \vee (\sin(a) = 0 \wedge \cos(a) \leq 0) \end{aligned}$$

which is not further reducible given what we currently know.

... and so REQD will respond:

1. $\lim_{x \rightarrow a^+} \sqrt{\cos(x) - \cos(a)}$
provided that $\sin(a) < 0$ or $(\sin(a) = 0 \ \& \ \cos(a) \leq 0)$.

This inequality holds just for $a \in [-\pi \pm 2k\pi, 0 \pm 2k\pi)$, for integral k , which is the correct restriction for the existence of the semi-deleted neighborhood

6 Conclusion

We conclude with several comments about the relevance of REQD systems to instruction—the larger purpose this work on REQDs serves.

1. REQD systems provide a coherent, always consistent, and mathematically natural environment in which to construct derivations interactively. REQD does not prevent a student from trying erroneous inferences, but its design requirement for consistency implies it will detect errors, as the examples herein have shown.
2. Because of restriction management, REQD can detect mathematically subtle errors, while virtually eliminating errors of a stenographic nature.
3. Although the management of restrictions is, in principle, no less complicated than computer algebra itself, restriction management is a distinctly background activity in REQD derivations. Since restriction management can be separated into generation and simplification components, constraints on the degree of simplification can be imposed without affecting validity of the represented inference. Consequently REQD systems can be tuned for computational economy.
4. There is much debate now about the nature of the calculus curriculum [7]. REQD systems with powerful inference rules are consistent with shifting the emphasis of calculus courses more toward analytical rather than computational skills. REQD systems, with a more finely distinguished set of rules are consistent with the traditional emphasis on methods of computation in calculus. But whatever the instructional emphasis, we believe REQD systems bring proof—the essence of mathematics—back into the foreground of calculus instruction.

References

- [1] R. Chuaqui, and P. Suppes. *An Equational Deductive System for the Differential and Integral Calculus*. To appear in the Proceedings of Colog 88, Journal of Symbolic Computation, 1989.
- [2] J. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra: Systems and Algorithms for Algebraic Computation*. Academic Press, London, 1988.
- [3] S. Dooley. *The Use of Domain Restrictions in Computer Algebra Systems*. Master's thesis, University of California, Berkeley, California, 14 December 1988.
- [4] R. J. Fateman. On the systematic construction of algebraic manipulation systems. Draft of 24 September 1987. Submitted to the Journal for Symbolic Computation.
- [5] D. Richardson. Some unsolvable problems involving elementary functions of a real variable. *Journal of Symbolic Logic*, 33:511-520, 1968.
- [6] E. P. Sacks. *Automatic Qualitative Analysis of Ordinary Differential Equations Using Piecewise Linear Approximations*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, February 1988.
- [7] L. A. Steen (Ed.). Calculus for a new century: a pump, not a filter. *Notes of the Mathematical Association of America*, 8, 1988.
- [8] P. Suppes, T. A. Ager, P. Berg, R. Chuaqui, W. Graham, R. E. Maas, and S. Takahashi. *Applications of Computer Technology to Pre-College Calculus: First Annual Report*. Technical Report 310, Institute for Mathematical Studies in the Social Sciences, Stanford University, April 1987.

Bunny Numerics

A Number Theory Microworld

*Craig Graci
Jack Narayan
Randy Odendahl*

State University of New York, College at Oswego

Abstract. *A microworld designed for use in number theoretic investigations is described. This microworld, bunny numerics, is being used to complement the workhorse turtle geometry microworld in a Logo based problem solving course that we have recently initiated at SUNY Oswego. The microworld is defined, examples of its use are provided, suggestions for its use are offered, and a few notes on its implementation are made.*

Contents

1. Introduction
2. The Bunny World
3. Basic Bunny Talk
4. Standard Bunnies, Breeds, and Birthing Operators
5. Selected Examples of Programming with Bunnies
6. Bunny Sets and Number Set Operators
7. Nonstandard Bunnies
8. Uses of Bunny Numerics
9. Some Implementation Notes
10. Concluding Remarks

1. Introduction

The bunny numerics microworld was inspired largely by Seymour Papert's conception of how to create a curriculum, which is "to create a network of microworlds, each one focussing on different areas of knowledge."¹

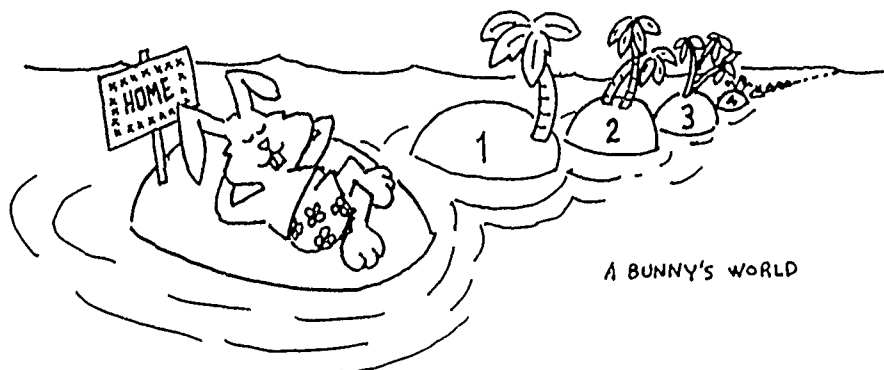
At SUNY Oswego we have recently introduced a two course sequence designed to satisfy a general education requirement in the area of mathematics and computation. The two courses, Elements of Problem Solving, Mathematics, and Computation, I and II, are grounded in Logo. They are intended to address mathematics in the broadest sense. That is, they aim to provide students with an understanding of the sorts of thought processes employed by mathematicians and computer scientists in their problem solving endeavors. In support of this course we have crafted a small number of microworlds which serve to complement the workhorse turtle geometry microworld.

This paper describes one of these microworlds, bunny numerics, which was designed for use in number theoretic investigations. Like the turtle geometry microworld, the bunny numerics microworld is embedded in Logo. However, since bunny numerics is an "add on," by contrast with turtle graphics which is inherent in Logo, the bunny numerics code must be explicitly loaded into the Logo system before it may be used.

Specifically, Sections 2, 3, 4, and 7 present the essential features of the bunny numerics microworld. Sections 5, 6, and 8 are intended to provide perspective. Section 9 contains brief remarks on the conceptual model underlying the bunny numerics microworld, and also on the use of Coral Software's ObjectLogo as the implementation language.

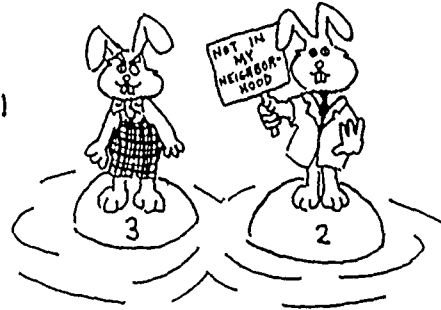
2. The Bunny World

The world of the bunnies may be thought of as an ocean dotted with a never ending "line" of islands. The islands are called home, 1, 2, 3, and so on.



There are various *breeds* of bunnies, corresponding, in the main, to kinds of numbers. For example, there are odd bunnies, even bunnies, Fibonacci bunnies, and prime bunnies. A given breed of bunny is generally limited in terms of the islands that it can visit. A prime bunny, for example, can land only on the prime islands, the islands numbered 2, 3, 5, 7, 11, etc., and also on the Home island. A bunny knows never to set foot on an island which is not suited to its kind. All bunnies are comfortable at Home, which is also the birth place of all bunnies.

ODD BUNNIES KNOW THEY'RE
NOT WANTED ON EVEN ISLANDS!

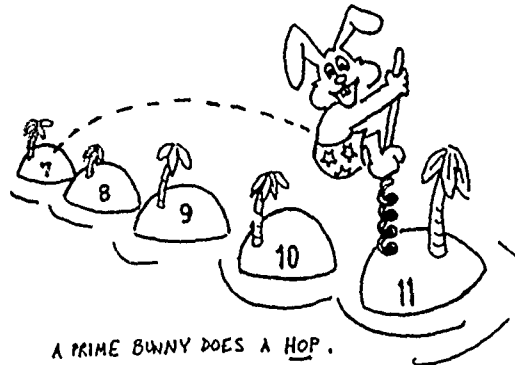


3. Basic Bunny Talk

Bunny talk is the set of Logo procedures one uses to communicate with bunnies. The most fundamental bunny talk procedures are: Hop, Location, Distance, HopAge, and HopHome. A brief description of each follows.

- Hop *bunny* (command)

The specified bunny moves to the next highest numbered island to which its kind is suited.



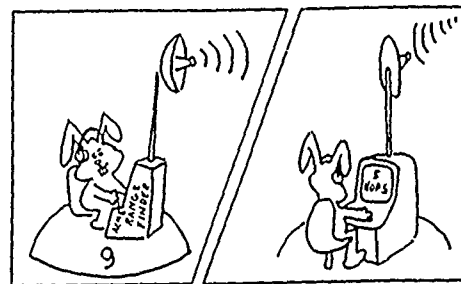
A PRIME BUNNY DOES A HOP.

- Location *bunny* (operator)
Loc *bunny*

The "name" of the island on which the specified bunny is presently resting is returned.

- Distance *bunny* (operator)
Dis *bunny*

The number of hops that the specified bunny is from Home is returned.



AN ODD BUNNY FINDS HIS DISTANCE FROM HOME.

- HopHome *bunny* (command)

The specified bunny hops Home.

- HopAge *bunny* (operator)
Age *bunny*

The number of hops that the specified bunny has taken since its birth is returned.

4. Standard Bunnies, Breeds, and Birthing Operators

Standard bunnies are bunnies who are born when bunny numerics is loaded. Initially, they are found lounging at home. Standard bunnies were included with young children in mind, and will generally be ignored by "grown ups." A good way to begin thinking about number theory is to simply generate some sequences of numbers, and look for patterns. The reader may wish to refer to Table 1 when reading the following examples. Note, particularly, the *variables* that are used to denote standard bunnies.

```
? ;;view some squares, assuming Sammy is at home
? REPEAT 10 [ Hop :Sammy Type ( Loc :Sammy ) Type "| | ]
1 4 9 16 25 36 49 64 81 100
```

```
? ;;view some multiples of 4, assuming Mark4 is at home
? REPEAT 10 [ Hop :Mark4 Type ( Loc :Mark4 ) Type "| | ]
4 8 12 16 20 24 28 32 36 40
```

A generalization of this idea is in order. The following procedure takes a bunny as input and displays the "names" of the first few islands on which it comes to rest.

```
TO Sequence :b :n
  HopHome ;note that this is not a "bunny invariant" procedure
  REPEAT :n [ Hop :b Type ( Loc :b ) Type "| | ]
END
```

```
? ;;display the first 10 Fibonacci numbers
? Sequence :Flo 10
1 1 2 3 5 8 13 21 34 55
```

```
? ;;display the first 10 prime numbers
? Sequence :Pierre 10
2 3 5 7 11 13 17 19 21 23
```

Standard breeds are breeds of bunnies that exists when the bunny numerics system is loaded. One can create virtually any number of bunnies of a particular bunny breed through application of appropriate *birthing operators*. A simple illustration employing two bunnies of like breed in a harmonious way is given by the following procedure for displaying pairs of *twin primes*,

i.e., prime numbers which differ from one another by two. This procedure also typifies the representational independence characteristic of many solutions to number theory problems expressed in bunny talk.

```

TO DisplayTwinPrimes
  Make "b1 PrimeBunny
  Make "b2 PrimeBunny
  Hop :b2
  FOREVER
  - [ Hop :b1 Hop :b2
  -   IF ( ((Loc :b2) - (Loc :b1)) = 2 ) [ Pr List ( Loc :b1 ) ( Loc :b2 ) ]
  - ]
END

? DisplayTwinPrimes
3    5
5    7
11   13
17   19
29   31
...

```

The table below identifies a *sampling* of the standard bunnies, breeds, and birthing operators.

Standard Breeds	Birthing Operators	Standard Bunnies
Even Bunny	EvenBunny	Ed
Square Bunny	SquareBunny	Sammy
Factorial Bunny	FactorialBunny	Fred
Fibonacci Bunny	FibonacciBunny	Flo
Multiple of i Bunny	MultipleBunny < i >	Mark1, Mark2, .., Mark12
Divisors of n Bunny	DivisorBunny < n >	Di1, Di2, .., Di20
Prime Bunny	PrimeBunny	Pierre
Perfect Bunny	PerfectBunny	Pearl

Table 1: Some Standard Bunnies, Breeds, and Birthing Operators

The standard breeds were rather arbitrarily chosen, and are merely a small fraction of the interesting bunny breeds. For a complete listing of the standard bunny numerics entities see the *Bunny Numerics Report* [2]. The definition of nonstandard breeds is discussed in Section 7.

5. Selected Examples of Programming with Bunnies

5.1 Generating Simple Lists of Numbers

The following procedure simply displays a specified number of factorials.

```

TO DisplayFactorials :n
  Make "FB FactorialBunny
  REPEAT :n [ Hop :FB Print Location :FB ]
END

```

```
? DisplayFactorials 7
```

```
1
2
6
24
120
720
5040
```

Similarly, one could write a procedure to print out the first n primes, cubes, etc. Of course we could have called upon Sequence to list the Factorials, but as they quickly become very large, the placement of each on a separate line seemed appropriate. Several students, upon seeing the bunnies in action, have asked about how various sequences of numbers are generated. This is the sort of interest that we had hoped bunny numerics would generate! We earnestly encourage interested students to investigate the generation of various number sequences in terms of the more primitive Logo procedures.

Displaying the multiples of a given number may be accomplished with the following procedure:

```
TO DisplayMultiples :m
  Make "MB MultipleBunny :m
  FOREVER [ Hop :MB Print ( Loc :MB ) ]
END
```

The multiple bunny breed is partitioned into subbreeds. The parameter provided to the birthing operator is used to select the particular subbreed from which the bunny is born.

5.2 Searching for Numbers with More than One Property

The example below presents a very primitive solution to computing the least common multiple of two integers. Such illustrations can help to make notions meaningful to beginners. The "leapfrogging" technique employed by the bunnies is a common idiom used in bunny talk programming.

```
TO LeastCommonMultiple :n1 :n2
  LocalMake "Jack ( MultipleBunny :n1 "Jack )
  LocalMake "Jill ( MultipleBunny :n2 "Jill )
  Hop :Jack PrintLoc :Jack
  Hop :Jill PrintLoc :Jill
  WHILE [ NOT ( ( Loc :Jack ) = ( Loc :Jill ) ) ]
  [
    IFELSE ( ( Loc :Jack ) < ( Loc :Jill ) )
    [ Hop :Jack PrintLoc :Jack ]
    [ Hop :Jill PrintLoc :Jill ]
  ]
  PrintLines 2
  ( Display "The LCM of " :n1 " and " :n2 " is: " ( Loc :Jack ) )
END
```

```

? LeastCommonMultiple 4 7
location of Jack: 4
location of Jill: 7
location of Jack: 8
location of Jill: 14
location of Jack: 12
location of Jack: 16
location of Jill: 21
location of Jack: 20
location of Jack: 24
location of Jill: 28
location of Jack: 28

```

The LCM of 4 and 7 is: 28

As may be surmised from this example, several IO utilities are included with the bunny numerics microworld, e.g., PrintLoc, Display, PrintLines, and TypeSpaces. Moreover, the use of an optional *name* parameter, which may be supplied to any birthing operator, is employed in the calls to the MultipleBunny birthing operator. This name is used by the PrintLoc command. (In ObjectLogo, the application of a procedure with some number of inputs other than the standard requires a LISP-like use of parentheses).

The following less prolix example uses the same leapfrogging technique to compute and display prime Fibonacci numbers.

```

TO DisplayFiboPrimes
  LocalMake "FB FibonacciBunny
  LocalMake "PB PrimeBunny
  Hop :FB Hop :PB
  FOREVER
  - [ IFELSE ( ( Loc :FB ) = ( Loc :PB ) )
    - [ Print ( Loc :FB ) hop :Fb hop :PB ]
    - [ IFELSE ( ( Loc :FB ) < ( Loc :PB ) ) [ Hop :FB ] [ Hop :PB ] ]
  - ]
END

? DisplayFiboPrimes
2
3
5
13
89
233
1597
...

```

5.3 Divisors

The number sequences focussed on thus far have all been infinite. In contrast, the sequences of numbers corresponding to the subbreeds of divisor bunnies are among those which are, in a sense, finite. These "sequences" are somewhat artificial, but nonetheless turn out to be very useful. A "divisor 10" bunny, for example, can land on the islands 1, 2, 5, and 10, in addition to Home. Recall the procedure Sequence:


```
? Sequence ( DivisorBunny 10 ) 8
1 2 5 10 home 1 2 5
```

The following procedure will neatly display the divisors of a given number.

```
TO DisplayDivisors :n
  LocalMake "Diva DivisorBunny :n
  REPEAT ( LongestTrip :Diva ) [ Hop :Diva Type Location :Diva TypeSpace ]
  PrintLine
END

? DisplayDivisors 23
1 23
? DisplayDivisors 24
1 2 3 4 6 8 12 24
```

The LongestTrip operator is a part of bunny numerics. It computes the maximum distance from home that a bunny may find itself, and it may be applied to any bunny. The computation will terminate, however, only if the longest trip is finite.

The procedure below displays a table of divisors for the first n natural numbers.

```
TO DisplayTableOfDivisors :n
  FOR [ 1 1 :n ]
    [ Type :i TypeSpaces ( 6 - ( Count :i ) ) DisplayDivisors :i ]
  END

? DisplayTableOfDivisors 6
1      1
2      1 2
3      1 3
4      1 2 4
5      1 5
6      1 2 3 6
```

Generating sequences of numbers and various tables for analysis is essential to finding patterns and making conjectures in the elementary theory of numbers. The bunnies can be extremely helpful in this regard.

6. Bunny Sets and Number Set Operators

There is provision in bunny numerics to create sets of numbers derived from number sequences. The principle bunny set constructor is *BunnySet*, which takes two inputs, a bunny, say b, and an integer, call it n. This operator returns a set corresponding to the first n elements of the sequence generated by b. There are also a variety of number set procedures included with the bunny numerics system for use with bunny sets. To illustrate:

```
? PrintSet BunnySet OddBunny 15
{ 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 }
```

```
? PrintSet BunnySet FibonacciBunny 7
{ 1 2 3 5 8 13 }
```

Notice the absence of two occurrences of "1" in the Fibonacci set. Note also that order is not significant within the braces.

This feature of bunny numerics can be used to describe many ideas cleanly. The procedure below, for example, generates primes using Eratosthenes' sieve method.

```
TO Sieve :n
  LocalMake "Numbers Diff ( BunnySet NaturalBunny :n ) ( Set 1 )
  LocalMake "Limit ( Sqrt :n )
  FOR [ i 2 :Limit ]
  [
    IF ( ElementOf :i :Numbers )
    [
      LocalMake
      "SpecialSet Diff ( BunnySet ( MultipleBunny :i ) :n ) ( Set :i )
      Make
      "Numbers ( Diff :Numbers :Specialset )
    ]
  ]
  OUTPUT ( :Numbers )
END

? PrintSet Sieve 50
{ 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 }
```

The operators Set, Diff, ElementOf, and the command PrintSet are all part of the aforementioned number set procedures included with bunny numerics.

7. Nonstandard Bunnies

The procedure *NewBunnyBreed* is used to create a new breed of bunnies. For the general form of this operator, and details of its use, the *Bunny Numerics Report* [2] may be consulted. Below are two examples of the use of the *NewBunnyBreed* operator in defining new bunny breeds.

A breed of geometric progression bunnies could be defined as follows:

```
NewBunnyBreed "GeoBunny [ Base Mult ]
[ Make "CI :Base ]
[ Make "CI ( :CI * :Mult ) ]
```

CI stands for "Current Island." Recalling, again, Sequence from Section 4:

```
? Sequence ( GeoBunny 2 3 ) 5
2 6 18 54 162
? Sequence ( GeoBunny 5 9 ) 3
5 45 405
```

As a second example, a breed of *wonder* bunnies may be defined in order to investigate the "wondrousness" number property discussed by Achilles and the Tortoise in Douglas Hofstadter's *Aria with Diverse Variations*.²

```
NewBunnyBreed "WonderBunny [ 1 ]
  [ Make "CI :1 ]
  [ IFELSE ( Odd :1 ) [ Make "CI ( :CI * 3 ) + 1 ] [ make "CI ( :CI / 2 ) ] ]
```

The following procedure *might* then be used to verify that a given number is, indeed, wondrous.

```
TO IsWondrous :i
  Make "WB WonderBunny :i
  FOREVER [ Hop :WB IF ( ( Loc :WB ) = 1 ) [ OP "True ] ]
END
```

8. Uses of Bunny Numerics

One can use bunny numerics in the ways alluded to thus far: to generate number sequences; to find numbers with particular properties; to test conjectures. The student should typically, perhaps with some direction from the teacher, read some of the history and lore of number theory, identify interesting questions, and explore these questions with the aid of the bunny numerics microworld.

Beyond this, one might exploit the bunny numerics microworld in a number of ways. For example, a tried and true induction game based on guessing the next number in a sequence can be nicely automated in the context of bunny numerics. We have written a version called INDUCE which takes a bunny as input, generally a nonstandard bunny of our own design, and then interacts with the player offering the opportunity to guess the underlying rule. We employ a very simple acceptance procedure. If the player can correctly identify the next three numbers in the sequence, we credit the player with knowing the rule. Each time the person fails to guess the rule, the next number in the sequence is divulged. The distance from Home of the bunny at the time the rule is finally guessed is displayed at the end of a game. INDUCE is quite like WHEEL OF FORTUNE, only a bit more interesting from the perspective of a mathematician - with one notable exception, perhaps.

The generation and solution of cross number puzzles are activities enhanced by the bunny numerics microworld. An interesting Artificial Intelligence project within the context of Logo, employing both the turtle graphics and the bunny numerics capabilities, would be to completely automate the generation of cross number puzzles. Good puzzles must have a certain "degree of interest" which is sufficiently difficult to describe as to, indeed, render their automatic generation a project within the domain of Artificial Intelligence. Regardless of how they are created, cross number puzzles

present very nice opportunities to apply strategies of *constrained search*, an important aspect of problem solving.

9. Some Implementation Notes

Our implementation of the bunny numerics microworld took very little time, largely because of the nature of the language that we used, namely Coral Software's ObjectLogo.

We exploited the object oriented features of ObjectLogo in modelling the bunnies. All bunnies have certain commonalities, e.g., they can all hop, determine their location, determine their distance from home, determine their "hopage," and find their way home. Thus a generic Bunny class was established as a direct subclass of the Logo class. Due to the fact that different bunny breeds hop in dramatically different ways, each breed requires its own refinement of the Hop procedure. Also, each bunny requires its own set of state variables, and thus its own birthing operator. The natural thing to do was to make each breed a subclass of the generic bunny class. This was all a straightforward exercise in object oriented programming. What was less straightforward was establishing nonstandard breeds as subclasses of the generic bunny class, "under the table" so to speak.

We exploited ObjectLogo's very direct kinship with LISP in order to achieve the undertaking just mentioned. Basically, the NewBunnyBreed procedure programmably generates ObjectLogo programs required to establish the new bunny breed classes as subclasses of the generic bunny class.

We also exploited ObjectLogo's inherent ability to operate on large integers. Multiline (hundreds of digit) factorials and perfect numbers, for example, are readily computed through bunny numerics, in a manner consistent with the computation of small factorials and perfect numbers.

10. Concluding Remarks

We are only now using the bunny numerics microworld in the first of our two new course offerings at SUNY Oswego. We hope soon to report on its successes and failures.

Citations

1. Papert, S. "MICROWORLDS: Transforming Education," in *Artificial Intelligence and Education, Volume One*, edited by R. Lawler and M. Yazdani, Ablex Publishing Co., 1987, page 60.
2. Hofstadter, D. *Godel, Escher, Bach*, Vintage Books, 1980, pp. 400 and 401.

References

- [1] Coral Software: *ObjectLogo Reference Manual*. Coral Software Corp., 1986.
- [2] C. Graci: *Bunny Numerics Report*. SUNY Oswego Department of Computer Science, 1989.
- [3] D. Hofstadter: *Godel, Escher, Bach*. Vintage Books, 1980.
- [4] R. Lawler and M. Yazdani (ed): *Artificial Intelligence and Education, Volume One*. Ablex Publishing Co, 1987.
- [5] S. Papert: *Mindstorms: Children, Computers and Powerful Ideas*. Basic Books, 1980.

Advanced Mathematics from an Elementary Viewpoint: Chaos, Fractal Geometry, and Nonlinear Systems

Wallace Feurzeig

Paul Horwitz

Albert Boulanger

BBN Laboratories, Cambridge, MA

Abstract. *We are conducting exploratory research to investigate the instructional issues and educational benefits from introducing both a new paradigm and a new area of applied mathematics into the high school curriculum. The new paradigm is experimental mathematics and the new area is mathematical chaos. By experimental mathematics we mean computer modeling of mathematical processes to gain insight into their structure and behavior so as to inform and guide mathematical inquiry. Mathematical chaos is the study of orderly and chaotic behavior in nonlinear processes and in the real world systems modelled by them. Both depend fundamentally on the use of computers and interactive graphics technology.*

School curricula often present the standard subjects in an intellectually impoverished and un compelling way, teaching modes of thinking and doing that are distinctly different from those used by practitioners. School math is not a model of real mathematics and school science is not genuine science. Education should be directed to grounding knowledge in experience and in contexts of use. Our thesis is that the introduction of experimental mathematics and mathematical chaos will help accomplish this by creating highly motivating computational environments that foster exploration and discovery and bridge the gulf between schoolwork and real mathematics and science.

1. Mathematical Chaos

Natural processes are inherently nonlinear. Nonlinear systems can give rise to very complex behaviors. Regular, stable, and predictable behavior can suddenly become highly irregular and unpredictable (a familiar example is turbulent flow). Until very recently we had no effective methods for analyzing, or even for observing and studying, the deep structure underlying these complex behaviors. This situation has changed dramatically with the emergence of a new area of applied mathematics called mathematical chaos, developed to study the nature of orderly and chaotic behavior in mathematical processes and in the real world systems modelled by them.

Mathematical chaos is founded on a set of remarkable discoveries: 1) that nonlinear processes can give rise to very complex unpredictable behaviors in a rich variety of systems — physical, chemical, and biological, 2) that these chaotic behaviors are

nevertheless deterministic and can be modeled by simple mathematical equations (those with few variables or with a small number of degrees of freedom), 3) that the processes by which systems approach chaos are themselves orderly, and 4) that the underlying deep structure of chaotic behavior is very similar across diverse domains and systems, perhaps even universal.

The study of this new mathematics depends heavily upon computationally intensive graphic methods. Its key findings, despite their dramatic and universal features, were all but impossible to discover without computers. The science of chaos is a child of the 20th century and could only have emerged as a subject for investigation and study in the computer era. Work on chaos is contributing to a deeper understanding of fundamental issues in mathematics such as the nature of algorithmic complexity, the difference between nondeterminism and randomness, and the deep structure of self-similar processes. During the past fifteen years chaos has had diverse and extensive applications throughout the sciences. This work is producing breakthroughs in our understanding of complex nonlinear systems of so profound a character as to constitute an intellectual revolution, a genuine paradigm shift in scientific thinking.

A major theme running through the history of 20th Century thought has been the gradual but steady erosion of certainty. This has taken place, by and large, along two separate, but parallel dimensions: the mathematical and the physical. In mathematics this century has seen the discovery that axiomatic set theory is inherently inconsistent, followed shortly by the proof that any such system must contain undecidable propositions, and eventually by the identification of particular examples. Through physics we have learned that the world is in principle unmeasurable and unpredictable beyond certain limits.

But these discoveries, fundamental and surprising though they were, had little practical meaning for most people. They applied only under rather esoteric conditions and their relevance for the everyday world was minor. However, for the past fifteen years or so — largely due to the advent of the computer, and building on pioneering work from the first decade of the century — we have begun to realize that even the "simple" mathematics and physics that we thought we understood has surprises in store for us. Simple, rational functions, when iterated many times, turn out to have unexpected properties; similarly, simple physical systems, in certain regimes, display wild and erratic behavior that may cause arbitrarily similar initial states to diverge exponentially, making it effectively impossible to predict the future behavior of the system. Thus the discovery of mathematical chaos lies at the confluence of the two great strands of uncertainty, mathematical and physical, and its consequences, though dimly perceived at present, are certain to be profound.

2. Experimental Mathematics

Computers have yet to make their most important intellectual contributions to the disciplines. Their enormous potential for supporting the creative and productive aspects of scientific, artistic, and scholarly thought and work is only beginning to be apparent.

Computers are already transforming the way science is done. Computer modeling is becoming a standard tool for experiment and theory. It is being used to study extremely complex processes, ranging in scale from the inner structure of the proton to star cluster formation and decay. It can be an illuminating source of creative insights about the structure and behavior of complex phenomena that were previously inaccessible, and it has made possible the solution of problems previously considered unsolvable. The use of computer-based models can be expected to provide dramatic breakthroughs in mathematics, physics, chemistry, genetics, economics, meteorology, pharmacology, demography, and other fields.

Computers will become increasingly important to mathematicians as basic research tools. This is already beginning to happen in the seminal area of mathematical chaos and nonlinear dynamics. Some of this work has great theoretical importance as well as rich applications. Chaos, far from being an isolated phenomenon, appears to be ubiquitous both in mathematics and nature. For example, iterated sequences of the basic elementary functions — sine, cosine, exponential, quadratic — exhibit chaotic behavior in the complex plane. This is a fairly recent and surprising revelation whose implications have yet to be fully investigated and understood.

Progress in this burgeoning research area would not have been possible without the use of powerful computational and graphic display facilities as essential investigative tools. Yet, the computational methods required to explore mathematical chaos and nonlinear systems are now accessible to high school students.

3. Implications for the High School Curriculum

Imagine U.S. history ending with the McKinley Administration, an English curriculum that makes no mention of Joyce or Faulkner, a course in physics that knows nothing of atoms and nuclei. The mathematics curriculum is unique among major secondary school subjects in that it contains absolutely no content that was developed or discovered in the twentieth century. Indeed, most of what is included in the conventional mathematics curriculum might well have been taught to George Washington.

There are reasons for this, of course. Most of the mathematics developed since the Renaissance has little relevance to the computational tasks that confront us all in everyday life. Moreover, much of it is deemed too abstract and too difficult for all but the ablest students. Yet it is evident that many very deep mathematical ideas — the concept of infinity, for instance — hold real interest, particularly for younger students. It is also undeniable that this interest is dissipated and lost, for the vast majority of students, probably as a byproduct of their exposure to the "real" mathematics of long division and the binomial theorem.

We do not intend, in this paper, to argue for either side of the long-standing, and at times quite bitter, debates on issues such as whether the availability of computers and calculators has made such topics as long division "obsolete". Rather, we wish to describe the remarkable symbiosis that we perceive between the direction of certain contemporary mathematical research and the needs of mathematics education, and to point out ways in which the computer can help to fill the gap between the two. Specifically, we believe that

recent research on chaos offers an unprecedented opportunity for students not only to learn some extremely important mathematics of very recent vintage, but in the process to experience the excitement and pleasure of mathematical inquiry and discovery.

The purpose of our project is to explore the introduction of the frontier research area of chaos into the high school curriculum. On the face of it, this may seem unrealistic. Advanced research typically implies sophisticated concepts and technically difficult methods well above the level of school mathematics. In the case of chaos, however, despite its modernity and its applicability to real-world situations, an introductory presentation requires little mathematics beyond high school algebra. Moreover, the visual displays that are the standard mode of representation of chaotic processes greatly facilitate understanding. There is a deep connection between chaos and fractal geometry. The graphic pictures that are generated as natural outputs of investigations are often breathtakingly beautiful objects in their own right — the connection between mathematics and visual art has never been so apparent.

We believe that a nontrivial introduction to the ideas and methods of chaos can be developed and presented in a way that is both accessible and compelling to a significant fraction of high school students. This material is ideally suited to give students authentic experience of what doing mathematics and science is really like in areas that are meaningful and truly interesting to them. It provides rich opportunities for successful mathematical exploration, inquiry, and discovery. We plan to generate projects in relatively uncharted areas where it is possible for students to make new findings. In introducing students to the concepts and techniques of mathematical chaos we are placing them in a position to conduct investigations in a manner quite analogous to that employed by professional mathematicians. And it is almost inevitable that they will, in fact, discover new things, for the surface has only been scratched in this field, and most of the territory remains unexplored.

We plan to investigate a rich variety of topics from the mathematics of chaos, fractals, and nonlinear systems, including applications of many kinds. We seek to develop a coherent conceptual framework for introducing the key ideas at a level appropriate for high school presentation. To this end we are creating software tools designed to aid students in carrying out mathematical experiments and explorations. These tools will enable students to build and run models of dynamical systems with complex behaviors, to see their effects unfold, and to manipulate and study the generated graphic structures in multiple representations and at all levels of detail. We have started to design learning activities centered on the use of the tools and designed to develop organically the knowledge needed to use them powerfully.

4. Topic Areas

Our main research activity is the exploration of the ideas and methods of chaos and nonlinear dynamics with a view toward developing a conceptual framework and exemplary materials in key and representative topics suitable for introduction into high school mathematics courses. The materials that will be developed are new but they grow

naturally out of traditional high school content. The sample topics described here illustrate the kinds of possibilities we envision and the approach and flavor of our presentations.

We introduce the subject of mathematical chaos to students by first familiarizing them with three fundamental concepts: iterated functions, maps, and fractals. Students then explore a wide variety of applications of chaos, e.g. to purely mathematical problems such as finding the roots of an equation; to the modeling of non-linear systems, such as the growth and decline of animal populations, the spread of infectious disease, and the beating of the human heart; and to the creation of fractal art and music.

The phenomenon of chaos is intimately linked to the behavior of functions, often very simple ones, when iterated many times. Only one of three things can happen: successive iterates of the function may approach a single fixed point; they may converge to a limiting orbit of points; or they may behave more erratically, never quite returning to a value they have taken on before. In the last case the iterated function sometimes displays an extremely sensitive dependence on initial conditions, so that neighboring starting points, when operated on repeatedly by the function, diverge very rapidly from one another, and all information about the starting point is lost. Behavior characterized by such an extreme sensitivity to initial conditions has been termed chaotic. The successive values taken on by the function closely resemble a random sequence, and indeed chaotic functions can be used as pseudorandom number generators. Because of their sensitive dependence on initial state, mappings of chaotic functions often display nearly self-similar structure on an infinitesimal scale, giving rise to curves and surfaces of fractional dimension, or fractals.

5. Mapping

A logical starting point is the concept of a map. To aid students in visualization as well as computation, we are building a multipurpose mapping tool (known as "MultiMap") that allows any figure drawn in one window on the Macintosh screen to be mapped into another according to whatever rule the user chooses. This mapping will be done very rapidly, in such a way that drawings made either free hand or with the aid of the computer in one window will simultaneously be mapped into any window or windows linked to the first.

We generalize the familiar notion of geographic maps by introducing an activity that combines two maps into one. For example, we can set up two windows, one of which contains a portrait of John F. Kennedy, the other a portrait of Marilyn Monroe. We can then map each of these onto a third window, in which we create a composite "map" consisting of a weighed average of the two input portraits (which can be accomplished quite easily by identifying key features — eyes, nose, etc. — on each of the two inputs). By moving a "scroll bar" or through some other suitable interface, students can modify the artificial portrait to look more or less like either of the two originals.

Using an image digitizer, we can input photographic portraits of individual students and transform them in interesting ways with MultiMap. For example, we can stretch their length twofold while halving their width, simulating the operation of a rolling pin on dough.

We can cut the resulting map in half and stack the right half over the left half, so as to restore the original dimensions. We can then iterate this process a number of times and observe how the original likeness becomes unrecognizable and apparently unrecoverable, an ostensibly random pattern. But this iterated map is reversible, and the original likeness can be restored by iterating the inverse map the same number of times — the effect seems magical.

6. Newton's Method

Iterated maps are useful in more traditional mathematical activities, such as finding the roots of equations. Newton's method is a well-known iterative procedure for locating the roots of equations in the complex plane. It can serve as an alternative to the quadratic equation formula routinely taught in high school algebra. The method has the dual advantage that it can be generalized to cubic and higher-order polynomial equations, and that it can be motivated and justified to students via an appropriate graphical representation.

We introduce Newton's method in the context of quadratic equations, with which students are already familiar, presenting it initially merely as an alternative to the use of the usual, somewhat mysterious, formula. The students start by making an initial guess. The method then involves the repeated application of an algorithm which ultimately converges on one or the other of the two roots.

We then pose the question: how does the choice of the initial guess determine the future behavior of the process? In particular, which of the two roots does the process ultimately converge on, and which initial guesses, if any, will result in its never finding a root? In order to answer this question, students use MultiMap and other graphic software tools to determine by trial and error the regions in the complex plane for which starting guesses converge to one or another of the roots of the equation. This is a legitimate research problem, first solved by Cayley (Peitgen, 1982).

For quadratic equations the solution is not surprising: connect the two roots by a straight line segment and construct the perpendicular bisector of this segment. Then the "basin of attraction" of each root (that is, the set of all initial points for which the method converges to that root) is simply the open half plane on one or the other side of the perpendicular bisector. Points on the bisector itself do not converge to either root and in fact their behavior is chaotic, in the sense that the behavior under iteration of neighboring points diverges very rapidly, so that all information relating to the initial point is lost. In modern terminology, the perpendicular bisector comprises the so-called Julia set of the iterated rational function that characterizes Newton's method.

This new kind of exploration, in which one asks about the behavior of an iterated function at each point in the complex plane, requires a new kind of software tool — one capable of producing a variety of new kinds of mappings. The most obvious mapping simply assigns a different color to each pixel on the screen depending on the behavior of the iterated function at the corresponding point on the complex plane. Thus, a natural map of the situation described above is to color all points in the basin of attraction of one

of the roots of the quadratic equation red, say, and of the other, green. This procedure divides the plane into two equal regions, separated by a straight line.

The development to this point has been straightforward and appears to be general. In fact, in Cayley's original paper he expresses the hope that he will soon be able to solve the equivalent problems for cubic and higher order equations. He never did so and the reason becomes, with the advent of today's computing power, graphically obvious. We show students how to generalize Newton's method from quadratic to cubic equations, and give them the task of mapping out the basins of attraction of each of the three (complex) roots. The resulting map is a quite unexpected and extremely complicated fractal picture. The reason for this is simply stated, though surprising. It can be rigorously shown that in the neighborhood of the Julia set (that set for which the function "cannot make up its mind" which of the three roots to converge to) there must be points belonging to each of the three basins of attraction. In geometric terms, then, (coloring the roots, say, red, green, and blue) at any point where two regions (say red and green) come together, the other (blue) region must meet both of them, as well. After some consideration of this startling statement the reader may well come to the conclusion that this situation is impossible. It is not, but it does require the introduction of fractals.

7. Fractals

MultiMap supports recursive maps. For example, it can map window A onto window B and then map window B back onto window A. This makes it a valuable tool for the study of iterated functions. For example, students can use MultiMap to construct pictures that contain "infinitely many" reduced copies of themselves. Such pictures can be constructed simply by creating a reduced scale mapping from one window to another, and then mapping the second window back onto the first, appropriately positioned. The iteration of these "condensation maps" often results in the creation of pictures that mimic such naturally occurring objects as ferns and clouds (Barnsley, 1986). In addition to being inherently interesting to students, these pictures illustrate the important idea of invariance under a scale transformation — an idea that underlies the concept of a fractal.

"Fractal" is the name coined by Benoît Mandelbrot to designate the convoluted curves and surfaces that exhibit approximate self-similarity at arbitrary scales (Mandelbrot, 1983). In a sense that Hausdorff and others have made explicit, such structures can be thought of as having non-integral dimensions. They are quite amazingly complex and often very beautiful. By virtue of its ability to generate recursive maps, MultiMap becomes a kind of "Fractal Construction Set" that enables students to create, modify and investigate fractals as objects of interest in their own right, even before they discover their deep connection with the phenomenon of chaos.

8. Modeling Nonlinear Processes

The properties of simple functions iterated many times are wonderful, unexpected and beautiful, but they may be expected to fall outside the set of inherently interesting topics for most high school students. To someone for whom the solving of equations — even

beautiful ones — is not particularly motivating, the fact that this task can be accomplished through the iterating of a simple function is unlikely to be of lasting interest. It is important, therefore, as we explore the spectrum of possible topics and activities, to move on to situations in which the iteration of a function implies something more than merely finding the roots of an equation. An obvious choice, and one that has rich mathematical and scientific applications, is to model a variety of processes that evolve in time. Each successive iteration of the function may be taken to represent a fixed time interval. If this interval is long enough to produce significant changes in the variables the resulting equation is a finite difference equation; if it is short on this scale, we consider it as approximating a differential equation. Though the mathematics of these two cases is in some important respects quite different, many of the techniques employed are the same, and we introduce both to students. We start by describing an example from ecology.

Consider a hypothetical population of, say, rabbits that has the property that every year on the average the number of rabbits becomes R times what it was the year before. Taking the population to be P , we model the time dependence by the function $f(P)=RP$. For $R<1$, repeated iteration of this function leads to an inexorable decay of the population toward 0 — all the rabbits die out. $R=1$ models a static, unchanging population, while $R>1$ leads to a Malthusian exponential growth in the rabbit population — a situation that would be disastrous were it not totally unrealistic. Obviously, many factors in the environment militate against such unbridled growth, scarcity of natural resources and the presence of predator species among them. To take such limiting factors into account, we modify our model to the form $f(P)=RP(1 - P)$, where we have normalized the population to some arbitrary "carrying capacity" and are therefore interested only in values of P between 0 and 1. Inasmuch as this change obviously restricts the rate of growth for values of P near 1, we might expect that for any particular value of R the population will grow to reach a corresponding limiting value, remaining in equilibrium at this value forever. This expectation is indeed borne out for small values of R .

But as we increase R some very surprising things happen to our model rabbit population. As R passes through the value 3.0, the population starts to oscillate between two different limit points, taking on each of them at alternate generations. (This non-intuitive result is, in fact, a good approximation to the behavior of some real animal populations.) This "bifurcation phenomenon" repeats itself. As R continues to grow, the number of limit points becomes 4. Shortly thereafter the solution bifurcates again and we have 8 limit points, then 16 and so forth. If we take the values of R at which these successive bifurcations happen, we find that they asymptotically converge to a geometrical series which itself has a definite limit point. At this limit point the behavior of our model rabbit population becomes "chaotic": the population no longer settles down to any particular limit point no matter how long we look. Furthermore, in this regime, if we start from two infinitesimally close initial populations the future behavior of these populations will diverge in finite time, making accurate prediction impossible in the absence of total information about initial conditions.

If we continue to increase R , we find additional peculiarities. Regions of all periodicities appear, interspersed with additional regions of chaotic behavior. The most

remarkable fact is that all of this complexity — bifurcations, chaos, various kinds of periodicity — is universal, in the sense that the exact nature of the underlying equation is not critical. This universality, first discovered by Feigenbaum approximately 15 years ago (Feigenbaum, 1980), is at the heart of the regularities in chaos that have characterized our growing knowledge of the whole field of nonlinear dynamics.

Without the computer it would be unrealistic to attempt to introduce differential equations to the high school mathematics curriculum. However, once one has made a connection in students' minds between iterating a function and modeling a time-evolving process it makes very little difference whether the time interval represented by the function is long or short. The major difference is that changes in the system being modeled become continuous, rather than discrete. If we visualize the state of the system as a localized dot on a graph, we will observe that dot to move continuously, rather than hopping from point to point. Moreover, the system need not be restricted to one dimension. If we have two populations, say rabbits and foxes, we can graph one along x and the other along y .

By generalizing our description of a system in this way, we introduce the notion of *phase space* as the collection of variables that are necessary to describe the system. This concept is a very powerful one. It can be applied to model the behavior of a great variety of nonlinear systems. For example, the spread of infectious disease has been modeled in (Schulman, 1986) using a simple chaotic equation (which has also been applied to galaxy formation!). The fractal growth of snowflakes is explored in (Nittmann, 1986). Aspects of the economics and politics of the arms race can be probed using the same equation we introduced to describe rabbits (Saperstein, 1984). Also, Skarda et al (Skarda, 1987) use chaos to describe how rabbits smell things.

With the advent of machine-based laboratories — inexpensive probes connected to computer programs for carrying out real-time data collection and processing tasks — many of the most interesting chaotic systems can now be studied directly in the classroom. Using an MBL system, we are designing classroom projects that involve the collection of real data generated by nonlinear systems. Our initial focus is on the double pendulum, a dynamical system consisting of two pendulums, one suspended from the bottom of the other in a way that enables both to rotate independently through angles up to 360 degrees. We have built and instrumented this complex "trapeze" toy. It exhibits many modes in which the excursions made, particularly by the lower pendulum, are complex, surprising, and fascinating to watch. This relatively straightforward mechanical device provides an ideal laboratory tool for studying the structure of chaos in physical processes. The chaotic dynamics exhibited in this laboratory system mirrors that characteristic of more complex real-world systems. Without MBL it would be nearly impossible to collect all the data required, and to develop graphic and analytic characterizations of the behaviors. However with computers, we can analyze the motion of such a double pendulum in real time in the classroom. Experiments of this kind demonstrate the applicability to real-world situations of the ideas behind mathematical chaos, thus bridging the largely artificial gap that has grown up between mathematics and science.

Acknowledgments

This project is supported, in part, by the National Science Foundation. The opinions expressed here are those of the authors and not necessarily those of the Foundation. Apple Computer, Inc. has awarded the project a grant of two Macintosh II computer systems and a LaserWriter II NTX printer to support project research and software development.

Bibliography

Peitgen H.O., D. Saupe, & F. v. Haeseler, "Cayley's Problem and Julia Sets", Mathematical Intelligencer, Vol. 6, No. 2 1982, pp. 11-20.

Mandelbrot, Benoit, The Fractal Geometry of Nature, Freeman, N.Y., 1983.

Barnsley, Michael, "Making Chaotic Dynamical Systems to Order", in Chaotic Dynamics and Fractals, M.F. Barnsley & S.G. Demko, eds., Academic Press, N.Y., 1986, pp. 53-68.

Feigenbaum, Mitchell J., "Universal Behavior in Nonlinear Systems", Los Alamos, Science, Summer 1980, pp. 4-27.

Schulman, Lawrence S., & Philip E. Seiden, "Percolation and Galaxies", Science, Vol. 233, 25 July 1986, pp. 425-431.

Nittmann, Johann, & H. Eugene Stanley, "Tip Splitting without Interfacial Tension and Dendritic Growth Patterns Arising from Molecular Anisotropy", Nature, Vol. 321, 12 June 1986, pp. 663-668.

Saperstein, Alvin M. "Chaos - A Model for the Outbreak of War", Nature, Vol. 309, 24 May 1984, pp. 303-305.

Skarda, Chrystine A., & Walter J. Freeman, "How Brains Make Chaos in Order to Make Sense of the World", Behavioral & Brain Sciences, Vol. 10, No. 2 1987, pp. 161-195.

Iterated Function Systems and the Inverse Problem of Fractal Construction Using Moments

Edward R. Vrscay and Christopher J. Roehrig

Department of Applied Mathematics, Faculty of Mathematics
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Abstract

Let K be a compact metric space and $w_i:K \rightarrow K$, $1 \leq i \leq N$, be a set of contraction maps, with assigned probabilities p_i . This *contractive iterated function system* (IFS) possesses a unique and invariant attractor set A . Given a target set S , the *inverse problem* consists in finding an IFS $\{K, \mathbf{w}, \mathbf{p}\}$ whose attractor A approximates S as closely as possible. We examine a numerical method of approximating a (fractal) target set S by minimizing the distance between the moments of S and A . This amounts to a nonlinear optimization of the parameters defining the IFS. In this way, both the geometry and shading measure encoded in S may be *simultaneously* approximated in a quantified procedure.

1. Introduction

A set of contraction maps defined over a bounded subset of $\mathbb{R}^n$ or $\mathbb{C}^n$ possesses a unique, compact "fixed point" set A , the attractor, which is invariant with respect to these maps: the set A is a union of the (distorted) copies of itself generated by the maps. This idea was developed by Hutchinson [1] to discuss self-similar fractal sets, and then extended and generalized by M. Barnsley, co-workers, and others [2-9]. The result, an *iterated function system* (IFS), is a powerful tool for both the *construction* of fractal sets and the *reconstruction* or approximation of sets which exhibit fractal-like properties over some scales; for example, leaves, trees and clouds. We draw the reader's attention to Barnsley's recent textbook on IFS [7], as well as to two papers [4,8] which summarize the method nicely for a general readership. The essence of the IFS method, provided by the *Collage Theorem* [3,7], is to exploit any self-tiling properties of the set S under consideration. From a practical aspect, the representation of complex images in terms of a small number of IFS parameters can represent a huge degree of data compression [4].

Given an initial approximation of maps and associated probabilities whose attractor A is to approximate the target set S , we know of no deterministic method of varying these parameters to optimize the approximation, either in Hausdorff metric or in shading. In this report, we propose that a method in the spirit of the classical moment problem can provide such a quantitative algorithm for the approximation of sets. In this way, both the *geometric* as well as *shading* information encoded in the target set S could be used simultaneously. The shading, considered as a distribution, defines a sequence of power moments G_i . Empirically, the distribution as well

as its moments could be obtained by digitization of the image. An IFS attractor set A supports a balanced measure whose moments g_j can be computed recursively from the parameters defining the IFS. We thus seek to approximate the measure living on S by minimizing a "distance" between the two moment sequences $\{G_i\}$ and $\{g_j\}$. The optimization procedure searches the *parameter space* of affine maps and probabilities to minimize the distance in moment space. The results of some preliminary investigations of sets in $\mathbb{R}$ and $\mathbb{R}^2$ are presented.

2. Iterated Function Systems, Attractors and Invariant Measures

Here, we outline very briefly the major definitions and properties of IFS. In the discussion below, (K, d) denotes a compact metric space with metric d . In applications, K will be a bounded subset of $\mathbb{R}^n$ ($n=1,2$). Also we let $\mathcal{S}$ denote the set of all compact subsets of K . For a more detailed discussion of the topological concepts involved, we refer the reader to the book of Falconer [10].

The distance between a point $x \in K$ and a set $A \subset K$ will be denoted as

$$d(x, A) = \inf_{y \in A} d(x, y). \quad (2.1)$$

The Hausdorff metric between two sets $A, B \subset K$ is defined as

$$h(A, B) = \max \left[\sup_{x \in A} d(x, B), \sup_{y \in B} d(y, A) \right]. \quad (2.2)$$

Let the ϵ -ball of a set $A \subset K$ be defined as $A_\epsilon = \{x \in K : d(x, A) < \epsilon\}$. Then $h(A, B) < \epsilon$ implies that $B \subset A_\epsilon$ and $A \subset B_\epsilon$. The elements of the set $\mathcal{S}$ form a complete metric space with metric h .

Now, let $\mathbf{w} = \{w_1, w_2, \dots, w_N\}$ denote a set of N continuous contraction maps on K , i.e. $w_i: K \rightarrow K$ and

$$d(w_i(x), w_i(y)) \leq s_i d(x, y), \quad \forall x, y \in K, \quad i=1, 2, \dots, N, \quad (2.3)$$

where $0 \leq s_i < 1$. Associated with these maps is a set of probabilities $\mathbf{p} = \{p_1, p_2, \dots, p_N\}$, which will become relevant below. The system $\{K, \mathbf{w}, \mathbf{p}\}$ defines a *contractive IFS* on K . Now define the action of $\mathbf{w}$ on a set $S \subset \mathcal{S}$ as

$$\mathbf{w}(S) \equiv \bigcup_{i=1}^N w_i(S). \quad (2.4)$$

and the iteration sequence

$$\mathbf{w}^{n+1}(S) \equiv \mathbf{w}(\mathbf{w}^n(S)) \quad , \quad n=1, 2, \dots \quad (2.5)$$

(For example, if S is a singleton point, then $\mathbf{w}(S)$ is a set of N (not necessarily distinct) points, $\mathbf{w}^2(S)$ a set of N^2 points, etc..)

Theorem 2.1 [1,2]: There exists a unique, compact set $A \subset \mathcal{S}$, the *attractor* of the IFS $\{K, \mathbf{w}\}$ (independent of $\mathbf{p}$), such that $\mathbf{w}(A) = A$ and $\mathbf{w}^n(S) \rightarrow A$ as $n \rightarrow \infty$ in Hausdorff metric, for all $S \subset \mathcal{S}$.

Hutchinson [1] shows that $\mathbf{w}$ is a contraction mapping in the complete metric space $\mathcal{S}$, hence it possess a unique "fixed point" $A \subset \mathcal{S}$. Barnsley and Demko [2] gave a proof in terms of coding sequences. From Eq. (2.4), the property $\mathbf{w}(A) = A$ shows that A is expressible as a union, or *tiling*, of (distorted) copies of itself. Some examples are:

1. $K = [0,1]$, $w_1(x)=x/3$, $w_2(x)=x/3 + 2/3$: A is the ternary Cantor set.
2. $K = [0,1]^2 \subset \mathbb{R}^2$, $w_1(x,y)=(\frac{1}{2}x, \frac{1}{2}y)$, $w_2(x,y)=(\frac{1}{2}x+\frac{1}{2}, \frac{1}{2}y)$, $w_3(x,y)=(\frac{1}{2}x+\frac{1}{4}, \frac{1}{2}y+\frac{1}{4}\sqrt{3})$: A is the Sierpinski gasket shown in Fig. 1(a).
3. K as in 2, $w_1, \dots, w_4$ linear affine transformations of the form in Eq. (3.7) below, with the coefficients and probabilities listed in Table 1: A is the *spleenwort fern* attractor shown in Figure 1(b).

The probabilities p associated with the maps w_i will now be relevant in determining the invariant balanced measures supported on the attractor A . These measures are central to our moment method approach. We begin by introducing the following algorithm which is useful in generating pictures of A on a computer screen.

The "Chaos Game": Pick an $x_0 \in K$ and define the iteration sequence

$$x_{n+1} = w_{\sigma_n}(x_n), \quad n=0,1,2,\dots, \quad (2.10)$$

where the index σ_n is chosen randomly from the set of indices $\{1,2,\dots,n\}$, with the probability of choosing i being p_i . If we plot the sequence $\{x_n\}$ for $n > n' \gg 1$, (say $n' = 50$), on a computer screen, then a pictorial representation of the attractor A will appear. This follows from

Theorem 2.2: Almost every orbit $\{x_n\}$ is dense on A .

The sequence must be attracted to A by the contractivity of the w map. The density of orbits follows from a symbolic dynamics-type argument: Each point $y \in A$ possesses a (not necessarily unique) coding which will be matched to an arbitrarily long sequence of digits (implying an arbitrary closeness to y) with probability 1 by the i.i.d. sequence $\{\sigma_n\}$. The computer image, however, is more than a picture of the geometry of set A . The probability that each pixel $p(i,j)$ defining A is visited by the random walk affords an approximation to an invariant measure supported on A and defined by the IFS. This is a consequence of the ergodic nature of the dynamical system defined in Eq. (2.10), as we outline below.

Let $\mathcal{B}(K)$ denote the set of Borel subsets of K . The "chaos game" essentially defines a discrete time Markov process $\{K, w, p\}$ defined by

$$P(x,B) = \sum_{i=1}^N p_i \delta_{w_i(x)}(B), \quad (2.11)$$

where $P(x,B)$ is the probability of transfer from $x \in K$ to the Borel subset $B \subset \mathcal{B}(K)$ and $\delta_y(B) = 1$ if $y \in B$ and 0 if $y \notin B$. For a contractive IFS, there exists a unique probability measure μ such that

$$\mu(B) = \int_K P(x,B) d\mu(x) \quad (2.12)$$

for all Borel subsets B of K [2]. This measure μ is referred to as the p -balanced measure of the IFS $\{K, w, p\}$ and its support is the attractor A . Elton has recently shown [5] (for even the weaker condition of "average contractivity" of the w_i maps) that the Markov process is ergodic (not in the sense of indecomposability, but in the Birkhoff sense). Starting at any $x_0 \in K$, the usual time-averaged distribution of the first $n+1$ points in the trajectory of Eq. (2.10), i.e.

$$\nu_n = \frac{1}{n+1} \sum_{k=0}^n \delta_{x_k}, \quad (2.13)$$

where δ_x denotes a point mass measure at x , converges weakly to the balanced measure μ as

$n \rightarrow \infty$ This accounts for the "picture" of the balanced measure yielded by the chaos game: Let the set B be the characteristic function of the subset of the plane represented by a pixel $p(i,j)$. Then the probability of visiting $p(i,j)$ is proportional to $\mu(B)$.

A noteworthy property which is a direct consequence of Eq. (2.12) is *invariance* of the measure [2], i.e. for an integrable function $f:K \rightarrow K$, $f \in L^1(A,\mu)$,

$$\int_A f(x) d\mu(x) = \sum_{i=0}^N p_i \int_A (f \circ w_i)(x) d\mu(x). \quad (2.14)$$

3. Moments of the Balanced Invariant Measure

If $\{K, w, p\}$ is a contractive IFS with attractor A , then the moments of the associated p -balanced measure supported by A are given by the (Lebesgue) integrals

$$g_{i_1 i_2 \dots i_n} = \int_A x_1^{i_1} x_2^{i_2} \dots x_n^{i_n} d\mu. \quad (3.1)$$

For convenience, the measure is assumed to be normalized, i.e. $g_{00\dots 0} = \int_A d\mu = 1$. We shall be concerned primarily with IFS on the line and in the plane. In all cases, our attention is restricted to linear (affine) maps w_i . There is a twofold advantage to this choice: (i) the geometry associated with such maps is simple, and (ii) their form permits a recursive computation of the moments, as will be shown below.

3.1 Moments of IFS Attractors Embedded in $\mathbb{R}$

We consider a general IFS defined by the linear maps

$$w_i(x) = s_i x + a_i, \quad |s_i| < 1, \quad i=1,2,\dots,N, \quad (3.2)$$

with associated probabilities p_i . From Eq. (2.13), setting $f(x) = x^n$, we have

$$g_n = \int_A x^n d\mu(x) = \sum_{i=1}^N p_i \int_A (s_i x + a_i)^n d\mu(x). \quad (3.3)$$

Expanding the polynomial, collecting like powers in x and integrating, we obtain the following recursion relation

$$\left[1 - \sum_{i=1}^N p_i s_i^n\right] g_n = \sum_{j=1}^n \binom{n}{j} g_{n-j} \left(\sum_{i=1}^N p_i s_i^{n-j} a_i^j\right). \quad (3.4)$$

(This formula, also given in [2] and [3], is valid for complex maps w_i in Eq. (3.2), with $s_i, a_i \in \mathbb{C}$.) By the assumptions in our definition of a contractive IFS, the coefficient of g_n on the left cannot vanish. Thus, the moments may be computed explicitly and uniquely by this recursion formula, with the initial value $g_0 = 1$. Conversely, since the attractor A is bounded, the moment problem is determinate, and an infinite sequence of moments g_n , $n=0,1,2,\dots$ determines a unique probability measure [12]. The moments g_n in Eq. (3.4) may be regarded as functions of the IFS parameters s_k, a_k, p_k , $k=1,2,\dots,N$, i.e. $g_n = g_n(s, a, p)$. Eq. (3.4) can be differentiated an arbitrary number of times with respect to any of these parameters (or combinations thereof). We formalize this compactly as follows.

Given an $N > 0$, and the IFS on $K \subset \mathbb{R}$ in Eq. (3.2), define the associated parameter space of $3N$ -tuple vectors $\pi = (s, a, p)$, that is,

$$\pi = (\pi_1, \dots, \pi_{3N}) = (s_1, \dots, s_N, a_1, \dots, a_N, p_1, \dots, p_N). \quad (3.5)$$

The feasible parameter space, Π_1^N , (the subscript refers to the embedding space $\mathbb{R}$) will be defined as the open subset of $\mathbb{R}^{3N}$ determined by the conditions

$$|s_i| < 1, \quad 0 < p_i < 1, \quad \sum_{i=1}^N p_i = 1, \quad (3.6)$$

No conditions on the a_i are necessary: K may be rescaled to ensure that $w_i: K \subset K$.

Theorem 3.1: The moments $g_n: \Pi_1^N \rightarrow \mathbb{R}$ are smooth functions of the parameters π_i .

Proof: from the form of Eq. (3.4).

3.2 Moments of IFS Attractors in $\mathbb{R}^2$

We now consider general IFS as defined by the linear affine transformations

$$w_i \begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} a_{11}(i) & a_{21}(i) \\ a_{12}(i) & a_{22}(i) \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1(i) \\ b_2(i) \end{pmatrix}, \quad i=1,2,\dots,N, \quad (3.7)$$

with associated probabilities p_i . Eq. (3.7) will be written in the compact form

$$w_i(\mathbf{x}) = \mathbf{A}_i \mathbf{x} + \mathbf{b}_i, \quad (3.8)$$

The matrices $\mathbf{A}_i$ must be contractive. The feasible parameter space, Π_2^N , will be a suitably defined open subset of $\mathbb{R}^{7N}$. (The parameters π_i are also standardized, as in Eq. (3.5).)

Setting $f(\mathbf{x}) = x^m y^n$ in Eq. (2.5), we have the following relation for the moments of the attractor:

$$\begin{aligned} g_{mn} &= \int_A x^m y^n d\mu \\ &= \sum_{i=1}^N p_i \int_A [a_{11}(i)x + a_{12}(i)y + b_1(i)]^m [a_{21}(i)x + a_{22}(i)y + b_2(i)]^n d\mu. \end{aligned} \quad (3.9)$$

Expanding the polynomials and integrating yields a complicated recursion relation for the g_{mn} :

$$\begin{aligned} g_{mn} &= \sum_{k=1}^N \left[\sum_{i_1=0}^m \binom{m}{i_1} a_{11}(k)^{i_1} \sum_{i_2=0}^{m-i_1} \binom{m-i_1}{i_2} a_{12}(k)^{i_2} b_1(k)^{m-i_1-i_2} g_{i_1 i_2} \right] \times \\ &\quad \left[\sum_{j_1=0}^n \binom{n}{j_1} a_{21}(k)^{j_1} \sum_{j_2=0}^{n-j_1} \binom{n-j_1}{j_2} a_{22}(k)^{j_2} b_2(k)^{n-j_1-j_2} g_{j_1 j_2} \right]. \end{aligned} \quad (3.10)$$

Because of the cross terms occurring in the products of Eq. (3.9), we can not solve for each g_{mn} explicitly, but must proceed as follows. From $g_{00} = 1$, write down the two equations of (3.10) corresponding to $m=0, n=1$ and $m=1, n=0$, then solve simultaneously for the unknowns g_{01} and g_{10} . The procedure is then continued: solve a system of $M+1$ linear inhomogeneous equations in the unknowns $\{g_{0M}, g_{1,M-1}, \dots, g_{M0}\}$ from a knowledge of the previously computed g_{mn} , $m+n < M$. The derivatives of the moments with respect to the parameters may also be computed in a recursive, albeit complicated manner. Smoothness of the moments also follows from Eq. (3.10), although some extra algebra is required to see that the matrices defining the relevant linear systems for the derivatives are nonsingular.

4. The Inverse Problem of Fractal Construction

4.1 Self-Tiling and the Collage Theorem

Up to this point, we have been viewing the IFS theory from only one direction, i.e. for a given contractive IFS $\{K, \mathbf{w}, \mathbf{p}\}$, there exists a unique attractor A with a $\mathbf{p}$ -balanced measure. We now consider the important *inverse problem*: given a set S , does there exist an IFS for which S is an attractor, or can S be approximated by an attractor A of an IFS to some arbitrary accuracy? The *Collage Theorem* gives insight to this problem.

Collage Theorem [3,7]: Let $S \subset \mathbb{S}$ and suppose that there exists a set of maps w_i so that Eq. (2.3) holds, and

$$h\left(S, \bigcup_{i=1}^N w_i(S)\right) < \epsilon. \quad (4.1)$$

Then,

$$h(S, A) < \frac{\epsilon}{1-s}, \quad (4.2)$$

where $s \equiv \max_i \{s_i\} < 1$, and A is the attractor of the IFS $\{K, \mathbf{w}\}$.

In other words, if a set S can be tiled with copies of itself to an arbitrary accuracy, then S is close to the attractor A of the IFS which produces the tiling. A natural procedure to follow, then, is to examine S for any self-similarities, determine the mappings w_i which effect these transformations, also ensuring that an appropriate number of maps are being employed. In this way, one attempts to approximate the geometric structure of S as best one can, i.e. minimize the Hausdorff distance $h(S, A)$. By varying the probabilities p_i associated with the maps, the balanced measure, i.e. the *shading* on S , may then be varied. The method has been very effectively developed by M. Barnsley and coworkers. It has been described in fair detail in [4,7,8]. To our knowledge, however, there is no direct quantitative relationship between the IFS parameters $\mathbf{w}$ and $\mathbf{p}$ and the Hausdorff distance $h(S, A)$ in Eq. (4.2). As such, there appears to be no definite algorithm which could indicate how these parameters should be modified to (i) further minimize $h(S, A)$ or (ii) better approximate the shading measure on S . The use of moments, as outlined in the next section, may provide a useful and quantitative scheme in this regard.

4.2 Moment Methods

For simplicity, we first focus on the inverse problem of construction on $\mathbb{R}$: extensions to the plane follow.

Suppose that we are given a target set S , scaled, for convenience, so that $S \subset [0,1]$, and supporting a measure ν , with moments $G_n = \int_S x^n d\nu$. Diaconis and Shahshahani [11] proposed that a knowledge of the moments $g_n = G_n$ could, in principle, be used to solve for the IFS parameters in Eq. (3.4). However, these equations are highly nonlinear with many solutions, and initial attempts at solving them have proved fruitless [13]. The complexity of this approach increases enormously in the two-dimensional case, Eq. (3.10).

As such, we have investigated an approach which seeks to minimize the "distance" between the true moments, G_n , and the moments, g_k , of a $\mathbf{p}$ -balanced measure supported by the attractor of an IFS. For a fixed N , the number of maps in the IFS, and M , the number of moments

G_n we wish to use, we have considered the (squared) Euclidean distance in "moment space", defined over the feasible parameter space Π_1^N :

$$D_M^N(\pi) \equiv \sum_{i=1}^M (g_i(\pi) - G_i)^2 h(i), \quad (4.3)$$

where $h: \mathbb{Z}^+ \rightarrow \mathbb{R}$ is a weight function. The inverse problem is now cast as a problem of minimizing D_M^N over Π_1^N . The nature of this function is complicated and has not been well explored, even in the more trivial cases. (It is smooth, but by no means necessarily convex.) Only in the limit $M \rightarrow \infty$ does $D \rightarrow 0$ imply the weak-* convergence $\mu \rightarrow \nu$. For finite M , this is not the case, nor does a small value of D necessarily imply a "closeness" of the two sets.

Note that the derivatives $dg_i/d\pi_j$, may be obtained in closed form, thus providing us with the vector, $\text{grad } D_M^N$. As such, gradient methods for optimization may be employed. (Our early investigations involved rather naive steepest descent method.) The numerical results reported here were obtained using the Harwell Library FORTRAN Subroutine VEO1AD. This routine performs a generalization of Davidon's method [14] which uses the gradient, approximates the hessian of D and deals with linear inequality constraints by projection techniques. Table 2 presents some sample calculations where the target set S is the ternary Cantor set on $[0,1]$, with uniform measure. The initial approximations to S are (i) $A=[0,1]$ with uniform measure, (ii) A a Cantor-like set with nonuniform measure. In both cases, the moments G_i , $i=1, \dots, M=20$ were employed. The weight function was simply taken to be $h(i) = 1$, i.e. equal weighting of the moments. Also presented are the values of the moment distance function D_M^N .

The inverse problem in $\mathbb{R}^2$ represents an even more interesting, as well as more difficult, computational problem. The number of parameters π_k grows as $7N$. As well, the objective functions D_M^N exhibit complicated structure, with a proliferation of local minima for which $D > 0$. There is also the possibility of a "degeneracy" of representations due to special symmetries of the target set S . We illustrate with a simple example: Consider the unit square on $[0,1] \times [0,1]$, with uniform measure, as the target set S . There are a continuum of possibilities of tiling this set with affine copies of itself. For example, any point (a,b) in the interior of this set determines four rectangles having this point as a common vertex, each of which are affine copies of S . In addition, each of these copies may be generated from S in a non-unique way. One could also imagine the existence of interfering minima lying on rays in parameter space, again for which $D > 0$.

In Figure 2 are presented some "snapshots" of the optimization method when S is the spleenwort fern attractor of Fig. 1(b). The reference moments G_{ij} , $i+j \leq 5$, were employed, i.e. $M=14$. They were calculated from the fern parameters in Table 1 using the moment equations (3.10). The approximating sets A are shown in dark, superimposed over the lightly shaded target set S . The squared distance in moment space D as well as the number of iterations, i.e. calls to the optimization subroutine, are given with each approximation. The degree of accuracy achieved with a relatively small amount of moments is encouraging. The entire calculation required about 3 min. CPU on an IBM 4341 mainframe.

5. Summary and Acknowledgments

We have outlined the theory and preliminary application of a moment-type method to the inverse problem of fractal set approximation with iterated function systems. Idealistically, one would hope that the method would be relatively insensitive to the accuracy of the initial

approximation to the IFS. As such, it could eliminate a dependence on an "artificial intelligence" to ascertain any rough geometrical symmetries of the target set S . The study is in its infancy. There are many avenues to be explored, for example: (1) a more detailed investigation of the objective function D in Eq. (4.3) for very simple cases, e.g. Cantor sets on the line, (2) examining various weight functions $h(i)$, (3) how the moment method handles "redundancy", e.g. how will it tolerate five IFS maps to approximate a four-map fern, (4) optimization routines which may improve the convergence of the method. The method of *simulated annealing*, which is employed in the Barnsley image compression method [4], is being explored to possibly bypass the problems encountered with interfering local minima. Also being formulated (with B. Forte) is a maximum-entropy-type principle [16] to optimize the information extracted from a given set of moments G_n .

We wish to thank Professors B. Forte, J. Elton and G. Mantica for stimulating conversations and encouragement in pursuing this problem. This work was supported in part by an Operating Grant (ERV) from the Natural Sciences and Engineering Research Council of Canada.

References

- [1] J. Hutchinson, Fractals and self-similarity, *Indiana Univ. J. Math.* **30**, 713-747 (1981).
- [2] M.F. Barnsley and S. Demko, Iterated function systems and the global construction of fractals, *Proc. Roy. Soc. Lond.* **A399**, 243-275 (1985).
- [3] M.F. Barnsley, V. Ervin, D.P. Hardin, J. Lancaster, Solution of an inverse problem for fractals and other sets, *Proc. Nat. Acad. Sci. USA*, **83**, 1975-1977 (1986).
- [4] M.F. Barnsley and A.D. Sloan, A better way to compress images, *BYTE Magazine*, January issue, 215-223 (1988).
- [5] J. Elton, An ergodic theorem for iterated maps, *Ergod. Th. and Dynam. Sys.* **7**, 481-488 (1987).
- [6] M.F. Barnsley, S.G. Demko, J. Elton and J.S. Geronimo, Invariant measures for Markov processes arising from iterated function systems with place-dependent probabilities, *Ann. Inst. H. Poincaré* (to appear).
- [7] M.F. Barnsley, *Fractals Everywhere*, Academic Press, NY, 1988.
- [8] M.F. Barnsley, Fractal modelling of real world images, in *The Science of Fractal Images*, by H.O. Peitgen and D. Saupe Edit., Springer Verlag, NY (1988).
- [9] W.D. Withers, Differentiability with respect to parameters of average values in probabilistic contracting dynamical systems, preprint (1987).
- [10] K.J. Falconer, *The geometry of fractal sets*, Cambridge University Press (1985).
- [11] P. Diaconis and M. Shahshahani, Products of random matrices and computer image generation, in *Random Matrices and Their Applications*, Vol. 50, *Contemp. Math.*, AMS, Providence, RI (1986).
- [12] N.I. Akhiezer, *The Classical Moment Problem*, Hafner, NY (1965).
- [13] D. Bessis and G. Mantica, private communication.

- [14] P.E. Gill, W. Murray and M.H. Wright, *Practical Optimization*, Academic Press, NY (1981).
- [15] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, *Science* **220**, 672 (1983).
- [16] L. Mead and N. Papanicolaou, Maximum entropy principles, *J. Math. Phys.* **25**, 2404 (1984) and references therein.

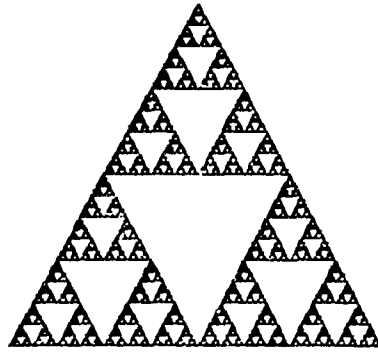


Figure 1(a): Sierpinski Gasket



Figure 1(b): Spleenwort Fern

Table 1: Spleenwort fern parameters

i	$a_{11}(i)$	$a_{12}(i)$	$a_{21}(i)$	$a_{22}(i)$	$b_1(i)$	$b_2(i)$	p_i
1	0.00	0.00	0.00	0.16	0.50	0.00	0.01
2	0.20	-0.26	0.23	0.22	0.40	0.05	0.07
3	-0.15	0.28	0.26	0.24	0.57	-0.12	0.07
4	0.85	0.04	-0.04	0.85	0.08	0.18	0.85

Table 2(i)

Step	s_1	s_2	a_1	a_2	p_1	p_2	D
1	0.500	0.500	0.000	0.500	0.500	0.500	4.439D-02
5	0.499	0.537	-0.019	0.504	0.488	0.514	1.719D-03
10	0.481	0.449	-0.116	0.559	0.437	0.568	9.608D-05
20	0.224	0.376	0.028	0.625	0.466	0.535	7.102D-07
30	0.248	0.363	0.023	0.637	0.476	0.525	4.462D-07
40	0.279	0.354	0.014	0.647	0.484	0.517	2.773D-07
50	0.302	0.346	0.008	0.655	0.490	0.510	1.102D-07
60	0.328	0.337	0.001	0.663	0.497	0.503	2.968D-08
70	0.333	0.333	0.000	0.667	0.499	0.500	1.007D-13

Table 2(ii)

Step	s_1	s_2	a_1	a_2	p_1	p_2	D
1	0.300	0.300	0.000	0.300	0.300	0.700	6.464D-01
5	0.269	0.429	-0.015	0.559	0.340	0.658	3.805D-02
10	0.294	0.422	-0.034	0.579	0.429	0.570	3.057D-05
20	0.295	0.347	0.010	0.654	0.487	0.512	1.930D-07
30	0.314	0.340	0.005	0.660	0.492	0.506	7.462D-08
40	0.333	0.333	0.001	0.667	0.498	0.500	1.876D-08
50	0.333	0.333	0.001	0.667	0.498	0.500	1.633D-12

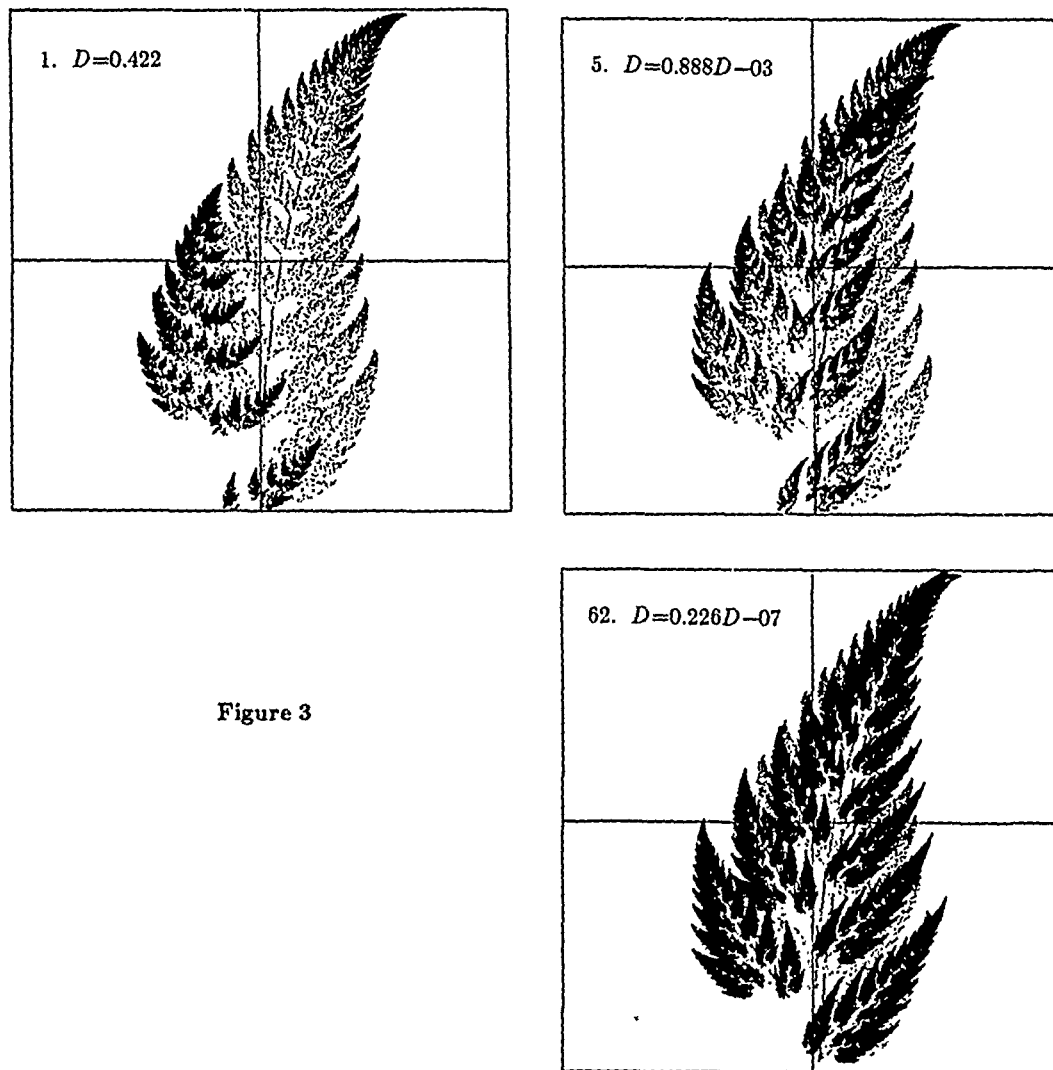


Figure 3

Working with ruled surfaces in solid modeling

John K. Johnstone

Dept. of Computer Science, Johns Hopkins Univ., Baltimore, MD 21218

Abstract. *The interplay between algebraic geometry and graphics/solid modeling is a natural and strong one. This paper addresses the topic of ruled surfaces, a class that has long been of interest to the mathematical community, and brings it more squarely into the realm of computer science by giving constructive algorithms for ruled surfaces. These algorithms allow ruled surfaces to be used more easily in a solid modeling system. Specifically, we show (a) how to identify that a surface is ruled from its equation (b) how to find the generator through a given point of a ruled surface and (c) how to find a directrix curve of a ruled surface. As an example of how these algorithms can be put to use in a solid modeling environment, we show how to parameterize a ruled surface.*

Ruled surfaces share properties of both curves and surfaces, which make ruled surfaces a very useful class in the difficult transition between curves and surfaces in solid modeling. They can be used to extend algorithms for curves (which are easier to develop) to algorithms for surfaces. The mathematical theory of curves and surfaces can continue to guide their incorporation into solid modelers although, as is shown in this paper, computer scientists will often have to develop constructive techniques to replace existential mathematical statements.

1 Introduction

Surfaces are more complicated objects than curves. Consequently, a number of problems in geometric modeling have been solved for curves but not for surfaces. Ruled surfaces (which are surfaces that can be generated by sweeping a line through space and include planes, cylinders, cones, saddles, and other important surfaces) are a useful subclass of surfaces, because they bridge the gap in complexity between curves and surfaces. Since they can be defined by a plane curve (the directrix curve) lying on the surface, algorithms for ruled surfaces tend to be easier to develop than algorithms for arbitrary surfaces.

Ruled surfaces are not only of interest because of their kinship to curves. Many of the surfaces that solid modelers use as primitives from which to build models are ruled surfaces. Three of the four most common surfaces (the plane, cylinder, cone, and sphere) are ruled, as are four of the six quadric surfaces. Ruled surfaces also arise naturally in man-made objects (e.g., propellers, car bodies [3]), since the sweeping of a line through space is a fundamental operation in machining. Finally, ruled surfaces bear a strong relationship to the generalized cylinder method of shape representation that is widely used by the vision community [7,8].

In this paper, we show how to perform the crucial reduction step from ruled surfaces to plane curves: finding a plane directrix curve of a ruled surface. Intuitively, the directrix curve is the curve that the line sweeps along in generating the ruled surface. A complementary step, finding the direction of the line at a given point of the directrix curve, is presented. The solutions to these two problems allow one to retrieve the generating sweep of a ruled surface from its implicit equation. The paper also clarifies the notion of ruled surface, by proving the equivalence of two definitions of ruled surface.

Although we are guided at all times by the mathematical theory, we shall need to introduce more precision in order to create constructive algorithms for ruled surfaces. For example, a suggestion of Theorem 2 below can be found in the mathematical literature, but it is imprecise. In particular, Sommerville states that "in general any plane section [of a ruled surface], not containing any generator, is a directrix curve" [9], but does not offer any elaboration of what 'in general' means. Moreover, he does not prove the statement (although a proof would have been of no use to us anyway, since his definition of directrix curve is different from ours).

The paper is structured as follows. Section 2 presents key definitions, some of which are purposely changed from the typical ones in the literature in order to satisfy the more constructive requirements of this paper. Section 3 establishes some basic facts that are important for the rest of the paper. The core of the paper is in Sections 4 and 5, where a method for finding the generator through a point, for identifying a ruled surface, and for finding a directrix curve are presented. Section 6 shows how to extend algorithms for plane curves to ruled surfaces (through an example) and sometimes even to arbitrary surfaces. The paper ends with some conclusions. All curves and surfaces in this paper are assumed to be nonlinear, irreducible, and algebraic. (Recall that a surface is algebraic if it can be defined by a polynomial $f(x, y, z) = 0$. It is reducible if it can be expressed as the union of two algebraic surfaces.)

2 Definitions

In this section, we introduce definitions for the ruled surface and its key components. There are several choices for the definition of a ruled surface. The weakest definition is that a ruled surface is (1) a surface that is equal to the union of a set of lines. (Equivalently, through every point of a ruled surface R , there exists a line that is completely contained in R .) A stronger definition that is the usual definition in the mathematical literature [2,9] and the one that we shall use is (2) a surface that can be generated by smoothly sweeping a line through space. (More formally, a ruled surface is the image of a continuous map $S : I \subset \mathbb{R} \rightarrow \mathbb{R}^3$, where $S(t)$ is a line for all $t \in I$. We also insist that the map is one-to-one almost everywhere: we do not want sweeps that return to sweep over a portion of the surface a second time.) An even stronger definition is (3) a surface that can be generated by sweeping a line along a plane algebraic curve (motivated by the observation that all of the quadric ruled surfaces can be generated by sweeping a line along a line or ellipse). We shall prove that this strongest definition is actually an equivalent definition when the surface is algebraic (Corollary 4).

A straight line that lies in a ruled surface is called a *generator* of the surface. Usually, there is only one generator through a point of a ruled surface. If there are two or more generators through a typical point of a ruled surface, then the ruled surface is *doubly ruled*, otherwise it is *singly ruled*. The only doubly ruled algebraic surfaces are the plane, the hyperbolic paraboloid, and the hyperboloid of one sheet [4].

A simple definition of a directrix curve in the literature is a curve on the ruled surface that is intersected by every generator of the surface [9]. However, this definition does not lend itself well to use in geometric modeling algorithms. For example, even the vertex of a cone would be a directrix curve under this definition. In particular, it should be easy to generate the surface from the directrix curve. Thus, we use a more pragmatic definition: a *directrix curve* of the ruled surface R is a curve C on R such that the generators that intersect C , choosing only one generator for every point of C , are sufficient to generate R . A directrix curve C is (almost) *strong* if (almost) every point of C is intersected by exactly one generator.

Example 2.1 *An elliptical cross-section of a hyperboloid of one sheet is a directrix curve.*

An elliptical cross-section of a circular cylinder or a cone is a strong directrix curve.

As it is presently defined, there is no assurance that the ruled surface can be generated by smoothly sweeping a line along a directrix curve. (In particular, the lines that intersect a directrix curve are enough to generate the surface, but it is not clear that it is possible to find a smooth sweep through them.) We now establish this fact for almost-strong directrix curves.

Lemma 1 *Let R be a ruled surface and let S be an almost-strong directrix curve of R . R can be generated by sweeping a line along S .*

Proof: We will show that R can be generated by sweeping a line along a strong directrix curve S . This is sufficient because, for the purposes of sweeping, an almost-strong directrix curve is equivalent to a strong directrix curve: of two adjacent points on an almost-strong directrix curve, only one can be intersected by two or more generators (by definition) and only one of these two or more generators can smoothly change into the single generator of the adjacent point.

By definition, R can be generated by sweeping a line through space. We shall show that the line must actually sweep along S . Suppose that the sweep leaves S at x . It must return and sweep through x again, otherwise points of S in some neighbourhood of x will not be visited by the sweep. Since there is only one generator through x , the sweep is in the same orientation when it leaves S at x as it is when it passes through x again (staying on the curve this time). Thus, it is possible to sweep smoothly along S without leaving: if the sweep ever leaves S at P , then simply follow the alternate direction from P that stays on S rather than leaving. This is a generating sweep, since by the definition of strong directrix curve, a sweep that passes through all of the points of S must be a generating sweep. ■

Lemma 1 shows that the term 'directrix curve' is well chosen: a directrix curve does indeed direct how the line should be swept through space. In Section 5, we will show that every singly ruled surface has an almost-strong directrix curve.

3 Facts about singularities of ruled surfaces

We begin with some theory about the singularities of a ruled surface, which play a key role in the development of our algorithms (as they do in most algorithms that deal with curves and surfaces). Geometrically, a *singularity of a surface* is a point at which the tangent plane is undefined [4]. Algebraically, a singularity of the surface $f(x, y, z) = 0$ is a point P such that $(\text{grad } f)(P) = 0 = f(P)$ [6], where $(\text{grad } f)()$ is the gradient of f , the vector of its partial derivatives. To see the equivalence of these definitions, recall that the tangent plane of a nonsingular point P is the plane perpendicular to the surface normal at P and $(\text{grad } f)(P)$ is the surface normal of a nonsingular point.

Before we can continue, we must present a very important theorem from algebraic geometry.

Theorem 1 (Bezout's Theorem, [6])

- (a) *An algebraic curve of degree m and an algebraic curve of degree n have at most mn intersections, unless one of the curves is contained in the other curve.*
- (b) *An algebraic curve of degree m and an algebraic surface of degree n have at most mn intersections, unless the curve is contained in the surface.*

- (c) An algebraic surface of degree m and an algebraic surface of degree n intersect in a collection of algebraic curves, unless one of the surfaces is contained in the other surface. The sum of the degrees of these curves of intersection is at most mn .

Lemma 2 The set of singularities of an irreducible algebraic surface is a finite set of algebraic curves and a finite set of points.

Proof: The proof is an application of the surface-surface and curve-surface versions of Bezout's Theorem to the intersection of four surfaces. By Theorem 1c, the solution set of $f_x = 0, f = 0$ is a finite set of algebraic curves, since f is irreducible and f is not a component of f_x (being of higher degree). Thus, $(f_x = 0, f = 0), f_y = 0$ is a finite set of points and algebraic curves (Theorem 1b). Similarly, $(f_x = 0, f = 0, f_y = 0), f_z = 0$ is a finite set of points and algebraic curves. ■

We now move on to the actual algorithms for ruled surfaces.

4 Computing the generator through a given point

In sweeping a line through space, it is not enough to know the curve along which to sweep. One must also know the direction of the line at every point of the curve. Thus, a crucial algorithm for ruled surfaces is computing the generator through a point. We wish to develop a formula for the generator through a point P of the ruled surface $f(x, y, z) = 0$. This formula should depend upon f and P . The first step is to find a necessary and sufficient condition for the line $P + tV$ to be a generator through P . The most obvious condition is $f(P + tV) = 0$ for all t , but the following condition proves to be more useful.

Lemma 3 Let P be a point of the ruled surface $f(x, y, z) = 0$ and let $V \in \mathbb{R}^3$, $V \neq 0$. Then the following are equivalent:

1. The line $P + tV$ is a generator of f .
2. $V \cdot (\text{grad } f)(P + tV) = 0$ for all $t \in \mathbb{R}$.

Proof: Differentiate $f(P + tV) = 0$ with respect to t , using the chain rule. ■

Lemma 3 will now be used to find the generator through P , by solving for V . In the equation $V \cdot (\text{grad } f)(P + tV) = 0$, P will be treated as a symbolic constant, since the formula for P 's generator should depend upon P . Thus, $V \cdot (\text{grad } f)(P + tV) = 0$ is an equation of degree n in the four variables $V = (v_1, v_2, v_3)$ and t . The following observations will be used to solve for v_1, v_2 , and v_3 .

- (1) $V \cdot (\text{grad } f)(P) = 0$ is a linear equation in the v_i . It can be used to solve quickly for one of the v_i in terms of the other two. (If P is a singularity, then $(\text{grad } f)(P) = 0$ and the above equation is trivial rather than linear. Thus, the formula that we develop will not be valid for singular points of the surface.)
- (2) Only V 's direction is important, not its length. Therefore, one of V 's coordinates (say v_j) can be set to 1. Both possibilities, $v_{i \oplus 1} = 1$ and $v_{i \oplus 2} = 1$, should be considered independently (where $\oplus$ is addition mod 3).

Two variables are eliminated by (1) and (2). That is, one can always choose $i \neq j$, because if v_j is the only nonzero coordinate of V , it cannot be a linear combination of $v_{j \oplus 1}$ and $v_{j \oplus 2}$. Thus, (1) and (2) reduce $V \cdot (\text{grad } f)(P + tV) = 0$ to an equation E in t and (without loss of generality) v_1 . All that remains is to solve for v_1 in E . Viewed as a polynomial in t , E has an infinite number of roots. Thus, each of the coefficients of

E must be zero, where the coefficients are polynomials in v_1 . This creates a system of n equations in v_1 . The following lemma establishes that, for singly ruled surfaces, the system of equations must yield exactly one value for v_1 . By Lemma 5, the system has two solutions for doubly ruled surfaces.

Lemma 4 *If the surface $f(x, y, z)$ is singly ruled, then $V \cdot (\text{grad } f)(P + tV) = 0$ has only one solution for V for general P .*

Proof: The above method derives a system of equations in v_1 and the symbolic constant P (where P represents an arbitrary nonsingular point). Each solution for v_1 (which may depend upon P) generates a vector V such that $P + tV$ is a generator through P . Suppose that the system has α simultaneous solutions for v_1 : $v_{1,1}, v_{1,2}, \dots, v_{1,\alpha}$. I claim that almost all of the points of the surface will be struck by α generators. The only points that might not be struck by α generators are singularities and points such that two of the solutions for v_1 become identical. That is, because the above method, in particular step (1), assumes that P is nonsingular, it cannot be used to make any conclusions about singularities; and if, for example, the solutions are $v_{1,1} = 1 - p_1$ and $v_{1,2} = 1 + p_1$, then most points will be struck by two generators but a point such as $P = (0, 3)$ will be struck by only one generator. Consider the points such that two of the solutions for v_1 become identical. View a solution for v_1 as a polynomial in P and view this polynomial as a surface. Then one can see that the points P_0 such that $v_{1,i}(P_0) = v_{1,j}(P_0)$, $i \neq j$ must form a set of measure zero (as a subset of the surface), because two distinct surfaces intersect in a set of measure zero with respect to any surface (Bezout's Theorem 1c). Similarly, the singularities of $f(x, y, z)$ form a set of zero measure as a subset of $f(x, y, z)$ (Lemma 2). Therefore, the points of the surface that are not struck by α generators form a set of zero measure. In particular, if $\alpha \geq 2$, then the surface is not singly ruled. ■

Corollary 1 *A point of a singly ruled surface that is intersected by two or more generators is a singularity.*

Proof: The set of points such that two of the solutions for v_1 become identical is empty, because there is only one solution to start with. ■

Let us review the status of our journey towards the computation of a generator through a given point P . We have reduced the generator computation for a nonsingular point to the solution of a system of equations in a single variable v_1 and established that, for singly ruled surfaces, there is only one solution for v_1 to be found. We shall solve the equation of lowest degree in the system for v_1 , and then eliminate those solutions that are not valid for the entire system by substituting into the other equations. Finally, each v_1 solution is grown into a $V = (v_1, v_2, v_3)$ vector.

There are two cases to consider, depending on the degree of the equation of lowest degree. If it is of degree less than 5, then it can be solved symbolically, so there is no problem with the symbolic constant P and we can find a formula for the generator(s) through any nonsingular point of the surface. However, if the equation of lowest degree is of degree 5 or higher, then the best we can do is to plug a specific point P_0 into the equation and solve it numerically, i.e., find the generator through one point at a time.

Example 4.1 $f(x, y, z) = y^2 - x^2 - z = 0$ is the equation of a hyperbolic paraboloid. $\text{grad } f(x, y, z) = (-2x, 2y, -1)$ and $V \cdot \text{grad } f(P) = -2p_1v_1 + 2p_2v_2 - v_3$, which can be used to set $v_3 = 2p_2v_2 - 2p_1v_1$. By setting $v_2 = 1$, $V \cdot \text{grad } f(P + tV)$ becomes $2t(1 - v_1^2)$. This polynomial is identically zero (when viewed as a polynomial in t), leading us to conclude that all of its coefficients are zero. That is, $2(1 - v_1^2) = 0$ or $v_1 = \pm 1$. (This is a rather trivial system of equations.) Therefore, $V = (1, 1, 2p_2 - p_1), (-1, 1, 2p_2 + p_1)$. Since v_1 is nonzero in these solutions, there is no need to go back and check $v_1 = 1$.

The method that we have described for finding the generator(s) $P+tV$ through a point P can also be used to discover if a surface is ruled. Given the surface $f(x,y,z) = 0$, one solves $V \cdot \text{grad } f(P+tV) = 0$ to find V such that $P+tV$ is a generator. If no V can be found or it is invalid (e.g., complex), then the surface is not ruled.

Example 4.2 $f(x,y,z) = x^2 + y^2 - z = 0$ is the equation of an elliptic paraboloid. $\text{grad } f(x,y,z) = (2x, 2y, -1)$ and $V \cdot \text{grad } f(P) = 2p_1v_1 + 2p_2v_2 - v_3$, which can be used to set $v_3 = 2p_1v_1 + 2p_2v_2$. By setting $v_2 = 1$, $V \cdot \text{grad } f(P+tV)$ becomes $-2p_2 + (2v_1^2 + 2)t$, whose coefficients must be zero. In particular, the linear term $2v_1^2 + 2$ implies $v_1 = \sqrt{-1}$, which is not valid. We conclude that this paraboloid is not ruled.

5 Finding a directrix curve

In the theory of ruled surfaces, directrix curves are a very important structure that will often be used in algorithms. Moreover, the directrix curve will be the main means of translating ruled surface problems to plane curve problems. Therefore, the development of a method for finding a directrix curve on a ruled surface is crucial. The following example suggests how this might be done.

Example 5.1 We would like to show that a planar cross-section of the ruled surface can be used as a directrix curve. However, a plane parallel to a cylinder's axis will not intersect the cylinder in a directrix curve, although all other planes will create an elliptical directrix curve. This suggests that one should not choose a plane that contains a generator of the cylinder. As another example, a planar cross-section of a cone that passes through the cone's vertex will not generate a directrix curve, although all other cross-sections shall. This suggests that points of the ruled surface that have several generators passing through them (the singularities of a singly ruled surface) can cause problems.

We use these observations to find a directrix curve.

Theorem 2 Let R be a ruled surface. If P is a plane that

- (a) intersects R
- (b) does not contain any generator of R
- (c) does not contain any singularity of R that is intersected by an infinite number of generators, and
- (d) does not contain an entire irreducible singularity curve of R .

then $R \cap P$ is a directrix curve.

Proof: This theorem is more easily proved if we work over projective space. R and P should be viewed as point sets in projective space. The restrictions on P must also hold over projective space, i.e., P must not contain any generators at infinity or singularities at infinity either.

The first requirement for a directrix curve C is that the generators that intersect C cover the surface. This is established for $R \cap P$ by showing that almost all generators of the ruled surface intersect P . Suppose for the sake of contradiction that an infinite number of R 's generators are parallel to P . In projective space, P contains a line at infinity and every generator that is parallel to P will hit this line. Thus, R has an infinite number of intersections with P 's line at infinity, implying by Bezout's Theorem (Theorem 1(b)) that P 's line at infinity is contained in the ruled surface. But this violates our assumption

that P does not contain any generator of R . We conclude that only a finite number of R 's generators are parallel to P or, equivalently, that almost all of R 's generators intersect P .

For the second requirement of a directrix curve, we must show that the generators that intersect $R \cap P$ still cover the surface if we restrict to one generator through each point of $R \cap P$. The proof for doubly ruled surfaces is straightforward and we only consider singly ruled surfaces. By Corollary 1, it suffices to show that $R \cap P$ contains only a finite number of surface singularities, each of which is intersected by only a finite number of generators. By Lemma 2, the singularities of R consist of a finite set of algebraic curves and a finite set of points. Because of restriction (d), the plane P can intersect every singularity curve in at most a finite number of points (Theorem 1(a)). Therefore, P contains only a finite number of singularities. Restriction (c) guarantees that each of these is intersected by only a finite number of generators. ■

This offers us a way of finding a directrix curve.

Corollary 2 *A randomly chosen planar cross-section of a ruled surface is, with probability one, a directrix curve.*

Proof: There are a finite number of singularity curves (Lemma 2). It can be shown that there are only a finite number of points that are struck by an infinite number of generators. Finally, of the planes that go through a given point of the surface, only a subset of measure zero (with respect to the entire set) will contain the finite number of generators through that point. We conclude that the set of planes that satisfy the restrictions of Theorem 2 are dense in the set of planes. ■

Corollary 3 *A ruled surface always has a plane algebraic directrix curve.*

For a singly ruled surface, the directrix curve of Theorem 2 will be almost-strong, since it will contain a finite number of singularities and singularities are the only points of a singly ruled surface that can be intersected by more than one generator (Corollary 1). This establishes two very interesting corollaries.

Corollary 4 *An algebraic ruled surface can be generated by sweeping a line along a plane algebraic curve. That is, for algebraic surfaces, the last definition of ruled surface in Section 2 is actually equivalent to (not stronger than) the definition we are using.*

Proof: A singly ruled surface has an almost-strong directrix curve, so apply Lemma 1. For doubly ruled surfaces, notice that they can be generated by sweeping a line along a line or ellipse. ■

Corollary 5 *There is a unique way of generating a singly ruled surface by sweeping a line through space.*

Proof: Let R be a singly ruled surface. It has an almost-strong directrix curve S . In the proof of Lemma 1, it was shown that every sweep of R contains a sweep along all of S . We claim that, for singly ruled surfaces, every sweep of R is exactly a sweep along S . A sweep along S already generates R , by definition of almost-strong directrix curve. Thus, every generator that does not intersect S will be singular (Corollary 1), since every point of this line is already contained in a generator through S . We conclude that there are only a finite number of generators that do not intersect S (Lemma 2). Thus, every sweep of R is a sweep along S . However, there is a unique way of sweeping a line along S : the line must sweep smoothly along S (with no changes of direction, because of the injective nature of a sweep), there is no flexibility in the choice of generator through a nonsingular point, and there is no choice in the direction to take at a singularity (because of both the surjective and injective nature of a sweep). ■

6 An application of the theory

In this section, we show how algorithms for plane curves can be extended to ruled surfaces. We use the example of parameterization, which uses the methods that we have developed in a straightforward manner. Parameterization is a problem that has been solved for (rational algebraic) plane curves [1] but not for surfaces (of degree higher than three). It is important because both the implicit and the parametric representation of a curve or surface are useful, so solid modelers desire the capability to translate between them. In order to parameterize a ruled surface, we proceed as follows. Given a surface $f(x, y, z)$, if necessary first test that it is ruled (Section 4). Assuming that it is ruled, find a plane directrix curve C (Section 5) and a formula V_p for the generator through P (Section 4). Then the parameterization of $f(x, y, z)$ is $c(s) + V_{c(s)}t$, where $c(s)$ is a parameterization of the plane curve C . A similar technique can be used for extending other plane curve algorithms to ruled surfaces, using the fact that a ruled surface is well defined by one of its directrix curves.

Example 6.1 Consider the hyperbolic paraboloid $f(x, y, z) = y^2 - x^2 - z = 0$ again. In Example 4.1, we showed that the two generators through a point P of the surface are $P + t(1, 1, 2p_2 - p_1)$ and $P + t(-1, 1, 2p_2 + p_1)$. None of the generators $P + t(\pm 1, 1, 2p_2 \mp p_1)$ lie in the $x = 0$ plane, and there are no singularities to avoid (since $f_x = -1 \neq 0$ is impossible). Therefore, a directrix curve can be generated by the cross-section of the surface by the plane $x = 0$, yielding the parabola $\{x = 0, y^2 - z = 0\}$. A parameterization of this directrix curve is clearly $(x, y, z) = (0, t, t^2)$. Thus, a parameterization of the hyperbolic paraboloid is $(0, s, s^2) + (1, 1, 2s)t = (t, s + t, s^2 + 2st)$.

Some algorithms for ruled surfaces can themselves be extended to algorithms for arbitrary surfaces. For example, the intersection of two surfaces $f(x, y, z)$ and $g(x, y, z)$ is equivalent to the intersection of $f(x, y, z)$ and a linear combination of $f(x, y, z)$ and $g(x, y, z)$. By finding a linear combination that is a ruled surface, one can take advantage of ruled surface properties to compute the intersection. (It is known that given any two quadric surfaces, it is always possible to find a linear combination that is ruled [5].)

7 Conclusions

In this paper, we have presented techniques for manipulating ruled surfaces. These algorithms make it easier for the computer scientist to incorporate this rich and interesting class into solid modeling systems. We have also rigourized some of the mathematical theory of ruled surfaces and developed some new facts. A future direction is to consider an extension of ruled surfaces: surfaces generated by sweeping low degree algebraic curves (e.g., circles) through space, rather than lines. This would enrich the class of surfaces while maintaining the simplicity of ruled surfaces. Under this extension, for example, one could model a sphere.

8 Appendix

The following lemma is important but it is discussed here because it would have disrupted the flow of the main exposition.

Lemma 5

1. Doubly ruled surfaces are nonsingular.

2. Every point of a nonplanar doubly ruled surface is intersected by exactly two generators.

Proof: (1) is easily established by testing the algebraic criteria for a singularity on the normal forms for the three doubly ruled surfaces (such as $\frac{y^2}{b^2} - \frac{z^2}{a^2} - \frac{x}{c} = 0$ for hyperbolic paraboloid [10]). For (2), let P be a point of a nonplanar doubly ruled surface R . P is intersected by at least two generators, by definition. P is not intersected by three coplanar generators, since their common plane would be a component of R (by Bezout's Theorem 1c), contradicting the irreducibility of R . It remains to show that P is not intersected by three noncoplanar generators. A well-known fact about ruled surfaces is that the tangent plane of any (nonsingular) point of a generator contains the generator. (Lemma 3 provides a simple proof of this fact, since $(\text{grad } f)(\cdot)$ is the surface normal.) Thus, if P is intersected by three noncoplanar generators, it must be a singularity, since a single tangent plane cannot contain all of the generators through P . But doubly ruled surfaces do not have any singularities. We conclude that P is intersected by exactly two generators. ■

References

- [1] S. S. Abhyankar and C. Bajaj, *Automatic parameterization of rational curves and surfaces III: algebraic plane curves*, Computer Aided Geometric Design, 1988, to appear.
- [2] W. L. Edge, *The Theory of Ruled Surfaces*, Cambridge University Press, Cambridge, 1931.
- [3] V. O. Gordon and M. A. Sementsov-Ogievskii, *A Course in Descriptive Geometry*, Mir Publishers, Moscow, 1980.
- [4] D. Hilbert and S. Cohn-Vossen, *Geometry and the Imagination*, P. Nemenyi, translator, Chelsea, New York, 1952.
- [5] J. Levin, *A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces*, CACM, 19(1976), pp. 555-563.
- [6] D. Mumford, *Algebraic Geometry I: Complex Projective Varieties*, Springer-Verlag, New York, 1976.
- [7] R. Nevatia, *Machine Perception*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
- [8] S. A. Shafer and T. Kanade, *The theory of straight homogeneous generalized cylinders*, Technical Report CMU-CS-83-105, Dept. of Computer Science, Carnegie-Mellon University, January 1983.
- [9] D. M. Y. Sommerville, *Analytical Geometry of Three Dimensions*, Cambridge Univ. Press, Cambridge, UK, 1934.
- [10] G. B. Thomas, Jr. and R. L. Finney, *Calculus and Analytic Geometry*, Addison-Wesley, Reading, MA, 1981.
- [11] R. J. Walker, *Algebraic Curves*, Springer-Verlag, New York, 1950.

Using MACSyma to Calculate the Extrinsic Geometry of a Tube in a Riemannian Manifold

Harry S.D. Mills
Department of Mathematics, University of Idaho
and
Micheal H. Vernon
Division of Natural Sciences, Lewis Clark State College

Abstract. *In this paper we present a MACSyma batch file that calculates the second fundamental form of a tubal hypersurface of a Riemannian manifold. This program is currently being used to investigate the extrinsic geometry of tubes about totally umbilic submanifolds in a complex space form and is implemented on a Sun 3160.*

0. Introduction

Submanifold theory has had a long and fruitful history in differential geometry. Of particular interest is the study of hypersurfaces (a generalization of the concept of surface). Usually we are interested in the extrinsic geometry of the hypersurface - that is, aspects of the geometry of the hypersurface that are determined by how it sits in the ambient space: its shape, size, curvature, etc.. It turns out that all this information is embodied in a tensor field called the second fundamental form (see [5]).

The focus in this paper is on tubal hypersurfaces: given a known submanifold (say a curve) of a known ambient space (such as Euclidean 3-space), the tube of radius r about this core submanifold is the set of all points at a distance r from the core. The extrinsic geometry of the tube is completely determined by the extrinsic geometry of the core and the intrinsic geometry of the ambient space. In Section 1, the mathematical theory for this relationship is described: the extrinsic geometry of the core serves as initial conditions for a system of differential equations determined by the intrinsic geometry of the ambient space. The solution to the system generates the second fundamental form of the tube.

Laplace transform theory is tailored to the specific case at hand in Section 2. We

develop the structures used in a MACSyma batch file that calculates the second fundamental form of tubal hypersurfaces of any Riemannian manifold. This program is presented in Section 3. It requires as input

- 1) the real dimension of the hypersurface,
- 2) the curvature tensor of the ambient space, and
- 3) geometric aspects of the "core" submanifold including its second fundamental form.

The final output is the second fundamental form of the tube. Section 4 contains the results of a sample run.

1. Calculating the Second Fundamental Form of a Tube

The initial discussion will be of a general nature: that of calculating the second fundamental form of a tube in a semi-Riemannian manifold. (For more detail, see [1], [2], [3], [4] and [8].)

Recall first the notions of cut point and cut locus. (A detailed and analytic discussion of cut loci can be found in Vol II of [5] and in [6].) A cut point of a point p in a Riemannian manifold M is a point $c=\gamma(t)$, where γ is a geodesic emanating from $p=\gamma(0)$ with the property that the length of $\gamma([0,t])$ is the same as $d_M(p,c)$ and for $s>t$, the length of the curve $\gamma(J)$, $J=[0,s]$, is greater than the distance $d_M(p, \gamma(s))$. For instance, if $p \in S^2(r)$, its only cut point is its antipodal point.

The cut locus of a point $p \in M$, written $\text{Cut}(p)$, is the set of all cut points of p . The cut locus of a point on a sphere is a singleton, whereas for a point p on a cylinder over S^1 in $\mathbb{R}^3$, $\text{Cut}(p)$ is the axial line opposite p . Define $c(p)=\min\{d(p,q) \mid q \in \text{Cut}(p)\}$.

Let N^m be an immersed submanifold of a Riemannian manifold M^n . Define the unit normal sphere bundle of N by:

$$S^{\perp}(N) = \{X \in (T(N))^{\perp} \mid \|X\|=1\}.$$

Set $c(N)=\inf\{c(p) \mid p \in N\}$. Now for each $r \in (0, c(N))$, define the tube of radius r about N in M to be the hypersurface given by

$$N_r = \{\exp_q(rX) \mid p \in S^{\perp}(N)\}.$$

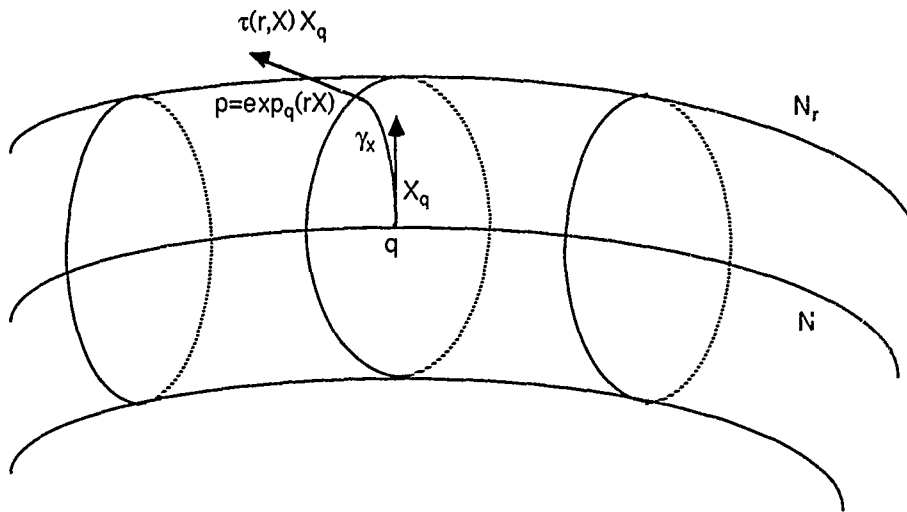
Let $\tau(t, X_q)$ be parallel translation of vector fields along the geodesic $(\gamma_X)_q : t \rightarrow \exp_q(tX)$.

For $p = \exp_q(rX) \in N_r$, $\tau(t, X_q) : T_q(M) \rightarrow T_p(M)$ is a linear isometry, ([7], p.66). By the generalized Gauss Lemma ([6], p. 121), we have

$$(1) \quad T_p(N_r) = \tau(r, X_q)(\{X_q\}^\perp) = \{\tau(r, X_q)X_q\}^\perp \quad \text{and}$$

$$(2) \quad T_p(N_r) \cong T_q(N) \oplus [\{X_q\}^\perp \cap T_q(N)^\perp]$$

where $\cong$ denotes the isomorphism of parallel translation.



For $X \in S^\perp(N)$ and $q \in N$, define $R_X(t) \in \text{End}[T_q(N)]$, for each $t > 0$, by

$$\tilde{R}_X(t)Y_q = \tau(t, X_q)^{-1} \{R(\tau(t, X_q)Y_q, \tau(t, X_q)X_q)\tau(t, X_q)X_q\}$$

where R is the curvature tensor of M . As we are primarily interested in the tangent space of the tube N_r , set

$$R_X(t) = \tilde{R}_X(t) |_{\{X\}^\perp}.$$

Finally, define $F(t, X) \in \text{End}(\{X\}^\perp)$, for each $X \in S^\perp(N)$ to be the solution of the initial value problem:

$$(3) \quad \frac{d^2}{dt^2} F(t, X_q) + R_X(t) \circ F(t, X_q) = 0$$

$$F(0, X_q) = P, \quad \frac{d}{dt} [F(t, X_q)] \Big|_{t=0} = (-A_X) \circ P + P^\perp$$

for each $q \in N$, where $P: \{X\}^\perp \rightarrow T(N)$ and $P^\perp: \{X\}^\perp \rightarrow T(N)^\perp \cap \{X\}^\perp$ are orthogonal projections of the vector bundle $\{X\}^\perp = T(N) \oplus [T(N)^\perp \cap \{X\}^\perp]$ onto the indicated component distributions, and A_X is the Weingarten map of X on N in M .

Theorem 1-[2, 4, 8]

The second fundamental form of N_r at $p = \exp_q(rX)$ is given by

$$(4) \quad H_r = \tau(r, X_q) \circ \left(\frac{d}{dt} F(t, X_q) \right) \Big|_{t=r} \circ F(r, X_q)^{-1} \circ \tau(r, X_q)^{-1}. //$$

Hence, in order to find an explicit representation of the second fundamental form of a tube, we need merely select a suitable basis of $T(N_r)$ using (1) and (2), solve (3) and then compute (4). Of course (4) says that $H_r \in \text{End}[T(N_r)]$ at $p = \exp_q(rX)$ is nothing more than parallel displacement of the endomorphism

$$\left(\frac{d}{dt} F(t, X_q) \right) \Big|_{t=r} \circ F(r, X_q)^{-1} \in \text{End}[\{X_q\}^\perp]$$

along the geodesic γ_X emanating from q and passing through p .

In case M is a symmetric space, once a suitable basis of $\{X_q\}^\perp$ is selected (where $q \in N$ and $X \in S^\perp(N)$), parallel displacement along the geodesic γ_X will preserve the basis and the respective orthogonality relations between its elements. Thus, in this case H_r will have the same matrix representation with respect to the displaced basis as $F'(r, X_q) \circ F(r, X_q)^{-1}$ has with respect to the chosen basis of $\{X_q\}^\perp$. This simplifies the calculation of (4) considerably.

One of the authors has successfully exploited this theory to build model spaces for the purpose of geometric classification of hypersurfaces (see [9] and [10]). These calculations were done by hand and were thereby restricted by tedium to certain cases.

2. The Computational Task

The authors have collaborated to write and implement a MACSyma program on a Sun 3/60 that automatically performs these calculations and hence enlarges the set of ambient spaces and "cores" that can be practically studied. We now discuss (3) and (4) from a computational point of view. As part of this process, we introduce the symbology used in the actual code.

Provided $F(t, X_Q)$ is of dimension K , we solve (3) by solving the K systems of differential equations corresponding to each of the K columns of $F(t, X_Q)$. The system arising from consideration of the j^{th} column of $F(t, X_Q)$ is the following:

$$\begin{aligned}
 & f''_{1,j} + r_{1,1}f_{1,j} + r_{1,2}f_{2,j} + \dots + r_{1,i}f_{i,j} + \dots + r_{1,k}f_{k,j} = 0 \\
 & f''_{2,j} + r_{2,1}f_{1,j} + r_{2,2}f_{2,j} + \dots + r_{2,i}f_{i,j} + \dots + r_{2,k}f_{k,j} = 0 \\
 & \vdots \\
 & f''_{i,j} + r_{i,1}f_{1,j} + r_{i,2}f_{2,j} + \dots + r_{i,i}f_{i,j} + \dots + r_{i,k}f_{k,j} = 0 \\
 & \vdots \\
 & f''_{k,j} + r_{k,1}f_{1,j} + r_{k,2}f_{2,j} + \dots + r_{k,i}f_{i,j} + \dots + r_{k,k}f_{k,j} = 0, \text{ where } r_{i,j},
 \end{aligned}
 \tag{5}$$

$f_{i,j}$, and $f''_{i,j}$ denote the i, j^{th} entries of R_X , $F(t, X_Q)$, and $\frac{d^2}{dt^2}F(t, X_Q)$, respectively.

Taking the Laplace transform of the i^{th} equation in (5) gives

$$s^2 \mathcal{L}(f_{i,j}) - s f_{i,j}(t_0) - f'_{i,j}(t_0) + r_{i,1} \mathcal{L}(f_{1,j}) + r_{i,2} \mathcal{L}(f_{2,j}) + \dots + r_{i,i} \mathcal{L}(f_{i,j}) + \dots + r_{i,k} \mathcal{L}(f_{k,j}) = 0, \text{ so}$$

$$r_{i,1} \mathcal{L}(f_{1,j}) + r_{i,2} \mathcal{L}(f_{2,j}) + \dots + (r_{i,i} + s^2) \mathcal{L}(f_{i,j}) + \dots + r_{i,k} \mathcal{L}(f_{k,j}) = s f_{i,j}(t_0) + f'_{i,j}(t_0), \text{ where}$$

$\mathcal{L}(f_{i,j})$ is the Laplace transform of the i, j^{th} entry of $F(t, X_Q)$ and $f_{i,j}(t_0)$ and $f'_{i,j}(t_0)$ are the i, j^{th} entries of $F(0, X_Q)$ and $\frac{d}{dt}F(t, X_Q) \Big|_{t=0}$, respectively.

Thus, we determine the Laplace transforms of the entries of the j^{th} column of $F(t, X_Q)$ by solving the matrix equation, $LMTRX.LFBAR = BBAR$, where

$$\text{LMTRX} = \begin{bmatrix} r_{1,1}+s^2 & r_{1,2} & \cdots & r_{1,k-1} & r_{1,k} \\ r_{2,1} & r_{2,2}+s^2 & \cdot & r_{2,k-1} & r_{2,k} \\ \vdots & \cdot & \cdot & \cdot & \vdots \\ r_{k-1,1} & r_{k-1,2} & \cdot & r_{k-1,k-1}+s^2 & r_{k-1,k} \\ r_{k,1} & r_{k,2} & \cdots & r_{k,k-1} & r_{k,k}+s^2 \end{bmatrix}, \text{LFBAR} = \begin{bmatrix} \mathcal{L}(f_{1,j}) \\ \mathcal{L}(f_{2,j}) \\ \vdots \\ \mathcal{L}(f_{k-1,j}) \\ \mathcal{L}(f_{k,j}) \end{bmatrix}, \text{BBAR} = \begin{bmatrix} sf_{1,j}(t_0)+f'_{1,j}(t_0) \\ sf_{2,j}(t_0)+f'_{2,j}(t_0) \\ \vdots \\ sf_{k-1,j}(t_0)+f'_{k-1,j}(t_0) \\ sf_{k,j}(t_0)+f'_{k,j}(t_0) \end{bmatrix},$$

and "." denotes matrix multiplication. Hence, $\text{LFBAR} = (\text{LMTRX})^{-1} \cdot \text{BBAR}$.

In the program to follow, the vector obtained by computing the inverse Laplace transforms of the entries of LFBAR is denoted by FBAR. The program will compute K different FBAR's, and affix each directly to $F(t, X_q)$, henceforth FMTRX, as a column thereof. Having computed FMTRX, it will remain to find $\frac{d}{dt}F(t, X_q) = \text{diff}(\text{FMTRX}, t)$, FMTRX^{-1} , and $H_q = \text{HMTRX} = \text{diff}(\text{FMTRX}, t) \cdot \text{FMTRX}^{-1}$. In MACSyma, this final step requires no further explanation.

3. The Software Package

The program that follows accepts as input from a separate data file the following mathematical objects:

- 1) the dimension K of the tubal hypersurface;
- 2) the curvature tensor R of the ambient space restricted to the tangent space of the tube;
- 3) the second fundamental form A of the core submanifold; and,
- 4) the projection matrix onto the tangent space of the core.

The code with comments is given below. Comments in MACSyma are demarcated by "/" and "/" at the beginning and end, respectively. Despite extensive internal commenting, italicized remarks have been added to the body of the code, where the manipulations may be unclear to the reader.

The batch command accesses an executable text file containing the initial conditions, an example of which is given in the Section 4. The main program itself is invoked by a similar command, entered manually by the user during an interactive session in MACSyma. Note typical UNIX path name:

```
batch("/usr/mickey/steve/programs/macsyma/demo_data");
```



```

/*PART I: GENERATE LMTRX FROM R. */
kill:s$ /*Ensure 's' will behave as variable.*/
LMTRX:R$ /*Start with R.*/
for i:1 thru K do( /*for 1*/
  LMTRX[i,i]:s^2 + LMTRX[i,i]$ /*Add "s^2" to diagonal element of R.*/
/*end for 1*/
LNVMTX:LMTRX^^-1$ /*Record the inverse of LMTRX.*/

/*PART II: SOLVE FOR EACH OF K COLUMNS OF FMTRX. */
LFBAR:[]$ /*Initialize vector as empty list. It empties itself each cycle.*/
for j:1 thru K do( /*for 2*/
  BBAR:[]$ /*Initialize and empty out each cycle.*/
  FBAR:[]$ /*Initialize and empty out each cycle.*/

/*PART IIa: GENERATE Jth BBAR FROM Jth COLUMNS OF FZRO AND FPRMZRO. */
for i:1 thru K do( /*for 3*/

```

FZRO and FPRMZRO correspond to $F(0, X_q)$ and $\left. \frac{d}{dt}F(t, X_q) \right|_{t=0}$, respectively.

```

x:FZRO[i,j]*s+ FPRMZRO[i,j], /*Construct lth entry of the Jth BBAR.*/
BBAR:endcons(x,BBAR)), /*Tack it on to BBAR.*/ /*end for 3*/

```

```

/*PART IIc: SOLVE FOR Jth LFBAR BY MATRIX MULTIPLICATION. */
LFBAR:LNVMTX.BBAR,

```

```

/*PART IId: FIND INVERSE LAPLACE TRANSFORMS OF Jth LFBAR. */
for i:1 thru K do( /*for 4*/
  x:first(first(LFBAR)), /*Extract lth entry of Jth LFBAR.*/
  LFBAR:rest(LFBAR), /*Delete same from LFBAR.*/

```

The following command, ilt(x,s,t), is a built-in feature of MACSyma, which takes the Laplace transform given by the first argument, x, treats it as a function of the second argument, s, and returns the inverse Laplace transform in terms of the third argument, t:

```

y:ilt(x,s,t),          /*Take inverse Laplace transform of lth entry of Jth LFBAR.*/
FBAR:endcons(y,FBAR)), /*Place result in FBAR.*/ /*end for 4*/

```

```

/* PART IIe: TACK ON Jth FBAR TO FMTRX. */

```

Having generated FBAR in the previous loop, we affix it to FMTRX, using the addcol(FMTRX, FBAR) command, which tacks the second argument on to the first argument as its last column. Thus a K by K matrix becomes a K by K+1 matrix. One may see that FMTRX is indeed K by K to start with by turning to the initialization file in the next section.

```

FMTRX:addcol(FMTRX,FBAR), /*Add to end.*/

```

FMTRX is now a K by K+1 matrix. We delete the first column, which to this point contains only zeros, using the submatrix command:

```

FMTRX:submatrix(FMTRX,1))$ /*Axe the first.*/ /*end for 2*/

```

```

/*PART III: COMPUTE SECOND FUNDAMENTAL FORM VIA MULTIPLICATION. */
HMTRX:diff(FMTRX,t).(FMTRX^^-1)$

```

Note that computation of $FMTRX^{-1}$ and $d/dt(FMTRX)$ is implicit in the above statement. It suffices, with MACSyma, to leave it at that.

```

FMTRX; /*Display results.*/
HMTRX; /*Display results.*/

```

4. A Sample Run

Following is an example of an initialization file to be invoked by the command "batch("usr/mickey/steve/programs/macsyma/demo_data")" in the main program. We keep the dimension small due to space restrictions. In this example, the ambient space is two-dimensional complex hyperbolic space (of four real dimensions) so R is set equal to the appropriate curvature tensor, restricted to a tube about a one-dimensional geodesic hyperbolic space curve with constant curvature, so the second fundamental form of the core (A below) is set equal to the zero matrix.

```

K:3$                                /*Assignment of dimension is explicit.*/
R:matrix([-1,0,0],                  /*Curvature tensor of ambient space.*/
          [0,-1,0],
          [0,0,-4],
FZRO:matrix([1,0,0],
             [0,0,0],                /*Projection of the ambient tangent space.*/
             [0,0,1],               /*onto the submanifold tangent space.*/
A:zeromatrix(K,K)$                  /*Second fundamental form of tube core.*/
FPRMZRO:-A.FZRO + ident(K) - FZRO  /*Initial condition of F.*/
FMTRX:zeromatrix(K,K)$              /*Initialize as null matrix.*/
HMTRX:%$                            /*Set equal to previous matrix.*/

```

The output from a run of this data set yields:

$$FMTRX = \begin{bmatrix} \cosh(t) & 0 & 0 \\ 0 & \sinh(t) & 0 \\ 0 & 0 & \cosh(2t) \end{bmatrix} \text{ and } HMTRX = \begin{bmatrix} \tanh(t) & 0 & 0 \\ 0 & \coth(t) & 0 \\ 0 & 0 & 2\tanh(2t) \end{bmatrix}.$$

This run verifies that a tube about a geodesic hyperbolic curve in a complex hyperbolic space is a contact hypersurface.

Similar runs concerning tubes about totally geodesic submanifolds in complex space forms of higher dimensions have verified results found in [9] and [10] as well as in other well-known works concerning the extrinsic geometry of hypersurfaces in Riemannian manifolds. The authors are currently using this code for runs that calculate principle curvatures of tubes about totally umbilic submanifolds of complex space forms. An analysis of these runs will provide a basis to formulate conjectures regarding a geometric classification of hypersurfaces in complex hyperbolic and projective spaces in terms of tubes about totally umbilic submanifolds. This will effectively generalize the results of [9] and [10].

Support for this research was provided by two Idaho State Board Of Education Higher Education Research Grants. Specific support for this paper was provided by Idaho SBOE grant number 88-068.

5. Bibliography

- [1] T. Cecil and P. Ryan, Focal sets and real hypersurfaces in complex projective space, Trans. Amer. Math. Soc., Vol 269, #2, Feb. 1982
- [2] B. Y. Chen and L. Vanhecke, Differential geometry of geodesic spheres, J. Reine und Angewandte Math., Band 325, 1981, 28-67
- [3] A. Gray and L. Vanhecke, The volumes of tubes about curves in a Riemannian manifold, Proc. London Math. Soc., (3), 44(1982), 215-243
- [4] R. Howard, The Weingarten map of a tube, Personal Communication
- [5] S. Kobayashi and K. Nomizu, Foundations of Differential Geometry, Vol I & II, John Wiley and Sons, 1969
- [6] W. Klingenberg, Riemannian Geometry, De Gruyter Studies in Mathematics 1, Walter De Gruyter 1982
- [7] B. O'Neill, The fundamental equations of a submersion, Michigan Math. J., 13(1966)
- [8] L. Vanhecke and T. Willmore, Jacobi fields and geodesic spheres, Proc. Roy. Soc. Edin., 82A, 233-240, 1979
- [9] M. Vernon, Contact hypersurfaces of a complex hyperbolic space, Tohoku Mathematical Journal, vol. 39, no. 2, June, 1987
- [10] M. Vernon, Some families of isoparametric hypersurfaces and rigidity in a complex hyperbolic space, Transactions of the American Mathematical Society, December, 1988

COMPUTER ALGEBRA IN THE THEORY OF ORDINARY DIFFERENTIAL
EQUATIONS OF HALPHEN TYPE

V.P. Gerdt, N.A. Kostov

Laboratory of Computing Techniques and Automation

Joint Institute for Nuclear Research

Head Post Office, P.O. Box 79, 101000 Moscow, USSR

Abstract. We present an algorithm for solving linear differential equations in spectral parameter of Halphen type. The integrability condition of the pair of equations of Halphen type gives the large family of nonlinear differential equations of Lax-Novikov type. This algorithm is implemented on the basis of the computer algebra system REDUCE.

1. Introduction

We consider a linear differential equation in parameter λ (spectral parameter)

$$L\Phi = \left(\frac{d^m}{dx^m} + \sum_{j=1}^{m-1} p_j(x) \frac{d^{m-j}}{dx^{m-j}} \right) \Phi = \lambda \Phi, \quad (1)$$

where $p_j(x)$ are expressed in terms of elliptic functions. There are two classical problems [1]:

I) For which linear differential equation (1) is there a nonzero family of eigenfunctions $\Phi(x, \lambda, k, \alpha)$, depending smoothly on the eigenfunction parameter λ , such that Φ is meromorphic function on the algebraic curve

$$C_g: R(k, \alpha) = k^N + \sum_{j=1}^{N-1} k^{N-j} r_j(\alpha), \quad \lambda = \lambda(k, \alpha), \quad (2)$$

where $r_j(\alpha)$ are meromorphic functions on the elliptic curve

$C_1: (\wp'(\alpha), \wp(\alpha)); [\wp']^2 = 4\wp^3 - g_2\wp - g_3$; g_2, g_3 -elliptic invariants and $\wp$ is the Weierstrass $\wp$ -function. We may view C_g as an N -fold covering of the elliptic curve C_1 . Our conventions and notations concerning elliptic functions are those of Whittaker and Watson [2, Chap. XX]. This problem goes back to Halphen [1]. The solution of this problem was given in [1] only when $m=3,4$. The more general Halphen's problems of equivalence and classification of ordinary differential equations are recently solved by Bercovich [3] using the method of factorization of differential operators. These problems are closely related to the problem 1). As an illustration we give the following example [1,4]. Let us consider the third order equation

$$\left(\frac{d^3}{dx^3} + 3q_2(x)\frac{d}{dx} + 3q_2'(x)\right)\Psi = \lambda\Psi. \quad (3)$$

and introduce the so called first and second Halphen's absolute invariants $h=3q_2, l=3q_2'$. There is the following theorem:

Theorem 1. [1] The necessary and sufficient condition of integration of equation (3) in terms of elliptic functions is the algebraic relation

$$h^3 = (1-n^2)l^2/4 + \text{const.}, \quad n\text{-integer number, } n \not\equiv 0 \pmod{3}.$$

Then eq.(3) has the following canonical form (Halphen equation [5])

$$\left(\frac{d^3}{dx^3} + (1-n^2)\wp(x)\frac{d}{dx} + (1-n^2)\wp'(x)/2\right)\Psi = \lambda\Psi \quad (4)$$

where $\wp(x)$ is the Weierstrass $\wp$ -function.

Similar analysis is also possible when $m > 3$. Some particular results are known when $m=4,5$. Below we shall call this family of equations the Halphen type equations. There is another useful approach to generating the equations of Halphen type. Let us recall some results on the algebra of commuting differential operators (Burchall-Chaundy theory) [6] and corresponding completely integrable systems (the so called Lax-Novikov equations [7-9]). We start with two linear differential operators

$$L_1 = \frac{d^k}{dx^k} + \sum_{i=1}^{k-1} u_i(x) \frac{d^{i-k}}{dx^{i-k}}, \quad L_2 = \frac{d^l}{dx^l} + \sum_{j=1}^{l-1} v_j(x) \frac{d^{j-l}}{dx^{j-l}}. \quad (5)$$

Then we consider the following nonlinear system of differential equations in u_1, v_j

$$[L_1, L_2] = 0, \quad (6)$$

which is equivalent to a condition of integrability of the system

$$L_1 \Phi = \lambda \Phi, \quad L_2 \Phi = \mu \Phi. \quad (7)$$

Theorem 2. (Burchnall-Chaundy) [6], see also [10]. The equation (6) is equivalent to algebraic relation of the following type

$$Q(L_1, L_2) = 0,$$

where Q is a polynomial, such that

1) The eigenfunction $\Phi(x, \lambda)$ is the meromorphic function on the algebraic curve $Q(\lambda, \mu) = 0$.

2) The coefficients of $Q(\lambda, \mu)$ are the first integrals of eqs. (6) and are expressed as a differential polynomials of u_1, v_j .

3) When k, l are relatively prime, the space L_λ of Φ is one dimensional. The system (6) is completely integrable and solutions u_1, v_j are expressed in terms of Riemann θ -function.

Example 1.

Let us consider eqs. (6) when $k=2, l=2k-1$

$$[L, L_1] = 0,$$

where $L = \frac{d^2}{dx^2} + u(x)$, L_1 are operators, which are computed using a relation found by Lax

$$\frac{\partial}{\partial t} L = [L, L_1] \quad (8)$$

L and L_1 are called the Lax pair. The general expression for the L_1 is given in [11]

$$L_1 = 1/2 \sum_{k=1}^{\infty} \left[\frac{\partial}{\partial u} H_{k-1} \frac{d}{dx} - 1/2 X_{k-1} u \right] L^{1-k} \quad (9)$$

for instance,

$$L_1 = 1/4 \frac{d}{dx}, \quad L_2 = -1/4 \frac{d^3}{dx^3} + 3u/8 \frac{d}{dx} + 3u/16,$$

$$L_3 = 1/4 \frac{d^5}{dx^5} - 5u/8 \frac{d^3}{dx^3} - 15u' \frac{d^2}{dx^2} - 25uu''/32 \frac{d}{dx} + 15uu'/32 - 15u'''/64$$

L_1 is a differential operator of degree $2i-1$. For explicit expressions of H_k, X_k see [11]. Using Lamé potential $u = 1(1-i)\wp(x)$ and formula (9), we can obtain an useful example of Halphen type operators, for instance,

$$\begin{aligned} L_2 &= \frac{d^3}{dx^3} - 3\wp(x) \frac{d}{dx} - 3\wp'(x)/2 \\ L_3 &= \frac{d^5}{dx^5} - 15\wp(x) \frac{d^3}{dx^3} - 45/2 \wp'(x) \frac{d^2}{dx^2} - 75/2 \wp''(x) \frac{d}{dx} \\ &\quad - 45/2 \wp^2(x) \frac{d}{dx} + 45/2 \wp(x)\wp'(x) - 15/8 \wp'''(x). \end{aligned} \quad (10)$$

The same technique can be applied to the next two examples.

Example 2. Let us consider the generalized Lamé equation with potential $u = 2 \sum_{i=2}^{1(1-i)/2} \wp(x-x_i)$, where x_i are some constants, such that

$$\sum_{i=2}^{1(1-i)/2} \wp'(x_i - x_j) = 0, \quad i \neq j$$

Equation of such a type was introduced by Dubrovin and Novikov [12]. By similar technique as in example 1 it is possible to construct new examples of Halphen type operators.

Example 3. Recently Treibich and Verdier [13] found new elliptic potentials $u(x)$ of the following type

$$u(x) = 1(1-i)\wp(x) + 2 \sum_{k=1}^M g_k(g_k+1)(\wp(x-\omega_k) - e_k), \quad (11)$$

where $0 \leq g \leq i-1$, (for M, ω_k see [13]). Let us introduce some of them ($i=3$, [13])

$$\begin{aligned} u(x) &= 6\wp(x) + 2[\wp(x-\omega_k) - e_k], \quad k=1,2,3 \\ u(x) &= 6\wp(x) + 2[\wp(x-\omega_k) - e_k] + 2[\wp(x-\omega_1) - e_1], \quad 1 \neq k=1,2,3 \end{aligned} \quad (12)$$

The potentials (11) allows us to obtain new examples of Halphen type operators.

II) The second problem is to construct the family of eigenfunctions $\Psi(x, \lambda)$. The general form of this function goes back to Hermite [14] (in the case $n=2$) and to Halphen [1] (in the case

$n=3$). This function was improved by Krichever [15] in the theory of finite-gap integration method especially in the case of generalized Lamé equation (see example 2). He also proved that this function satisfies the Baker-Akhiezer (BA) axiomatics [16]. Below we describe an algorithm of construction of function $\Phi(x, \lambda)$, which we call Hermite-Halphen (HH) algorithm. The particular implementation of HH-algorithm on the computer algebra REDUCE is given. The mathematical background of this algorithm in more details is presented in [17].

In the paper [18] the following problem was studied:

III) For which linear ordinary differential operators
$$L = \sum_{j=0}^1 L_j(x) \frac{d^j}{dx^j}$$
 is there a non-zero family of eigenfunction $\Phi(x, \lambda)$ depending smoothly on the eigenfunction parameter λ , which is also an eigenfunctions of a linear differential operator $A = \sum_{r=0}^m A_r(\lambda) \frac{d^r}{d\lambda^r}$

$$A\Phi(x, \lambda) = \theta(x)\Phi(x, \lambda),$$

for an eigenvalue θ which is function of x . The complete answer was given in the case of Schrodinger operator. Most of the computations in this paper have been carried out using computer algebra system VAXSYMA. The relation between the problems II) and III) is under the progress.

In the papers [19,20] the Lamé equation was studied from the number theory point of view.

2. Notations

Let us introduce the functions

$$\Phi(x, \lambda) = \exp(kx) \{ a_0(\lambda, k, \alpha) \Phi(x, \alpha) + \sum_{j=1}^N a_j(\lambda, k, \alpha) \frac{d^j}{dx^j} \Phi(x, \alpha) \} \quad (13a)$$

$$\Phi(x, \lambda) = \exp(kx) \left\{ \sum_{i=1}^{1(1-1)/2} b_i \Phi(x-x_i, \alpha) \right\} \quad (\text{I.M. Krichever, [16]}) \quad (13b)$$

$$\Phi(x, \lambda) = \exp(kx) \left\{ \sum_{k=1}^M g_k(g_k+1) [a_{0k} \Phi(x-\omega_k, \alpha)] + \sum_{l=1}^{g_k-1} a_{lk} \frac{d^l}{dx^l} \Phi(x-\omega_k, \alpha) \right\} \quad (\text{V.Z. Enol'skii}) \quad (13c)$$

where

$$\Phi(x, \alpha) = \sigma(\alpha - x) / (\sigma(\alpha)\sigma(x)) \exp(\zeta(\alpha)x), \quad (14)$$

and σ, ζ are Weierstrass σ, ζ -functions [2].

Recall that the function (14) is a solution of Lamé equation

$$\left(\frac{d}{dx}\right)^2 - n(n+1)\wp(x))\Phi = \lambda\Phi, \quad (15)$$

when $n=2$. It is easy to see that the following Laurent series expansion of $\Phi(x, \alpha)$ hold

$$\Phi(x, \alpha) = 1/x + \sum_{j=1}^{\infty} \Phi_j x^j, \quad (16)$$

Inserting (16) into the (15, $n=2$), we have the following recurrent formula

$$[j(j-1)-2]\Phi_j - 2\wp_{j-1} - 2 \sum_{\substack{n,k \\ (n+k=j-2)}} \wp_n \Phi_k = \wp(\alpha)\Phi_{j-2}, \quad j > 2, \quad (17)$$

where we use the well known expansion of $\wp$ -function [2]

$$\wp(x) = 1/x^2 + \sum_{j=1}^{\infty} \wp_j x^{2j} \quad (18)$$

Some first Φ_j are

$$\Phi_1 = -1/2\wp(\alpha), \quad \Phi_2 = \wp'(\alpha)/6, \quad \Phi_3 = -\wp(\alpha)^2/8 + g/40, \quad \Phi_4 = \wp(\alpha)\wp(\alpha)'/60,$$

3. Algorithm

HERMITE-HALPHEN

Input:

ordinary differential equation of Halphen type.

Output:

function (13a), $a_1 = a_1(\lambda, k, \alpha)$,

N -fold covering on the torus C_1 (see (2)).

[1] Inserting (13a) into the ODE (1) and using the expansions (14), (16) generate the system of linear algebraic equations

$$G_m(a_1(\lambda, k, \alpha)) = 0, \quad m=1, 2, \dots, \quad (19)$$

by equating the coefficients at the $1/x^g$ ($g \in \mathbb{N}$).

[2] Solve the system (17) and write a_1 in terms of $k, \lambda, \wp(\alpha), \wp'(\alpha)$.

[3] By eliminating a_1 in (19) find the following system of nonlinear algebraic equations

$$F_1(k, \lambda, \wp(\alpha), \wp'(\alpha)) = 0, \quad F_2(k, \lambda, \wp(\alpha), \wp'(\alpha)) = 0 \quad (20)$$

where $[\wp'(\alpha)]^2 = 4\wp^3(\alpha) - g_2\wp(\alpha) - g_3$.

[4] Solve the nonlinear eqs. (20) with respect to k, λ

$$k = k(\wp'(\alpha), \wp(\alpha)), \quad \lambda = \lambda(\wp'(\alpha), \wp(\alpha)),$$

using some appropriate technique, for example, usual elimination method [21] or Buchberger's approach, based on construction of the Groebner basis [22]. Find the N -fold covering (2).

We have implemented the HH-algorithm on the basis of the system REDUCE 3.2 and the program characteristics are the following:

- computer ES-1061 (IBM 370),
- high speed storage required, depends on the problem, minimum 600K,
- number of lines 210.

In the last step 4 we use the method of elimination. We test our program with Lamé equation $n=2+9$, and Halphen equation ($n=4, 5$). More general implementation is possible using function (13b) (see example 2) and also function (13c) (see example 3). An open problem is the generation of all equations of Halphen type.

4. Examples

Example 4. This example illustrates the basic steps in the realization of Hermite-Halphen algorithm (HH-algorithm) described above.

Let us consider the Lamé equation (15, n=4), M=3.

[1] Inserting the function (3) into the Lamé equation we obtain

$$\begin{aligned} [1/x^5] \quad & 3ka_3 - a_2 = 0, \\ [1/x^4] \quad & 3(\lambda - k^2)a_3 - 6ka_2 + 7a_1 = 0, \\ [1/x^3] \quad & (\lambda - k^2)a_2 - 2ka_1 + 9a_0 = 0, \\ [1/x^2] \quad & 3(5\wp^2 + g_2)a_3 - 20/3 \wp'a_2 + (10\wp - k^2 + \lambda)a_1 - 2ka_0 = 0, \\ [1/x] \quad & -8\wp\wp'a_3 + 5(3\wp^2 - g_2) - 20/3 \wp'a_1 + (10\wp + k^2 - \lambda)a_0 = 0. \end{aligned}$$

[2] The solution of the system (17) is

$$a_3 = 1, a_2 = 3k, a_1 = 3k^2 - 3/7\lambda, a_0 = k(k^2 - 3/7\lambda)$$

[3] After simple manipulations we obtain

$$F_1 = 35k^4 - k^2(30\lambda + 210\wp) + 140\wp'k + 3\lambda^2 - 105\wp^2 - 21g_2 + 30\wp\lambda = 0,$$

$$F_2 = (5\lambda - 140\wp)k^3 + 210\wp'k^2 + (-3\lambda + 45\wp\lambda + 126g_2 - 420\wp^2)\lambda + 70\wp\wp' - 25\lambda\wp' = 0.$$

[4] Using the method of elimination for the 10-fold covering of the elliptic curve C_1 we have

$$\begin{aligned} & k^{10} - 45\wp k^8 + 120\wp'k^7 + (-630\wp^2 + 399/4 g_2)k^6 + 504\wp\wp'k^5 + \\ & (-1050\wp^3 + 1725/4 g_3 + 735/4 \wp g_2)k^4 + (360\wp^2\wp' - 165\wp'g_2)k^3 + \\ & (-189/4 g_2^2 - 315\wp^4 + 2205/4 \wp^2 g_2 - 855/2 \wp g_3)k^2 + \\ & (-163\wp\wp'g_2 + 125\wp'g_3 + 40\wp^3\wp')k + \\ & -9\wp^5 - 75/4 \wp g_2^2 - 75/4 g_2 g_3 + 9/4 \wp^2 g_3 + 309/4 \wp^3. \end{aligned}$$

Example 5.

Let us consider the Halphen equation (4) when $n = 4$

$$\left(\frac{d}{dx}\right)^3 - 15\wp(x)\frac{d}{dx} + 15/2 \wp'(x))\Phi = \lambda\Phi. \quad (21)$$

Assume that Φ has the following form

$$\Phi = \exp(kx)(a_0\Phi(x, \alpha) + a_1 \frac{d}{dx} \Phi(x, \alpha) + a_2 \frac{d^2}{dx^2} \Phi(x, \alpha)) \quad (22)$$

Inserting (22) in (21) we have

$$\begin{aligned} [1/x^5] \quad & 2ka_2 - a_1 = 0, \\ [1/x^4] \quad & k^2a_2 - a_0 = 0, \\ [1/x^3] \quad & (2k^3 + 5\wp' - \lambda)a_2 + (6k^2 - 15\wp/2)a_1 - 9ka_0 = 0, \end{aligned}$$

$$\begin{aligned}
[1/x^2] \quad & (-5k\wp + 3/5 g_2) a_2 + (-k^3 + 15/2 k\wp + \lambda/2) a_1 - 3k^2 a_0 = 0, \\
[1/x] \quad & (45/4 k\wp^2 - 3\wp\wp') a_2 - (5k\wp' + 45/8 \wp^2) a_1 + \\
& (k^3 - 5/2 \wp' + 15/2 \wp k - \lambda/2) a_0 = 0.
\end{aligned}$$

Step by step elimination of a_1 gives

$$\begin{aligned}
a_0 = k^2, \quad a_1 = 2k, \quad a_2 = 1, \quad \text{eq. } [1/x^3] \Rightarrow \lambda = 5(k^3 - 5k\wp + \wp'), \quad \text{eq. } [1/x^2] \Rightarrow g_2 = 0, \\
\text{eq. } [1/x] \Rightarrow k^5 - 25/2 k^2 \wp' + 45/2 \wp^2 k - 3\wp\wp' + 15/2 \wp k^3 - \lambda/2 k^2 = 0.
\end{aligned}$$

Acknowledgements. We are very much indebted to V.Z. Enol'skii for many suggestions and discussions. We also would like to thank L.M. Bercovich for the useful information about the Halphen's problem and Halphen's classification when $n=4,5$.

References

- [1] Halphen G.H. *Memoire sur la reduction des equations differentielles lineaires aux formes integrales*. Mem. pres. l'Acad. des Sci. de France, 1884, 28, No. 1, p. 1.
- [2] Whittaker E.T., Watson G.N. *A course of modern analysis*. Cambridge, Cambridge University Press, 1927.
- [3] Bercovich L.M. *Canonical forms of ordinary differential equations*. Arc. Math. (Brno, CSSR), 1988, 24, No. 1, p. 25.
- [4] Bercovich L.M. *Absolute invariants and Korteweg de Vries equation*. In: Group theoretical methods in physics, Proc. of the third seminar (Yurmala, 1985), Moscow, Nauka, 1986 (in Russian).
- [5] Kamke E. *Differentialgleichungen: Lösungsmethoden und Lösungen*. Chelsea Publishing Co., New York, 1959.
- [6] Burchinal J.L., Chaundy T.W. *Commutative ordinary differential equations*. Proc. London Math. Soc., 1923, 21, p. 420.
- [7] Novikov S.P. *The periodic problem for the Korteweg de Vries equation*. Funct. Anal. & Appl., 1975, 8, p. 236 (in Russian).
- [8] Lax P. *Periodic solution of the KdV equation*. Comm. Pure & Appl. Math., 1975, 28, p. 141.
- [9] Krichever I.M. *The method of algebraic geometry in the theory of nonlinear equations*. Usp. Mat. Nauk, 1977, 32, p. 185 (in Russian).
- [10] Chudnovsky D.V. *The generalized Riemann-Hilbert problem and the spectral interpretation*. In: Nonlinear Evolution Equations and Dynamical Systems. Lect. Notes in Phys., 120, Springer, New York, 1980.
- [11] McKean H.P., van Moerbeke P. *The spectrum of Hill's equation*. Invent. Math., 1975, 30, p. 217.
- [12] Airault H., McKean H.P., Moser J. *Rational and elliptic solutions of the KdV equation and a related many-body problem*. Comm. Pure & Appl. Math., 1977, 30, 95.
- [13] Verdier J.L. *New elliptic solitons*. Preprint, 1987, Paris.

- [14] Hermite C. *Oeuvres*. Vol. 3, Paris, Gauthier-Villars, 1912.
- [15] Krichever I.M. *Elliptic solutions of Kadomtsev-Petviashvili equation and integrable particle systems*. *Funct. Anal. & Appl.*, 1980, 14, p. 45 (in Russian).
- [16] Dubrovin B.A., Matveev V.B., Novikov S.P. *Non-linear equations of KdV type, finite-zone linear operators and abelian varieties*. *Russ. Math. Surveys*, 1976, 31, p. 59.
- [17] Belokolos E.D., Enols'kii V.Z., Bobenko A.I., Matveev V.B. *Algebro-geometrical principles of superposition of finite-gap solutions of integrable nonlinear equations*. *Usp. Mat. Nauk*, 1986, 41, 3 (in Russian);
Belokolos E.D., Enol'skii V.Z. *Reduction of theta-functions and Humbert finite-gap potentials of Lamé, Treibich-Verdler, etc.* Preprint IMP-88-051988, Kiev, 1988.
- [18] Duistermaat J.J., Grunbaum F.A. *Differential equations in spectral parameter*. *Comm. Math. Phys.*, 1986, 103, p. 177.
- [19] Chudnovsky D.V., Chudnovsky G.V. *Remark on the nature of the spectrum of Lamé equation*. *Lett. Nuovo Cim.*, 1980, 29, p. 545.
- [20] Chudnovsky D.V., Chudnovsky G.V. *Applications of Padé approximation to the Grothendieck conjecture on linear differential equations*. *Lect. Notes in Math.*, vol. 1135, Number theory, Springer, 1986.
- [21] Moses J. *Solution of a system of polynomial equations by elimination*. *Comm. ACM*, 1966, 9, p. 634.
- [22] Buchberger B. *Groebner basis: a method in the symbolic mathematics*. In: *Progress, directions and open problems in multidimensional system theory* (ed. Bose N.K.), Dordrecht, Reidel, 1985, p. 184.

SYMBOLIC DERIVATION OF EQUATIONS FOR MIXED FORMULATION IN FINITE ELEMENT ANALYSIS

H. Q. Tan[‡]

*Department of Mathematical Sciences
The University of Akron
Akron, Ohio 44325*

ABSTRACT One important application area of symbolic manipulation systems is to use them as preprocessors to generate numerical code for target machines, thus facilitating the tedious pre-processing involved in numerical computing, particularly in Finite Element Method (FEM). However, pre-processing the given formulation often generates unacceptable results due to the number of derivation steps involved as well as the exponential growth of expressions produced. Presented in this paper are some of the procedures and techniques we have developed for efficient derivation of element equations in FEM. Examples are given for applications of those procedures and techniques in the derivation of element equations for mixed formulations.

1. INTRODUCTION

The finite element method has many applications in aerospace, civil, mechanical and other engineering disciplines. Large software packages, e. g. NASTRAN [16] and NFAP [2], exist for the analysis of a wide-range of engineering problems. Although these packages are written in a modular form to accommodate program changes, extensive derivation of finite equation and manual coding are still necessary for incorporation of new analysis features. In this regard, utilization of symbolic manipulations for derivation of finite element equations, including element matrices, constitutive relations and other related numerical equations as well as automatic code generation are most attractive.

The potential benefit of such an approach are clearly indicated [1,4,5,6]. However, unintelligent application of symbolic manipulations will have limited values. One major obstacle is the exponential growth of derived expressions, which require significant storage space and computer time. Our research on the integration of symbolic and numerical computations has led to the construction of software automation tools, named FINGER (FINite element code GEnerator), LAXTOR (material mATriX generaTOR) and SDICE (Symbolic DerIvation of Constitutive Equations) that automate the derivation of formulas in finite element analysis, constitutive model research and generation of code for numerical calculations. Current capacities of these softwares

[‡] Work reported herein has been supported in part by the U. S. National Aeronautics and Space Administration Lewis Research Center under Grant No. NAG 3-872 and by the National Science Foundation under Grant No. EET 87-14628.

include:

- (1) derivation of element equations or matrices for the development of finite element method (FINGER)[7,9,10,14],
- (2) derivation of mathematical formulas for new constitutive material models (SDICE)[8,12],
- (3) derivation of material matrix pertaining to finite element calculations (LAX-TOR)[3,11], and
- (4) based on the symbolic computations, automatic generation of FORTRAN code in a form specified by the user for the equations or matrices based on the symbolic computations.

Practical problems in finite element analysis and constitutive model research involve large expressions. Without design and implementation of problem-oriented procedures and techniques, the formula derivation can become time consuming and the generated equations and code will be lengthy and inefficient. Therefore, the major problems have to be addressed are:

- (1) the derivation of symbolic formulas must be made efficient and resourceful to handle the large expressions associated with practical problems, and
- (2) techniques must be employed to reduce the inefficiencies that are usually associated with automatically generated code.

In the next few sections, we will discuss, through the application of FINGER in the derivation of plane stress and plane strain element equations of mixed formulation as an example, the features of our approach to those problems.

2. EQUATION DERIVATIONS

Shell elements have many applications in structural analysis. Of all the shell elements described in the literature thus far, the degenerated shell elements based on the isoparametric displacement formulation and the Mindlin/Reissner plate theory, which accounts for transverse shear deformations, appear to be most popular for both linear and nonlinear shell analysis. However, the original form of degenerated shell elements has been found to exhibit overly stiff behavior in bending for thin shells, due mainly to the severe constraints introduced by the conditions of vanishing membrane and/or shear strain components. This has been often referred to as membrane/shear locking effect which placed a severe limitation on the application range of such elements. A promising approach to overcome the locking effect is the hybrid or hybrid/mixed method, in which the interpolation functions for displacements and stresses are independently but consistently assumed, in order to control parasitic shear or membrane terms. The success of hybrid or hybrid/mixed formulation in a large variety of constrained-media applications, such as thin plate and shell problems and incompressible materials, has been demonstrated in various publications. Indeed, in view of its great potential, more extensive use of this approach is anticipated in the near future.

In mixed formulation for shell element, for example, the element stiffness K is given as:

$$K = G^T (H^{-1}G) \quad (1)$$

and the element stresses are obtained from

$$\sigma = P (H^{-1}G) q$$

in which

$$H = \int P^T S P \, dv \quad (2)$$

$$G = \int P^T B \, dv \quad (3)$$

where S is elastic compliance matrix, which may have different forms for plane stress and plane strain. for example, in two-dimension case, S is given as

$$\frac{1}{E} \begin{bmatrix} 1-\mu & 0 \\ -\mu & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 2(1+\mu) \end{bmatrix}.$$

When dealing with plane strain, we use the same S except the definitions of Young's modulus E and Poisson's ratio μ . The B is strain-displacement matrix and q is nodal displacement vector.

The approach we adopted is to derive the $[H]$ and $[G]$ matrices in symbolic forms, and then to compute $[K]$ matrix numerically.

The expressions of $[B]$ and $[P]$ matrices have direct effect on the forming of $[G]$ and $[H]$ matrices. In order to derive these two matrices, we have developed some procedures tailored just for this applications, i.e.

- (1) defining of functions to represent modularized computations,
- (2) labeling common expressions to avoid expression growth,
- (3) substituting labels during intermediate steps by pattern match procedures to represent the lengthy expressions,
- (4) simplifications through several derivation steps by specially designed routines, and
- (5) utilization of symmetric relations in the given problem.

When deriving $[B]$ matrix, the strain-displacement relation is abbreviated as

$$(E) = [L](U)$$

where (E) is strain vector, $[L]$ is strain-displacement differential operator, (U) is nodal displacement vector. For example in two-dimension case, we have,

$$E = \begin{bmatrix} \xi_x \\ \xi_y \\ \gamma_{xy} \end{bmatrix},$$

$$[L] = \begin{bmatrix} \frac{d}{dx} & 0 \\ 0 & \frac{d}{dy} \\ \frac{d}{dy} & \frac{d}{dx} \end{bmatrix},$$

$$U = \begin{bmatrix} U_x \\ U_y \end{bmatrix}.$$

where

$$U_x = \sum_{i=1}^n N_i' u_x^i$$

$$U_y = \sum_{i=1}^n N_i' u_y^i$$

in which N_i' is the coordinate shape functions for $i = 1, \dots, n$ and n is the number of element nodes.

Let r and s be the natural coordinates and

$$NP' = \begin{bmatrix} N_r' \\ N_s' \end{bmatrix},$$

where the subscripts indicate partial differentiation. Then we have

$$J = NP' \cdot [X, Y]$$

where J is Jacobian matrix, $X = [x_1, \dots, x_n]$ and $Y = [y_1, \dots, y_n]$. To capture symmetry and generate more efficient code later, the function

$$jrow(var) := [N_s' \cdot var, -N_r' \cdot var]$$

is used in FINGER. This allows J^{-1} to be expressed as

$$J^{-1} = \frac{1}{detj} \begin{bmatrix} jrow(Y) \\ -jrow(X) \end{bmatrix},$$

where $detj = \det(J)$.

Let $NPT' = transpose(NP')$ and let NPT_i' be the i th column of NPT' , now the $[B]$ matrix can be represented as

$$\begin{bmatrix} jrow(Y) \cdot NPT_i' & 0 \\ 0 & -jrow(X) \cdot NPT_i' \\ -jrow(X) \cdot NPT_i' & jrow(Y) \cdot NPT_i' \end{bmatrix}$$

for $i = 1, 2, \dots, n$.

In deriving shape functions for stresses [P] by FINGER in two-dimension four-node case, we first define a contravariant stress tensor in the (r, s) domain,

$$\tau^{rs} = \begin{bmatrix} \tau^{11} & \tau^{12} \\ \tau^{21} & \tau^{22} \end{bmatrix}$$

and in the physical (x, y) domain,

$$\sigma^{ij} = \begin{bmatrix} \sigma^{11} & \sigma^{12} \\ \sigma^{21} & \sigma^{22} \end{bmatrix}$$

To account for the effect of distorted element geometry, we perform

$$\begin{aligned} \sigma^{ij} &= \left[\frac{\partial x^i}{\partial r^m} \right]_o \left[\frac{\partial x^j}{\partial r^n} \right]_o \tau^{mn} \\ &= \bar{J}_{im}^o \bar{J}_{jn}^o \tau^{mn} \end{aligned}$$

where $\bar{J}^o$ is obtained by evaluating transpose Jacobian matrix at $r = 0$ and $s = 0$ and we introduce labels to represent its entries:

$$J_o = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}. \quad (4)$$

Now, we have

$$\sigma^{11} = (\bar{J}_{11}^o)^2 \tau^{11} + (\bar{J}_{12}^o \bar{J}_{11}^o + \bar{J}_{11}^o \bar{J}_{12}^o) \tau^{12} + \bar{J}_{12}^o \bar{J}_{12}^o \tau^{22} \quad (5)$$

$$\sigma^{22} = \bar{J}_{21}^o \bar{J}_{21}^o \tau^{11} + (\bar{J}_{22}^o \bar{J}_{21}^o + \bar{J}_{21}^o \bar{J}_{22}^o) \tau^{12} + \bar{J}_{22}^o \bar{J}_{22}^o \tau^{22}$$

$$\sigma^{12} = \bar{J}_{11}^o \bar{J}_{21}^o \tau^{11} + (\bar{J}_{12}^o \bar{J}_{21}^o + \bar{J}_{11}^o \bar{J}_{22}^o) \tau^{12} + \bar{J}_{12}^o \bar{J}_{22}^o \tau^{22}.$$

By substituting the labels defined in (4) for Jacobian matrix into (5), we have

$$\sigma^{11} = a_{11}^2 \tau^{11} + 2a_{21}a_{11} \tau^{12} + a_{21}^2 \tau^{22} \quad (6)$$

$$\sigma^{22} = a_{12}^2 \tau^{11} + 2a_{12}a_{22} \tau^{12} + a_{22}^2 \tau^{22}$$

$$\sigma^{12} = a_{11}a_{12} \tau^{11} + (a_{12}a_{21} + a_{11}a_{22}) \tau^{12} + a_{21}a_{22} \tau^{22}$$

If we assume the stress functions

$$\tau^{11} = \beta_1 + \beta_2 s$$

$$\tau^{22} = \beta_3 + \beta_4 r$$

$$\tau^{12} = \beta_5,$$

then by substituting them back into (6), grouping terms with respect to r and s and introducing labels again to represent these terms which are only constants in integration (see next section), we finally have,

$$\sigma^{11} = \bar{\beta}_1 + (a_{11}^2 s) \beta_2 + (a_{21}^2 r) \beta_4 \quad (7)$$

$$\sigma^{22} = \bar{\beta}_3 + (a_{12}^2 s) \beta_2 + (a_{22}^2 r) \beta_4$$

$$\sigma^{12} = \bar{\beta}_5 + (a_{11} a_{12} s) \beta_2 + (a_{21} a_{22} r) \beta_4.$$

Since

$$\sigma = P \cdot \beta,$$

by dropping bars in (7) we eventually get the expressions for [P] as

$$P = \begin{bmatrix} 1 & a_{11}^2 s & 0 & a_{21}^2 r & 0 \\ 0 & a_{12}^2 s & 1 & a_{22}^2 r & 0 \\ 0 & a_{11} a_{12} s & 0 & a_{21} a_{22} r & 1 \end{bmatrix} \quad (8)$$

and

$$\beta^T = [\beta_1 \ \beta_2 \ \beta_3 \ \beta_4 \ \beta_5]$$

3. COMPUTATIONAL PROCEDURES AND SYMBOLIC INTEGRATION

In deriving formulas, special attention is paid to the identification of common sub-structures and/or expressions as the computation proceeds, not afterward. Without these steps, the final result of the numerical programs cannot be optimized to an acceptable level. For this reason, it requires the design and implementation of problem-oriented procedures that are tailored just for the task of deriving equations.

The [H] matrix (2) is symmetric. Therefore, only those unique integrands are computed. Let us examine the procedure for [H] derivation. From (8), if we define a matrix entry-wise multiplication "*", it is obvious that the [P] matrix can be

represented as

$$P = P' * P^*$$

where

$$P^* = \begin{bmatrix} 1 & a_{11}^2 & 0 & a_{21}^2 & 0 \\ 0 & a_{12}^2 & 1 & a_{22}^2 & 0 \\ 0 & a_{11}a_{12} & 0 & a_{21}a_{22} & 1 \end{bmatrix},$$

$$P' = \begin{bmatrix} 1 & s & 1 & r & 1 \end{bmatrix}.$$

And the transpose of [P] can be represented accordingly.

By this definition, [H] matrix now becomes

$$H = \int_{-1}^1 \int_{-1}^1 P^{*T} * P^T \cdot S \cdot P^* * P' \det j \, dr ds,$$

and by rearranging it, we have

$$H = \int_{-1}^1 \int_{-1}^1 P^{*T} \cdot S \cdot P^* * P'^T \cdot P' \det j \, dr ds,$$

where "*" is the same entry-wise matrix multiplication defined before.

Now, the [H] matrix is the entry-wise multiplication of

$$H^* * H',$$

where

$$H' = \int_{-1}^1 \int_{-1}^1 P^T \cdot P' \det j \, dr ds$$

$$H^* = P^{*T} \cdot S \cdot P^*.$$

So, instead of H, we only need to integrate H' and can keep the expression as well as the code size small.

For [G] matrix, the integrands involve the strain-nodal displacement matrix and it is defined in symbolic form by FINGER (see previous section). In order to catch the symmetric relation, we first use the above defined P^{*T} . For example, (3) can be expressed as,

$$G = P^{*T} \cdot G^*$$

where

$$G^* = \int_{-1}^1 \int_{-1}^1 (1 \text{ or } r \text{ or } s) B^* \, dr ds$$

and

$$B^* = B \det j.$$

Now, only those terms matching the pattern

$$(1 \text{ or } r \text{ or } s) (jrow(var) \cdot NPT_i')$$

need to be integrated instead of the whole matrix.

These procedures certainly simplify the equations for code generation and more efficient code can be produced.

Previous work in employing symbolic systems such as MACSYMA [15] for finite element computation was based on user-level programs which do not contain any intelligence in the manner that symbolic derivations are being carried out. As a result, the ability of handling realistic cases in practice is limited.

Let us take the the computation of of H' as an example. For element in three-dimension eight-node case the H' is an 18×18 matrix and each entry of the matrix is a polynomial of 512 terms ($i = n^d$, where i = number of terms in the polynomial, n = number of nodes, d = element dimension). If we were going to integrate this matrix, the expression size is so big that it cannot be accepted.

During the top-level design of the system special attention is paid to identify the symmetry in the given problem. The integrands for our application only contain r 's, s 's, t 's, x 's, y 's and z 's (in three-dimension case), or

$$H' = \int_{-1}^1 f(r, s, t) dr ds dt.$$

If we rearrange the above form into

$$\int f(r, s, t) dr ds dt = \int u(r) dr \int v(s) ds \int w(t) dt,$$

and since it is symmetric, we can integrate one function g with dummy variable, say u , instead of the entire expression. And the final result will be produced by pattern match procedures. So, the approach to compute H' is to compute, as the first step,

$$\int_{-1}^1 g(u) du.$$

Furthermore, the $detj$ in (4) is formed by the partial differentiation of shape functions with respect to the local coordinates r, s, t , and the polynomial only contains combinations of the terms in the forms of

$$(r+1), (r-1), (s+1), (s-1), (t+1), (t-1).$$

Therefore, the expression for the function $g(u)$ can only be

$$\int_{-1}^1 [(+ \text{ or } -) (u+1) (+ \text{ or } -) (u-1)]^i u^j du \quad (9)$$

With this relation found, we can proceed to integrate the expression and compute H' with pattern match procedures.

However, computation by use of the integration package of MACSYMA gives unsatisfactory performance (very slow). And by further examination of the expression of (9), we can find that the result of integration can only be one of the forms in

$$2 \frac{1}{j+1}, \text{ for } i = 0, j = 0, 2, 4, \dots \quad (10)$$

$$0, \text{ for } i = 0, j = 0, 1, 3, 5, \dots \quad (11)$$

$$2 \left[(+ \text{ or } -) \frac{1}{j+1} (+ \text{ or } -) \frac{1}{j+3} \right], \text{ for } i = 1, j = 0, 2, 4, \dots \quad (12)$$

$$2 \left[(+ \text{ or } -) \frac{1}{j+2} (+ \text{ or } -) \frac{1}{j+2} \right], \text{ for } i = 1, j = 1, 3, 5, \dots \quad (13)$$

Therefore, we do not need to integrate the polynomial at all, and a pattern match procedure can scan the polynomial and replace the terms in the polynomial with the above numerical values.

The term $detj$ is formed by the function $jrow (var)$ (for two-dimension case) or $jrow (var1, var2)$ (in three-dimension case). Since the function $jrow$ is defined as dot product of the partial differentiation of shape functions with respect to the local coordinates r, s, t and the global coordinates x, y, z , then $jrow (X, Y)$ is related to $jrow (Y, Z)$, $jrow (Z, X)$, $jrow (Y, X)$, $jrow (Z, Y)$ and $jrow (X, Z)$ by symmetry, even though these six expressions cannot be regarded as identical. However, if we use $jrow (a, b)$ with dummy variables a , and b , then all six functions are identical in the sense of computation results. In this way, we can cut the computation by five sixth.

We think that discovering symmetric relations plays a vital rule in the code optimization process and thus helps generate more efficient code.

4. CONCLUSION

We have discussed the use of a symbolic computation system to derive equations in finite element analysis. Even though, only two-dimension case is given here as an example, the equation derivation procedures can be applied to three-dimension case without modification. It is hoped that the approach discussed here may find other applications.

5. REFERENCES

- [1] Cecchi, M. M. and Lami, C. "Automatic generation of stiffness matrices for finite element analysis", *Int. J. Num. Meth. Engng* 11, 1977, pp.396-400.
- [2] Chang, T. Y. "NFAP - A Nonlinear Finite Element Program (Version 85.1)", Department of Civil Engineering, University of Akron, Akron Ohio, 1985.
- [3] Chang, T. Y., Saleeb, A. F., Wang, P. S. and Tan, H. Q. "On the Symbolic Manipulation and Code Generation for Elasto-Plastic Material Matrices", *Engineering with Computers-- An International Journal for Computer-Aided Mechanical and Structural Engineering*, Springer-Verlag, New York, 1986.
- [4] Korncoff, A. R. and Fenves, S. J. "Symbolic generation of finite element stiffness matrices", *Comput. Structures*, 10, pp. 119-124, 1979.
- [5] Noor, A. K. and Andersen, C. M. "Computerized Symbolic Manipulation in Non-linear Finite Element Analysis", *Comput. Structures* 1 pp. 9-40 1981.
- [6] Noor, A. K. and Andersen, C. M. "Computerized symbolic Manipulation in structural mechanics-progress and potential", *Comput. Structures* 10, pp. 95-118, 1977.
- [7] Tan H. Q. "Automatic Equation Derivation and Code Generation in Engineering Analysis", *Proceedings of IEEE International Computer Science Conference '88 (Artificial Intelligence: Theory and Practice)*, Hong Kong, 1988.
- [8] Tan, H. Q. "Automatic Derivation of Constitutive Equations", *Proceedings of IMACS International Conference on Expert Systems for Numerical Computing*, West Lafayette, Indiana, 1988.
- [9] Tan, H. Q., Chang, T. Y. and Wang, P. S., "Symbolic Derivation of Finite Element Equations and Automatic Code Generation", *ROBEXS-87 Proceedings*, Pittsburgh, U. S. A., 1987.
- [10] Tan, H. Q. "Integration of Symbolic and Numerical Computations in Finite Element Analysis", *Proceedings of 12th IMACS World Congress on Scientific Computation*, Paris, France, 1988.
- [11] Tan, H. Q. "Symbolic Derivation of Material Property Matrices in Finite Element Analysis", *Proceedings of ASME Winter Annual Meeting*, Chicago, 1988.
- [12] Tan, H. Q., Dong, X., Arnold, S. M. "Symbolic Derivation of Constitutive Equations", *Proceedings of ASME Winter Annual Meeting*, Chicago, 1988.
- [13] Wang, P. S., Tan, H. Q., Saleeb, A. F. and Chang, T. Y. "Code Generation for Hybrid Mixed Mode formulation in Finite Element Analysis", *Proceedings, SYM-SAC'86*, Toronto, Canada, 1986.
- [14] Wang, P. S. "FINGER: A Symbolic System for Automatic Generation of Numerical Programs in Finite Element Analysis", *J. Symbolic Computation*, (1986) 2, Academic Press Inc. (London) Ltd. 1986.
- [15] "MACSYMA Reference Manual", version 10, The MATHLAB Group, Laboratory for Computer Science, MIT, 1984
- [16] "MSC-NASTRAN Application Manual - A General Purpose Finite Element Program", MacNeal-Schwendler Corporation, Los Angeles, California.

Semantics in Algebraic Computation

D. L. Rector

Department of Mathematics
University of California at Irvine
drector@orion.cf.uci.edu

I am interested in symbolic computation for theoretical research in algebraic topology. Most algebraic computations in topology are hand calculations; that is, they can be accomplished by the researcher in times ranging from hours to weeks, and they are aimed at discovering general patterns rather than producing specific formulas understood in advance. Furthermore, the range of algebraic constructs used in such calculations is very wide. Consequently, the most important design considerations for symbolic manipulation tools in topology are

- ease of construction of new symbolic structures, and
- deftness in control of algebraic simplification.

The researcher must have fine control of whether a subexpression in a formula is reduced to a "simpler" form or left in some factored condition, for simplification usually results in loss of information that may be very expensive to recover—indeed, the mere multiplication of two integers is, in general, so expensive to undo, that the process is the basis of the most common public key cryptosystem.

One way to provide a user with fine control of an algebraic rewrite system is to attach control information governing choice of rewrite rules and heuristics to the semantic properties of subexpressions. For example, one would like to be able to say to a simplifier

"reduce trigonometric functions to sines and cosines, multiply out in-

tegers, and leave polynomials in X and Y in factored form."

A computer algebra system with some semantic capability is *Scratchpad II* developed by Richard Jenks, *et. al.* at IBM Watson Research Laboratory [5]. *Scratchpad II* incorporates parameterized data type constructors in the style of *CLU* which greatly enhance reusability of algorithmic code. For example, a polynomial ring constructor $Pol[R]$ will, given a data type R provided with operations satisfying the axioms of a ring, create executable code for the data type of polynomials with coefficients in R without any recoding by a user. This is done without significant sacrifice in performance. Jenks introduced the notion of the *category* of a data type to provide a database management facility for keeping track of the properties satisfied by data type constructors. In addition, *Scratchpad II* contains an automatic type inference facility in its interactive parser to relieve the user of much of the chore of specifying the data types of input expressions. This facility is primarily oriented toward commutative ring theory.

A second system of interest is *Views* developed by Abdali, Cherry, and Soiffer at Tektronix Labs [1]. That system, which also incorporates the notion of a category, extended the object oriented programming facilities of *Smalltalk* to allow objects to be viewed in multiple ways as mathematical objects. *Smalltalk's* object system allows a user great flexibility in reuse of code by allowing objects to access code from different languages.

My aim is to synthesize and abstract the

semantic capabilities of **Scratchpad II** and **Views** to provide a simple and flexible semantic analysis tool that can be used by a mathematician to encode the semantic knowledge of his own computational world. I intend it to provide

- a semantic control language for an algebraic simplification system,
- a type inference facility, and
- a structure to catalog and access computational algorithms.

In particular, this system will, unlike **Scratchpad II**, separate the semantic notion of an algebraic domain from any specific data representation or any data type construction mechanism in an underlying computer language.

The ideas presented here are somewhat preliminary. They are based on a mathematician's naive faith that a powerful and easy to use formalism grows from an elegant theoretical understanding—in this case, *algebraic specification*. I will set forth the current state of the art in algebraic specification to show how it provides an adequate theoretical and computational framework to characterize the mathematical notions of category and functor. I will then provide a higher level structure, an *algebraic theory* to organize categories and functors into a domain for automatic type inferencing. Finally, I will comment briefly on how the notion of a *model morphism* can permit the smooth integration of declarative and algorithmic definitions of algebraic domains.

I apologize in advance to experts in algebraic specification for occasional use of non-standard terminology and unintentional appropriation of ideas. I have tended to use mathematical terminology, since mathematics is the intended area of application, and have independently reinvented some notions known previously to the algebraic specification community. I believe that my notion of algebraic theory, in the sense of a domain for automatic type inference, is new.

I would like to thank Y. V. Srinivas and Ira Baxter, students of Peter Freeman, for

teaching me theoretical software engineering and for many useful suggestions.

1 Algebraic Specification.

Algebraic Specification, borrowed directly from mathematics, is a software engineering tool based on the idea of definition by universal diagrams. For example, the data type *AdditiveIntegers* may be specified most simply as the free abelian group on one generator; that is, it is a set *Int* together with operations

$$\begin{aligned} + &: Int \times Int \rightarrow Int \\ - &: Int \rightarrow Int \\ 0 &: \rightarrow Int \\ 1 &: \rightarrow Int \end{aligned}$$

such that

$$\begin{aligned} (x + y) + z &= x + (y + z) \\ x + y &= y + x \\ x + -x &= 0 \\ -x + x &= 0 \end{aligned}$$

which is *initial*¹ in the category of all algebraic structures with the same operations and equations. While theological arguments rage among computer scientists about the relative value of algebraic specification, it is clearly an appropriate tool in formalizing the mathematical subject it was borrowed from.

1.1 Order sorted algebras.

I will use in what follows a formulation of initial semantics due to Goguen [4]. A good general reference is [3]; [2] provides a good introduction to category theory.

Let $S = (S, \leq)$ be a partially ordered set (*poset*) with order relation $\leq$. The elements of S will be called *sorts* and will take on the semantic role of *data types*, where the $\leq$ relation corresponds to type inclusion. An *S-sorted set* A is a family of sets $\{A_s\}$, indexed by the elements of S , such that $s \leq t$

¹An object e is initial in a category C if whenever x is an object of C , there is a unique map $e \rightarrow x$ in C .

implies $A_s \subseteq A_t$. An S -sorted function $f : A \rightarrow B$ of S -sorted sets is a family of functions $f_s : A_s \rightarrow B_s$.

Let S^* denote the set of finite length strings $s_1 \dots s_n$, $n \geq 0$. The order relation $\leq$ may be extended to S^* by $s_1 \dots s_m \leq t_1 \dots t_n$ if $m = n$ and $s_1 \leq t_1, \dots, s_n \leq t_n$. An *order sorted signature* with sort set S and operations Σ is a pair (S, Σ) where S is a poset and Σ is a family $\{\Sigma_{w,s} | w \in S^*, s \in S\}$ of sets of operation symbols such that $\sigma \in \Sigma_{v,s} \cap \Sigma_{w,t}$, and $v \leq w$ imply $s \leq t$ (monotonicity). An operation $\sigma \in \Sigma_{w,s}$ is completely specified by σ , w , and s . We shall call that 3-tuple the *signature* of the operator and denote it by $\sigma : w \rightarrow s$ or $w \xrightarrow{\sigma} s$; w is called the *arity*, s the *sort*, and the pair (w, s) the *rank* of σ . If w is the empty string λ , we call the operation $\sigma : \lambda \rightarrow s$ a *constant* and denote it by $\sigma : \rightarrow s$ or $\sigma : s$. We allow a function symbol σ to be *overloaded* in the sense that σ may appear with more than one $\Sigma_{w,s}$; however, a constant may not be overloaded because of the monotonicity condition.

An *order sorted algebra* with signature (S, Σ) is an S sorted set A together with functions

$$\sigma_{w,s} : A_{w_1} \times \dots \times A_{w_n} \rightarrow A_s.$$

for each pair $(w, s) \in S^* \times S$ and symbol $\sigma \in \Sigma_{w,s}$. Operations must be compatible in the sense that $\sigma_{v,s}(x) = \sigma_{w,t}(x)$ whenever $\sigma \in \Sigma_{v,s} \cap \Sigma_{w,t}$, and $v \leq w$.

For technical reason's (see [4]) we want to require of our signatures that they be *coherent* in the sense that

1. S is *coNoetherian*—i.e., there is no infinite descending chain $s_1 > s_2 > \dots$ in S .
2. S is *locally filtered*—i.e., if s and t are in the same connected component of S (there exists a chain $s = s_1 \leq t_1 \geq s_2 \leq \dots \leq t_n = t$), then there exists $u \in S$ such that $s, t \leq u$.
3. Σ is *regular*—i.e., whenever $\sigma \in \Sigma_{w,t}$ and $u \leq w$, there is a least rank (v, s) such that $u \leq v$ and $\sigma \in \Sigma_{v,s}$.

1.2 The algebra of terms and equations.

Let $\Sigma = (S, \Sigma)$ be a (coherent) order-sorted signature. The *algebra of terms* in Σ is the order-sorted algebra T_Σ defined recursively by

1. if $\sigma : \rightarrow s$ then $\sigma \in T_s$.
2. if $\tau_1 \in T_{w_1}, \dots, \tau_n \in T_{w_n}$ and $\sigma \in \Sigma_{w,s}$, then
$$\sigma(\tau_1, \dots, \tau_n) \in T_s$$
3. if $\tau \in T_s$ and $s \leq t$, then $\tau \in T_t$.

with operations

$$\sigma_{w,s} : T_{w_1} \times \dots \times T_{w_n} \rightarrow T_s.$$

given by

$$\sigma_{w,s}(\tau_1, \dots, \tau_n) = \sigma(\tau_1, \dots, \tau_n).$$

T_Σ is the *free* order-sorted algebra with signature Σ . A consequence of regularity is for each term τ , there is a least sort $s \in S$ such that $\tau \in T_s$ [4].

The following discussion of equations differs somewhat from [4]. Let X be a set of typed variables over S distinct from Σ ; that is, X is a set of pairs $\{x_i : w_i, \dots, x_n : w_n\}$ of symbols x_i and associated sorts w_i . The *algebra of forms* in Σ is the algebra of terms $T_\Sigma[X] = T_{(\Sigma \cup X)}$. If A is a Σ -algebra, a *substitution* in A is an S -sorted map $\theta : T_\Sigma[X] \rightarrow A$. A substitution is uniquely determined by giving the values $\theta(x_i) \in A_{w_i}$.

An *equation* over Σ is a set X of typed variables together with a pair of terms $\tau, \theta \in T_\Sigma[X]$ that have the same least sort. We denote an equation by

$$(\forall x_1 : w_1, \dots, x_n : w_n) \tau = \theta,$$

or $(\forall X) \tau = \theta$ when the types are understood from context. As an added refinement, we could also introduce conditional equations [4]. I will omit them here for simplicity.

If Γ is a set of equations, a Σ -sorted algebra A satisfies Γ if for each equation $(\forall X) \tau = \theta$ in Γ and substitution $\phi : T_\Sigma[X] \rightarrow A$, one has $\phi(\tau) = \phi(\theta)$. Given

a set of equations Γ , we may construct that algebra of terms satisfying Γ , $\mathcal{T}_{\Sigma, \Gamma}$, which is the quotient of $\mathcal{T}_{\Sigma}$ by the equivalence relation generated by Γ and the operations on $\mathcal{T}_{\Sigma}$. Then [4], $\mathcal{T}_{\Sigma, E}$ is the (unique) initial object in the category of all Σ -sorted algebras that satisfy the equations Γ .

1.3 Example: the rational numbers.

object *RationalNumbers* is

sorts *Int*, *Rat*, *PosInt*, *NZInt*, *NZRat*.

subsorts

PosInt < *NZInt* < *Int* < *Rat*,
NZInt < *NZRat* < *Rat*.

operations

$+$: *Int* $\times$ *Int* $\rightarrow$ *Int*
 $+$: *PosInt* $\times$ *PosInt* $\rightarrow$ *PosInt*
 $+$: *Rat* $\times$ *Rat* $\rightarrow$ *Rat*
 $-$: *Int* $\rightarrow$ *Int*
 $-$: *NZInt* $\rightarrow$ *NZInt*
 $-$: *Rat* $\rightarrow$ *Rat*
 $-$: *NZRat* $\rightarrow$ *NZRat*
 $\times$: *Int* $\times$ *Int* $\rightarrow$ *Int*
 $\times$: *NZInt* $\times$ *NZInt* $\rightarrow$ *NZInt*
 $\times$: *PosInt* $\times$ *PosInt* $\rightarrow$ *PosInt*
 $\times$: *Rat* $\times$ *Rat* $\rightarrow$ *Rat*
 $\times$: *NZRat* $\times$ *NZRat* $\rightarrow$ *NZRat*
 $()^{-1}$: *NZRat* $\rightarrow$ *NZRat*
 0 : $\rightarrow$ *Int*
 1 : $\rightarrow$ *PosInt*

equations

$(x + y) + z = x + (y + z)$
 $x + y = y + x$
 $x + -x = 0$
 $-x + x = 0$
 $(x \times y) \times z = x \times (y \times z)$
 $x \times y = y \times x$
 $1 \times x = x$
 $x \times 1 = x$

$x \times (y + z) = (x \times y) + (x \times z)$
 $(x + y) \times z = (x \times z) + (y \times z)$
 $(\forall x : NZRat) x \times x^{-1} = 1$

end *RationalNumbers*

The subsort *NZInt* is required in this example to insure regularity of the negation operation. Notice that the various subsorts of *Int* and *Rat* are used to represent the values of certain predicates and represent the domains of partially defined functions. Subsorts can also be made to represent error conditions [4]. Goguen has developed a logical programming language, **OBJ3**, which directly implements term algebras from order-sorted signatures with equations [7].

2 Categories and Specifications.

The above formulation of semantics provides an ideal tool for representing mathematical semantic information in a computer. We will encode the mathematical concept of a category as an order-sorted signature plus equations.

Definition 1 A specification is a pair $C = (\Sigma_C, \Gamma_C)$ where Σ_C is an order-sorted signature and Γ_C is a set of equations.

Definition 2 A model of a specification C is an Σ_C -sorted algebra A which satisfies the equations Γ_C . The model category of C is the category $\mathcal{M}_C$ of all models A of C and Σ_C -sorted homomorphisms between them.

A specification always has at least one model, the term algebra $\mathcal{T}_{\Sigma, \Gamma}$, which is initial in $\mathcal{M}_C$.

2.1 Equational deduction.

A sticky point in the definition of specification is how to characterize when two specifications are the same or related. The following rules of inference apply to equations over an order-sorted signature Σ [4].

1. Reflexivity.

$$\overline{\Gamma \vdash (\forall X)\tau = \tau}$$

2. Symmetry.

$$\frac{\Gamma \vdash (\forall X)\tau = \theta}{\Gamma \vdash (\forall X)\theta = \tau}$$

3. Transitivity.

$$\frac{\Gamma \vdash \tau = \theta, \quad \Gamma \vdash \theta = \psi}{\Gamma \vdash \tau = \psi}$$

4. Restriction.

$$\frac{\Gamma \vdash (\forall x_1 : t_1, \dots, x_n : t_n)\tau = \theta}{\Gamma \vdash (\forall x_1 : s_1, \dots, x_n : s_n)\tau = \theta}$$

for $s_1 \leq t_1, \dots, s_n \leq t_n$.

5. Congruence. If $\alpha, \beta : \mathcal{T}_\Sigma[X] \rightarrow \mathcal{T}_\Sigma[Y]$ are substitutions, and $\tau \in \mathcal{T}_\Sigma[X]$, then

$$\frac{\text{for each } x \in X, \Gamma \vdash (\forall Y)\alpha(x) = \beta(x)}{\Gamma \vdash (\forall Y)\alpha(\tau) = \beta(\tau)}$$

6. Substitutivity. If $\alpha : X \rightarrow \mathcal{T}_\Sigma[Y]$ is a substitution, then

$$\frac{\Gamma \vdash (\forall X)\tau = \theta}{\Gamma \vdash (\forall Y)\alpha(\tau) = \alpha(\theta)}$$

Unfortunately, deduction by these rules is undecidable since the word problem for groups is an example of equational deduction. We will therefore take the first four rules to define equivalence of specifications when computability is an issue.

Definition 3 If Γ and Ψ are two sets of equations over an order-sorted signature Σ then Ψ is weakly derivable from Γ if each equation of Ψ follows from Γ by renaming of variables and rules 1 to 4 above. We write $\Gamma \vDash \Psi$.

Clearly, $\Gamma \vDash \Psi$ implies $\Gamma \vdash \Psi$ which in turn implies $\mathcal{M}_{(\Sigma, \Gamma)} \subseteq \mathcal{M}_{(\Sigma, \Psi)}$.

2.2 Views.

Definition 4 Let $\mathcal{C}$ and $\mathcal{D}$ be specifications, a view of $\mathcal{C}$ as $\mathcal{D}$, written $V : \mathcal{C} \Rightarrow \mathcal{D}$, is a pair (f, g) where f is an order preserving function $S_{\mathcal{D}} \rightarrow S_{\mathcal{C}}$ of sorts and g is a family of functions $g_{w,s} : (\Sigma_{\mathcal{D}})_{w,s} \rightarrow \Sigma_{\mathcal{C}}$ such that

1. if $\sigma \in (\Sigma_{\mathcal{D}})_{v,s}$, then $g_{v,s}(\sigma) \in (\Sigma_{\mathcal{C}})_{w,t}$, where $f(v) \leq w$ and $t \leq f(s)$.
2. if $\sigma \in (\Sigma_{\mathcal{D}})_{v,s} \cap (\Sigma_{\mathcal{D}})_{w,t}$ where $v \leq w$, then $g_{v,s}(\sigma) = g_{w,t}(\sigma)$.
3. if for each set of typed variables $X = \{x_1 : s_1, \dots, x_n : s_n\}$ and form $\tau \in \mathcal{T}_{\Sigma_{\mathcal{D}}}[X]$ we let $v^*(\tau) \in \mathcal{T}_{\Sigma_{\mathcal{C}}}[v^*(X)]$ be defined inductively by

- (a) $v^*(x_i : s_i) = (x_i : f(s_i))$, and
- (b) $v^*(\sigma(\tau_1, \dots, \tau_r)) = g(\sigma)(v^*(\tau_1), \dots, v^*(\tau_r))$.

then

$$\Gamma_{\mathcal{C}} \vdash v^*(\Gamma_{\mathcal{D}}).$$

If we wish to check algorithmically that a pair (f, g) is a view, we must replace the last condition by $\Gamma_{\mathcal{C}} \vDash v^*(\Gamma_{\mathcal{D}})$.

A view $v : \mathcal{C} \Rightarrow \mathcal{D}$ induces a functor $v_* : \mathcal{M}_{\mathcal{C}} \Rightarrow \mathcal{M}_{\mathcal{D}}$ in an obvious way. A view $v = (f, g)$ is *faithful* or an *embedding* if f is cofinal—that is, for all $s \in S_{\mathcal{D}}$, there exists $t \in S_{\mathcal{C}}$ such that $f(s) \leq t$. If v is faithful, then the induced functor v_* is faithful.

2.3 Parameterized specifications.

Definition 5 Let $\mathcal{C}$ and $\mathcal{D}$ be specifications. A parameterization of $\mathcal{D}$ by $\mathcal{C}$ is a view $v = (f, g)$, $v : \mathcal{D} \Rightarrow \mathcal{C}$, such that f and g are inclusions. Denote this structure by $\mathcal{D}_v(X : \mathcal{C})$, where X is a variable symbol, or $\mathcal{D}(X : \mathcal{C})$ when v is clear from context. The functor v_* is called the forgetful functor of the parameterization. [3]. Given a model A of $\mathcal{C}$, let $\mathcal{M}_{\mathcal{D}(A)}$ denote the category of pairs (B, α) consisting of an object of $\mathcal{M}_{\mathcal{D}}$ and a $\mathcal{C}$ -morphism $\alpha : A \rightarrow v_*(B)$.

The key result that concerns us is

Proposition 1 *If A is a model of $\mathcal{C}$ then there is an object $\mathcal{F}_v(A)$ and $\mathcal{C}$ -morphism $\phi : A \rightarrow v_*(\mathcal{F}_v(A))$ such that for any model B and $\mathcal{C}$ -morphism $\alpha : A \rightarrow v_*(B)$ there is a unique $\mathcal{D}$ -morphism $\beta : \mathcal{F}_v(A) \rightarrow B$ such that $v_*(\beta) \circ \phi = \alpha$. Furthermore, the construction $\mathcal{F}_v(A)$ is a functor $\mathcal{M}_{\mathcal{C}} \Rightarrow \mathcal{M}_{\mathcal{D}}$. It is called the free functor of the parameterization.*

This proposition says that $\mathcal{M}_{\mathcal{D}(A)}$ has an initial object. The proof uses a generalization of the term algebra of a specification [3].

Several special cases are sufficiently important to have their own notations. Let $\mathcal{C}_1, \dots, \mathcal{C}_n$ be specifications. The *product specification*, $\Pi \mathcal{C}_i = \mathcal{C}_1 \times \dots \times \mathcal{C}_n$, is the disjoint union of the specifications $\mathcal{C}_i$. This construction generalizes easily to inverse limits over a finite diagram of specifications, and is functorial over the category of specifications and views. If $\dots, X_n$ are symbols and $\mathcal{D}$ is parameterized by $\Pi \mathcal{C}_i$, we may denote this parameterization by $\mathcal{D}(X_1 : \mathcal{C}_1, \dots, X_n : \mathcal{C}_n)$.

If $\mathcal{C} = (S, \Sigma, \Gamma)$ is a specification and X is a set of typed variables not in Σ , let $\mathcal{C}[X] = (S, (\Sigma \cup X), \Gamma)$. We will call it the *$\mathcal{C}$ -algebra generated by X* . If A is a $\mathcal{C}$ -model, then the initial algebra of $\mathcal{C}[X](A)$ is the *algebra of A -forms $A[X]$* . Similarly, if Θ is a set of equations over Σ , then the quotient of $\mathcal{C}$ by Θ is the specification $\mathcal{C}/\Theta = (S, \Sigma, (\Gamma \cup \Theta))$. $\mathcal{C}/\Theta$ is a contravariant functor of equational deduction.

3 Type inference.

At least three levels of type inference are useful to an algebraic simplification system:

sort: determine, for example, whether a form in *RationalNumbers* is in *NZRat*.

domain: determine, for example, whether a form is in *RationalNumbers*.

category: determine, for example, whether *RationalNumbers* is to be considered a *Ring*, a *Field* or an *OrderedSet*.

Two problems arise in type inferencing: automatic insertion of coercions, and polymorphism. The object of the first is to find "smallest" domains in which all operators in the expression and its subexpressions are defined. For example, the polynomial ring functor $Pol(R)$ comes equipped with a canonical inclusion $R \hookrightarrow Pol(R)$, and the integers come equipped with a canonical inclusion $Int \hookrightarrow Rat$ into the rational numbers. In expressions such as

$$1 + (3/2)X,$$

we want to make the inference that 3 and 2 must be coerced to *Rat* in order to carry out $/$; $(3/2)X$ must be coerced to $Pol(Rat)$; and 1 must be coerced first to *Rat* and then to $Pol(Rat)$. Inferencing of this kind is normally made from the bottom up, (the inside of the expression outward): the type of each constant is first determined. Then, given an operator all of whose operands are typed, a knowledge base of functors and canonical inclusions is searched to find one or more coercions that will permit the operator to be typed.

The second problem is more difficult. Consider

$$\frac{(x+y)^{1+n} + 1}{1+nx}$$

where the user wants to work with rational functions over a finite field of p -elements. This formula presents the problem of polymorphic constants. To a mathematician, the types of each subexpression are immediately clear: n is an integer variable which must be reduced modulo p in the denominator of the expression, x and y are finite field variables, 1 appearing in the exponent is an integer and the other 1's are the multiplicative identity in the finite field. Inferences of this kind are often made from the top down: the total expression must be over a finite field, so the fraction bar and its two operands must be also. Exponentiation requires an integer second argument; therefore, $1+n$ must be an integer; and so forth.

In this section I will provide a structure to deal with the first of these two problems. I hope to deal with the second in a later paper.

3.1 Algebraic Theories.

I will provide for a knowledge base to deal with automatic coercion by introducing another level of abstraction: *algebraic theory*.

The first component of a theory C that we need is a *hierarchy of categories*. Let $(C, \subseteq)$ be a poset, and Cat a monic functor from C to the category of specifications and views. We will assume that for each relation $s \subseteq t$ in C that the view $Cat_s \Rightarrow Cat_t$ is a parameterization and an embedding, and we shall call Cat_s a *specialization* of Cat_t . Let $Cat_s = (S_s, \Sigma_s, \Gamma_s)$. Our notation will be simplified if we assume that the view $Cat_s \Rightarrow Cat_t$ is an inclusion of S_t in S_s and Σ_t in Σ_s where $S_t \subseteq S_s$ is cofinal.

Given a hierarchy of categories, (C, Cat) , we now need to specify a family Θ of *functor symbols*. Functors arise in two ways: either *initial specifications*, or *formal functors*—that is, symbols that are to be interpreted as functors on models defined by some algorithmic method (see *model morphisms* below). Here we are only concerned with the formal properties of functors. We assume the pair (C, Θ) is an *order-sorted signature*, and we will use the symbol Θ also to denote our whole theory. We will call a term $F \in T_\Theta$ a *domain*. We will write $F \in s$ if the least sort t of F is a specialization of s .

Finally, we need to allow for domain inferencing of expressions. Let $\Delta = T_\Theta$ be the set of domains associated to our signature (C, Θ) . We need a predicate on Δ representing “canonical inclusion.” For $F, G \in s$, write $F \hookrightarrow G$ for “ F is canonically included in G in the category s .” This predicate must satisfy the following:

1. if $F \hookrightarrow G$, then $F, G \in s$.
2. if $F \hookrightarrow G$ and $s \subseteq t$, then $F \hookrightarrow G$.
3. $F \hookrightarrow F$.
4. if $F \hookrightarrow G$ and $G \hookrightarrow H$, then $F \hookrightarrow H$.

Another way of viewing all this is to consider for each $s \in C$ the set $\Delta_s = \{F \in s\}$ to be the objects of a category with at most one morphism $F \rightarrow G$ whenever $F \hookrightarrow G$,

where $F \hookrightarrow F$ corresponds to the identity morphism. The view $Cat_s \Rightarrow Cat_t$ for $s \subseteq t$ induces a *forgetful functor* from Δ_s to Δ_t . If $F \hookrightarrow G$ and $G \hookrightarrow F$ we write $F \equiv G$.

3.2 The total algebra.

Continue the notation of the previous section. We want to construct a *total algebra*, $Tote$, of our theory.

Let T be the set of all triples $c.f.s$ where $c \in C$, $s \in S_c$, and f represents an equivalence class of domains $F \in c$. Order T by the relation $c.f.s \preceq d.g.t$ if

$$\begin{cases} d \subseteq c, \\ f \hookrightarrow g, \text{ and} \\ s \leq t. \end{cases}$$

It is easy to check that this is a partial order.

Construct an operator set Φ as follows: for each $w \in T^*$, $d.g.t \in T$, where $w_i = c_i.f_i.t_i$, let $\sigma \in \Phi_{w,s}$ if

$$\begin{cases} d = c_i, \text{ for all } i, \\ f_i = g, \text{ for all } i, \text{ and} \\ \sigma \in \Sigma_{t_1, \dots, t_n, s}. \end{cases}$$

For constants we must make a special rule: $\sigma \in \Phi_{\lambda, d.g.s}$ if d is maximal such that $\sigma \in S_d$, g is initial in d and $\sigma \in \Sigma_{\lambda, s}$. This clearly requires further assumptions on Θ .

We will make the following assumptions for all theories:

1. C is Noetherian and has finite least upper bounds.
2. If $d \subseteq c$, $s \in S_d$ and $s \leq d$ in S_c , then $d \in S_d$.
3. For each $c \in C$, $(\Delta_c, \hookrightarrow)$ is co-Noetherian and has finite greatest lower bounds.

The following are valid under these assumptions.

Lemma 1 Φ is an order sorted signature.

Proof. Straightforward.

Lemma 2 Φ is regular.

Proof. By [4], lemma 7, it suffices to show that if $w_0 \preceq w_1$ and $w_0 \preceq w_2$ and $\sigma \in \Phi_{w_1, t_1} \cap \Phi_{w_2, t_2}$ then there exists $w \preceq w_1, w_2$ such that $\sigma \in \Phi_{w, s}$ and $w_0 \preceq w$. The rest of the proof is lengthy but straightforward.

Using now the results of [4], we have immediately our main theorem.

Theorem 1 *If τ is a term in the operators Σ and τ is typeable—that is, there exists c.f.s such that $\tau \in (T_\Phi)_{c.f.s}$ —then τ has a least type c.f.s. We call c the most general category, f the smallest domain, and s the least sort of τ .*

3.3 Example: Polynomials.

category *Sets*

sorts *S*

end *Sets*

category *Rings* = *Sets* + *Integers* +

operations

$$+ : S \times S \rightarrow S$$

$$- : S \rightarrow S$$

$$\times : S \times S \rightarrow S$$

$$\text{id} : \text{Int} \rightarrow S$$

$$\lambda x n.x^n : S \times \text{PosInt} \rightarrow S.$$

equations ...

end *Rings*

category *VariableSets* = *Sets* +

sorts *Vars*

subsorts *Vars* < *S*

end *VariableSets*

theory *Polynomials*

categories *Sets*, *Rings*, *VariableSets*.

subcategories *Variables* < *Sets*,
Rings < *Sets*

functors

$$[] : \text{VariableSets}$$

$$(\forall X : \text{symbol}).[X] : \text{Vars} \Rightarrow \text{Vars}$$

$$Z : \text{Rings}$$

$$(\lambda R X).\text{Pol}_R[X] : \text{Rings} \times \text{Vars}$$

$$\Rightarrow \text{Rings}$$

inclusions

$$[] \hookrightarrow [X]$$

$$[X][Y] \hookrightarrow [Y][X]$$

$$[X][X] \hookrightarrow [X]$$

$$F \hookrightarrow G \Rightarrow [X]F \hookrightarrow [X]G$$

$$A \hookrightarrow B \Rightarrow \text{Pol}_R[A] \hookrightarrow \text{Pol}_R[B]$$

$$R \hookrightarrow S \Rightarrow \text{Pol}_R[A] \hookrightarrow \text{Pol}_S[A]$$

$$A \xrightarrow{\text{Sets}} \text{Pol}_R[A]$$

$$R \xrightarrow{\text{Rings}} \text{Pol}_R[A]$$

end *Polynomials*

3.4 Decidability.

In order to carry out typing of expressions, one would clearly like the canonical inclusion relation on the functor algebra of a theory to be decidable. The following result is an aid to setting up decidable relations.

Proposition 2 *Let a functor algebra T_Θ be generated by a finite set of functors Θ . Let R be a finite set of relations $F_i \hookrightarrow G_i$, and let $E \subseteq \Theta$ be a finite subset of the functors which preserve canonical inclusion. Let $\mathcal{R}$ be the relation on T_Θ generated by the axioms of canonical inclusion plus the following axiom: if $F_i \hookrightarrow G_i$ and $H \in E$ then*

$$H(F_1, \dots, F_n) \hookrightarrow H(G_1, \dots, G_n)$$

Let $\mathcal{L}$ be the lexicographic order on T_Θ . If $R \subseteq \mathcal{L}$ then the relation $\mathcal{R}$ is decidable.

Proof. The lexicographic order provides a notion of the complexity of a term. Since each relation in R increases the complexity of a term, there are only finitely many inferences to check if two terms are related.

Decidability of our typing algorithm is not strictly necessary; in practice, it suffices that

our algorithm be *semi-decidable*—that is, typeable expressions will eventually be typed. The reason for this is that expression typing is part of an interface with a user who will almost always generate easily typed expressions. A typing system may be programmed to yell for help if a typing problem uses excessive resources. Experience shows that with the closely related problem of typing the polymorphic λ -calculus, that the algorithm is reasonable computationally although only semi-decidable [6].

4 Models.

I discuss in this section the computational meaning of a *model* of a functor. Since I am proposing a semantic system as an adjunct to a rewrite rule system, a functor can, in principle, be automatically implemented by term rewriting from its initial specification. That is the approach of OBJ3. However, such implementations alone are unlikely to satisfy either the efficiency needs or hacking instinct of mathematical users. I propose here a meaning for model which encompasses both algebraic and algorithmic specifications of a model.

We assume here that our implementation language supports functions as first class objects.

Definition 6 A model morphism for a functor $F : C_1 \times \dots \times C_n \Rightarrow D$ is a function $M(M_1, \dots, M_n)$ of model morphisms $M_1, \dots, M_n$ that returns a function from operators in D to functions in the underlying implementation language. That is, given implementations for objects in $C_1, \dots, C_n$ the function $M(M_1, \dots, M_n)(\sigma)$ implements the operation σ of of an object of D .

The advantage of treating models in this abstract way is that—assuming our rewrite rule system can read model morphisms—a computation specified by algebraic techniques is indistinguishable from one specified algorithmically. In particular, our semantic system itself can be implemented in such a rewrite system so that components

of theories—such as, *VariableSets* in the example above—can be implemented with the obvious easy and fast algorithms.

References

- [1] S. K. Abdali, G. W. Cherry, and N. Soifer, *A Smalltalk System for Algebraic Manipulation*, OOPSLA '86 Conference Proceedings, SIGPLAN Notices, Vol. 21 No. 11, Association for Computing Machinery, Nov. 1986, pp. 277–283.
- [2] M. Arbib and E. Manes, *Arrows, Structures, and Functors*, Academic Press, New York, 1975.
- [3] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification 1*, Springer Verlag, Berlin, Heidelberg, 1985.
- [4] J. A. Goguen and J. Meseguer, *Order-Sorted Algebra I*, draft manuscript, May 1988.
- [5] R. Jenks, R. Sutor, and Stephen Watt, *Scratchpad II: an abstract datatype system for mathematical computation*, Mathematical Aspects of Scientific Software, The IMA Volumes in Mathematics and its Applications 14, Springer Verlag, 1988, pp. 157–182.
- [6] F. Pfenning, *Partial Polymorphic Type Inference and Higher-Order Unification*, Proc. of 1988 ACM Conf. on Lisp and Functional Programming, Association for Computing Machinery, 1988, pp. 153–163.
- [7] D. Rydeheard and R. Burstall, *Computational Category Theory*, Prentice Hall, New York, 1988.
- [8] R. S. Sutor and R. D. Jenks, *The type inference and coercion facilities in the Scratchpad II interpreter*, Proc. SIGPLAN 87 Symp. on Interpreters and Interpretive Techniques, SIGPLAN Notices, 22,7 pp. 56–63.

SYMBOLIC COMPUTATION WITH SYMMETRIC POLYNOMIALS AN EXTENSION TO MACSYMA

Annick Valibouze
LITP,
4, Place Jussieu, 75252 Paris Cedex 05
and "GRECO DE CALCUL FORMEL" No 60
UUCP: avb@litp.univ-p6-7.fr

INTRODUCTION

We present here a package of manipulations of symmetric polynomials implemented in Franzisp. This package, called SYM, constitutes at present an extension of the system of symbolic computation MACSYMA. It performs a few manipulations on symmetric polynomials; it can also be used for direct applications. Some algorithms extend easily to functions that are symmetric with respect to sets of variables (i.e. multi-symmetric functions); these functions will be dealt with in the present paper.

1 Definitions and notations

1.1 Partitions and multi-partitions

Let us first introduce the notion of a partition, which is the basic object that allows us to represent the symmetric polynomials in the most possible contracted form. For more details, the reader is referred to [Andrews] or [Macdonald].

A *partition* is a finite or infinite sequence $I = (i_1, i_2, \dots, i_n, \dots)$ of non-negative integers in decreasing order: $i_1 \geq i_2 \geq \dots \geq i_n \geq \dots$, and containing only a finite number of non-zero terms. We make the convention that sequences only differing by the number of zeros at the end are equal. For example $(2, 1)$ and $(2, 1, 0, 0)$ are the same partition. The non-zero i_k of I are called the *parts* of I . The number of parts is the *length* of I and the sum of the parts is the *weight* of I . We shall call *multi-partition of order p* a finite sequence I of length p , $I = (I_1, I_2, \dots, I_p)$, where each I_k is a partition.

1.2 Generalities about symmetric functions

Let $\mathcal{A}$ be a ring, and let $D = (d_1, d_2, \dots, d_p)$ be an element of $\mathbb{N}^p$ with $d_1 + d_2 + \dots + d_p = n$. Let $X = (x^{(1)}, x^{(2)}, \dots, x^{(p)})$, where each $x^{(r)}$ is an alphabet of d_r variables $x_1^{(r)}, x_2^{(r)}, \dots, x_{d_r}^{(r)}$. Then $R_D = \mathcal{A}[X]$ is the ring of polynomials in the n variables $x_i^{(r)}$ ($i = 1, \dots, d_r$ and $r = 1, \dots, p$) with coefficients in $\mathcal{A}$. The product $S_{d_1} \times S_{d_2} \times \dots \times S_{d_p}$ of the symmetric groups S_{d_r} will be

denoted by S_D .

For each element σ of S_n and each finite sequence of n elements $T = (t_1, t_2, \dots, t_n)$, $\sigma(T)$ is the sequence $(t_{\sigma(1)}, t_{\sigma(2)}, \dots, t_{\sigma(n)})$. This generalizes as follows: let $T = (t^{(1)}, t^{(2)}, \dots, t^{(p)})$ be a p -tuple of finite sequences $t^{(r)}$ of d_r element. For each element $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_p)$ of S_D we define:

$$\sigma(T) = (\sigma_1(t^{(1)}), \sigma_2(t^{(2)}), \dots, \sigma_p(t^{(p)})).$$

$G_{S_D}(T)$ will be the *stabilizer* of T under the action of S_D (the subgroup of the elements of S_D leaving T unchanged). If $f \in R_D$, $O_{S_D}(f)$ will denote the orbit of f under S_D , i.e., the set of polynomials h of R_D such that $h(X) = f(\sigma(X))$ for an element σ of S_D .

A polynomial P of R_D is said to be *multi-symmetric* of order D if $P(X) = P(\sigma(X))$ for all $\sigma \in S_D$ (i.e. $\text{card}(O_{S_D}(P)) = 1$). This algebra of invariants will be denoted by $R_D^{S_D}$. If $p = 1$ we simply say that P is *symmetric*.

For $p = 1$ we take $D = d_1 = n$ and $X = (x_1, x_2, \dots, x_n)$. If $U = (u_1, u_2, \dots, u_n)$ is an element of $\mathbb{N}^n$, we define the monomial X^U by:

$$X^U = x_1^{u_1} x_2^{u_2} \dots x_n^{u_n},$$

and if U is a D -tuple having p finite sequences of integers $u^{(1)}, \dots, u^{(p)}$ of length $d_1, \dots, d_p$, respectively, then:

$$X^U = (x^{(1)})^{u^{(1)}} \dots (x^{(p)})^{u^{(p)}}.$$

With a multi-partition I we associate the *monomial form* $M_I(X)$, which is the sum of the elements of the orbit of X^I under the S_D -action:

$$M_I(X) = \sum_{\sigma \in S_D / G_{S_D}(I)} X^{\sigma(I)}.$$

Examples:

$$- p = 1 - M_{(3,2)}(x, y, z) = x^3 y^2 z^2 + y^3 x^2 z^2 + z^3 x^2 y^2.$$

$$- p = 2 - \text{For } X = ((x, y), (a, b, c)) \text{ we have } M_{((3,2),(1,1))}(X) = x^3 y^2 ab + x^3 y^2 ac + x^3 y^2 bc + y^3 x^2 ab + y^3 x^2 ac + y^3 x^2 bc.$$

1.3 Contracted and partitioned forms

A *monomial form* $M_I(X)$ will be represented either by a monomial X^J of the orbit of X^I , called a *contracted monomial form*, or by the partition I . If we give it a coefficient, then the monomial form is represented by a *contracted term* or by a *partitioned term*, the latter being a list in which the first element is the coefficient and the rest is the partition. We can now represent a symmetric polynomial (or a multi-symmetric polynomial) by a *contracted polynomial*, the sum of the contracted terms, or by a *partitioned polynomial*, the list of the partitioned terms.

For example the contracted polynomial associated with the polynomial $3x^4 + 3y^4 - 2xy^5 - 2x^5y$, symmetric in the variables x et y , is $3x^4 - 2xy^5$ and the partitioned polynomial is $[[3,4], [-2,5,1]]$.

1.4 Types of arguments

For the description of the function we use the following notations:

card is the cardinality of the set of the variables.

e_i : i^{th} elementary symmetric function

p_i : i^{th} power function

$\text{l\_ele} = [e_1, e_2, e_3, \dots, e_n]$, where the number n is important in some definitions of the function

$\text{l\_cele} = [\text{card}, e_1, e_2, e_3, \dots, e_n]$

$\text{l\_pui} = [p_1, p_2, p_3, \dots, p_m]$

$\text{l\_cpui} = [\text{card}, p_1, p_2, p_3, \dots, p_m]$

$\text{sym} < \text{---} >$ is a symmetric polynomial, but the representation is not specified

$\text{fmc} < \text{---} >$ contracted monomial form

$\text{part} < \text{---} >$ partition

$\text{tc} < \text{---} >$ contracted term

$\text{tpart} < \text{---} >$ partitioned term

$\text{psym} < \text{---} >$ symmetric polynomial in its extended form

$\text{pc} < \text{---} >$ symmetric polynomial in a contracted form

$\text{multi\_pc} < \text{---} >$ multi-symmetric polynomial in a contracted form under S_D

$\text{ppart} < \text{---} >$ symmetric polynomial in its partitioned form

$P(x_1, \dots, x_q)$ is a polynomial in the variables $x_1, \dots, x_q$

lvar is a list of variables of X in the case $p = 1$. $\{\text{lvar}_1, \dots, \text{lvar}_p\}$ is a list of lists of variables representing the multi-alphabet X and where the variables of lvar_j represent the d_j variables of the alphabet $x^{(j)}$.

Remarks :

1- The functions of SYM can complete the lists, such as l_cele , with formal values. This values are $e1$ for the i^{th} elementary symmetric function and $p1$ for the i^{th} power function.

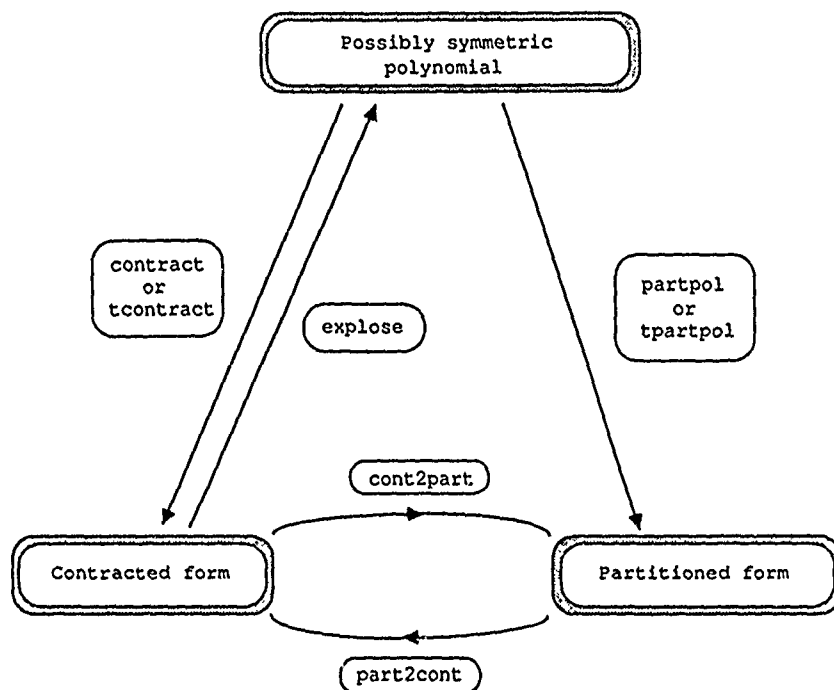
2- There exist many kinds of evaluations for the polynomials under MACSYMA : ev , expand , rat , ratsimp . With SYM the choice is possible with a flag oper . In each call of a function, SYM tests if the variable oper is modified. In this case, the modification is made as follows: if $\text{oper} = \text{meval}$, it uses the ev mode, if $\text{oper} = \text{expand}$, it uses the expand mode (it is more efficient for the numeric calculations), if $\text{oper} = \text{rat}$, it uses the rat form and if $\text{oper} = \text{ratsimp}$, it uses the ratsimp form.

2 Description of the available functions

2.1 Change of representation

- $\text{tpartpol}(\text{psym}, \text{lvar}) \rightarrow \text{ppart}$,
 $\text{partpol}(\text{psym}, \text{lvar}) \rightarrow \text{ppart}$
 give, in the lexicographic order, increasing and decreasing, respectively, the partioned polynomial associated with the polynomial psym . The function tpartpol tests if the polynomial psym is actually symmetric.
- $\text{contract}(\text{psym}, \text{lvar}) \rightarrow \text{pc}$
 $\text{tcontract}(\text{psym}, \text{lvar}) \rightarrow \text{pc}$
 act as partpol and give the contracted form.

- `cont2part(pc,lvar) → ppart`
gives the partitioned polynomial associated with the contracted form pc.
- `part2cont(ppart,lvar) → pc`
gives a contracted form associated with the partitioned form ppart.
- `explode(pc,lvar) → psym`
gives a contracted form associated with the extended form psym.



Examples:

```
tpartpol(expand(x^4+y^4+z^4-2*(x*y+x*z+y*z)), [x,y,z]);
```

```
[[1, 4], [- 2, 1, 1]]
```

Now suppose that the polynomial $2a^3bx^4y$ is the contracted form of a symmetric polynomial in $Z[x,y,z]$.

```
pc : 2*a^3*b*x^4*y$
```

```
psym : explode(pc,[x,y,z]);
```

```

      3      4      3      4      3      4
2 a  b y z + 2 a  b x z + 2 a  b y z
      3      4      3      4      3      4
+ 2 a  b x z + 2 a  b x y + 2 a  b x y

```

If we use the function `contract` we find again the contracted form:

```
contract(psym,[x,y,z]);
      3      4
      2 a b x y
partpol(psym,[x,y,z]);
      3
      [[2 a b, 4, 1]]
ppart : cont2part(pc,[x,y,z]);
      3
      [[2 a b, 4, 1]]
part2cont(ppart,[x,y,z]);
      3      4
      2 a b x y
```

2.2 The partitions

- `kostka(part1,part2)` (written by P.ESPERET) gives the Kostka number associated with the partitions `part1` et `part2`.
- `treinat(part)` → list of the partitions that are less than the the partition `part` in the natural order and that are of the same weight.
- `treillis(n)` → list of the partitions of weight equal to n .
- `lgtreillis(n,m)`
- `lgtreillis(n,m)` → list of the partitions of weight equal to n and of length equal to m .
- `ltreillis(n,m)`
- `lgtreillis(n,m)` → list of the partitions of weight equal to n and the length less than or equal to m .

2.3 The orbits

- `orbit(p(x1,...,xn),lvar)` → $O_{S_n}(p)$
gives the list of polynomials of the orbit $O_{S_n}(p)$ where the n variables of p are in the list `lvar`. This function does not consider the possible symmetries of p .
- `multi_orbit(p,[lvar1,lvar2,...,lvarp])` → $O_{S_D}(p)$
gives the orbit of p under S_D (see above), where the variables of the multi-alphabet X are in the lists `lvari`.

```
orbit(a*x+b*y,[x,y]);
      [a y + b x, b y + a x]
```

```
orbit(2*x+x**2,[x,y,z]);
      2      2      2
      [z + 2 z, y + 2 y, x + 2 x]
```

```
multi_orbit(a*x+b*y,[[x,y],[a,b]]);
```

```

      [b y + a x, a y + b x]
multi_orbit(x+y+2*a, [[x,y], [a,b,c]]);

[y + x + 2 c, y + x + 2 b, y + x + 2 a]

```

2.4 Contracted product of two symmetric polynomials

The formula is in [V1] or [V2] and the proof in [V2].

- `multsym(ppart1, ppart2, card) → ppart`
The arguments are two partitioned forms and the cardinality. The result is in partitioned form.

For example, take two symmetric polynomials in their contracted form. We first compute the product in the standard way and then with the function `multsym`. We are in $\mathbb{Z}[x, y]$ and the two contracted forms `pc1` and `pc2` are associated with the two symmetric polynomials `p1` and `p2`.

```

pc1 : x^2*y$
pc2 : x$

p1 : explode(pc1, [x,y]);
      2    2
      x y + x y

p2 : explode(pc2, [x,y]);
      y + x

```

There is the product of the two symmetric polynomials with the standard operation of MAC-SYMA:

```

prod : expand(p1*p2);
      3      2 2    3
      x y + 2 x y + x y

```

we verify below that this is the extended form of the product obtained with the function `multsym`:

```

contract(prod, [x,y]);
      3      2 2
      x y + 2 x y

ppart1 : cont2part(pc1, [x,y])$
ppart2 : cont2part(pc2, [x,y])$

part2cont(multsym(ppart1, ppart2, 2), [x,y]);
      2    2 3
      2 x y + x y

```

2.5 Change of basis

The monomial forms $M_I(X)$ where I varies in the set of partitions of length $\leq n$ are an $\mathcal{A}$ -base of the free $\mathcal{A}$ -module $R_D^{\mathcal{A}}$. The Schur functions also form an $\mathcal{A}$ -base of the free $\mathcal{A}$ -module.

We have the following algebra bases: the elementary symmetric functions ($\mathcal{A}$ -base), the power functions ($\mathcal{Q}$ -base if $\mathcal{A} = \mathbb{Z}$) and the complete functions ($\mathcal{A}$ -base).

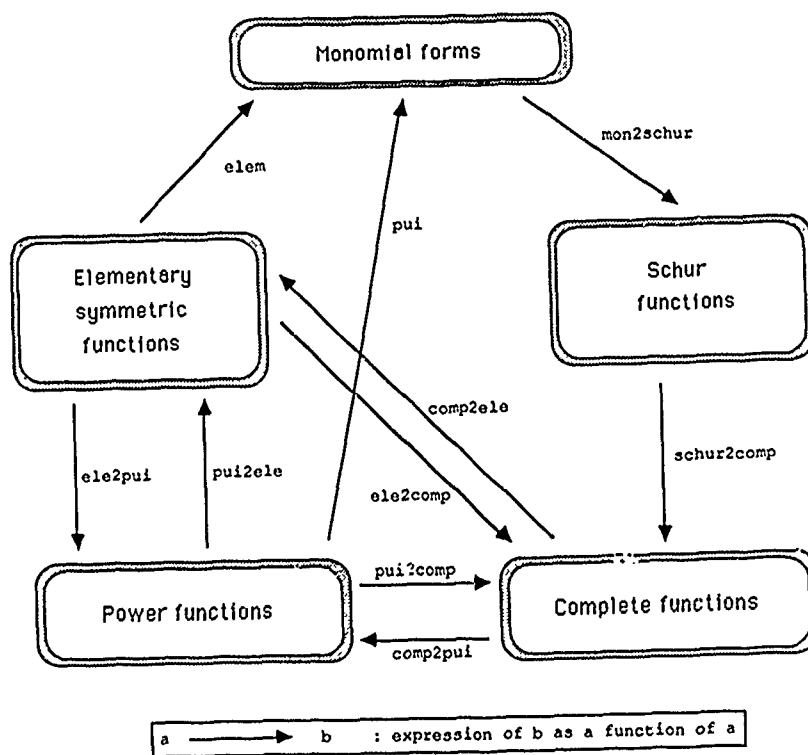
When $I = (\overbrace{1, 1, \dots, 1}^i, 0, 0, \dots, 0)$ where $0 \leq i \leq n$, the monomial form $e_i(X) = M_I(X)$ is also called the i^{th} elementary symmetric function over X , with the convention $e_0 = 1$ and $e_i = 0$ for $i > n$. When $I = (i)$, the monomial form $p_i(X) = M_I(X) = \sum_{x \in X} x^i$ is called the i^{th} power function over X , ($p_0 = n$). The i^{th} complete symmetric function, $h_i(X)$, is the sum of the monomial forms $M_I(X)$ where the weight of I is i , ($h_0 = 1$ and $h_r = 0$ if $r < 0$).

Let M be a matrix and $I = (i_1, i_2, \dots, i_n)$ a sequence of $\mathbb{Z}^n$; let M_I be the minor of M constructed with the lines $1, 2, \dots, n$ and the columns $i_1 + 1, i_2 + 2, \dots, i_n + n$, with the convention $M_I = 0$ if there exists r such that $i_r + r \leq 0$.

Let $S = (h_{i-j})_{i \geq 1, j \geq 1}$ be an infinite matrix:

$$\begin{pmatrix} h_0 & h_1 & h_2 & h_3 & \dots \\ 0 & h_0 & h_1 & h_2 & \dots \\ 0 & 0 & h_0 & h_1 & \dots \\ 0 & 0 & 0 & h_0 & \dots \\ 0 & 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

where the h_i are the complete functions. We call Schur function of index I the minor S_I .



- `elem(l_cele,sym,lvar) → P(e1,...,eq)`
decomposes the symmetric polynomial `sym` into the elementary symmetric functions with the algorithm in [V1].
- `multi_elem([l_cele1,...,l_cele_p],multi_pc,[lvar1,...,lvar_p]) → P(l_cele1,...,l_cele_p)`

We have the multi-symmetric polynomial `multi_pc` in its contracted form. This function decomposes successively in each package `l_cele_i` of elementary symmetric functions of the alphabet $x^{(i)}$ (see the section 1.2).

- `pui(l_cpui,sym,lvar) → P(p1,...,pq)`
decomposes the symmetric polynomial `sym` into the power functions with the algorithm in [V1].
- `multi_pui([l_cpui1,...,l_cpui_p],multi_pc,[lvar1,...,lvar_p]) → P(l_cpui1,...,l_cpui_p)`
see `multi_elem`.

In this examples the symmetric polynomials are in contracted form.

```
elem([],x**4 - 2*y*z,[x,y,z]);
      4      2      2
e1  - 4 e2 e1  + 4 e3 e1 + 2 e2  - 2 e2 - 4 e4
```

If the cardinality is 3 we have:

```
elem([3],x**4 - 2*y*z,[x,y,z]);
      4      2      2
e1  - 4 e2 e1  + 4 e3 e1 + 2 e2  - 2 e2
```

If the cardinality is 3 and $e_1 = 7$ we have:

```
elem([3,7],x^4-2*x*y,[x,y]);
      2
28 e3 + 2 e2  - 198 e2 + 2401
```

For the power functions we know that if the cardinality of the alphabet X is n then the i^{th} power function, $i > n$, depends algebraically on the $p_1, p_2, \dots, p_n$. For this reason, when the function `pui` completes the list `l_cpui` with formal values and the degree of the polynomial `sym` is greater than n , the i^{th} power functions for $i > n$ do not appear. For this computation the function `pui` uses the function `puieduc` (see below).

For following formulas for the change basis we refer to [Macdonald] and [Lascoux, Schützenberger].

- `ele2pui(m,l_cele) → l_cpui`
gives the first m power functions as functions of the elementary symmetric functions with the [Girard]-Newton formulas.
- `pui2ele(n,l_cpui) → l_cele`
gives the elementary symmetric functions when we know the power functions. If the flag `pui2ele` is `girard`, the result is the first n elementary symmetric functions and if its is `close`, the result is the n^{th} elementary symmetric function.

In the following example we find the first 3 elementary symmetric functions when the power functions are generic.

pui2ele(3,[]);

$$[3, p_1, \frac{p_1^2}{2} - \frac{p_2}{2}, \frac{p_2^2}{3} - \frac{p_1 p_2}{2} + \frac{p_1^3}{6}]$$

Now the cardinality of the alphabet X is 3 and the first power function is equal to 2. We remark that the 4th elementary symmetric function is zero, because the cardinality is 3. We compute the first three power functions below.

pui2ele(4,[3,2]);

$$[3, 2, \frac{4 - p_2}{2}, \frac{p_3 - 3 p_2 + 4}{3}, 0]$$

ele2pui(3,[]);

$$[3, e_1, e_1^2 - 2 e_2, 3 e_3 - 3 e_1 e_2 + e_1^3]$$

In the next example, since the cardinality is 2, the 3th elementary symmetric function is zero:

ele2pui(3,[2]);

$$[2, e_1, e_1^2 - 2 e_2, e_1^3 - 3 e_1 e_2]$$

- $\text{puireduc}(n, \text{l\_cpui}) \rightarrow [\text{card}, p_1, p_2, p_3, \dots, p_n]$
gives the first n power functions when the first n are known. The cardinality is the first element of l_cpui .

In this example $\text{card} = 2$ and we search the first three power functions. We can give numerical values to the first two power functions in the list l_cpui .

puireduc(3,[2]);

$$[2, p_1, p_2, \frac{3 p_1 p_2}{2} - \frac{p_1^3}{2}]$$

- $\text{ele2comp}(m, \text{l\_cele}) \rightarrow \text{l\_ccomp}$
gives the first m complete functions as functions of the elementary symmetric functions.
- $\text{pui2comp}(n, \text{l\_cpui}) \rightarrow \text{l\_ccomp}$
gives the first m complete functions as functions of the power functions.
- $\text{comp2ele}(n, \text{l\_ccomp}) \rightarrow \text{l\_cele}$
gives the first m elementary symmetric functions as functions of the complete functions.
- $\text{comp2pui}(n, \text{l\_ccomp}) \rightarrow \text{l\_cpui}$
gives the first m power functions as functions of the complete functions.
- $\text{mon2schur}(\text{liste}) \rightarrow \text{pc}$
compute the Schur functions as functions of the monomial forms. The list appearing as argument is a p -uple I of integers, it represents S_I , the Schur function of index I (see Section 1.2).

- `schur2comp(P, [h1, ..., hq])` → list of lists

The polynomial P is $A[h_1, \dots, h_q]$, where the h_i are the complete functions. This function expresses P as a function of the Schur functions, denoted by S_I in MACSYMA. It is imperative to express the complete functions by means of a letter h "concatenated" with an integer (ex: h_2 or h_5).

We first verify that the Schur function $S_{(1,1,1)}$ is equal to the third elementary symmetric function and that $S_{(3)}$ is equal to the third complete function (this is a general result).

```
mon2schur([1,1,1]);
```

$$x_1 x_2 x_3$$

```
mon2schur([3]);
```

$$x_1^2 x_2^2 x_3^2 + x_1^3 x_2^3 + x_1^3 x_3^3$$

```
mon2schur([1,2]);
```

$$2 x_1^2 x_2^2 x_3^2 + x_1^2 x_2^2$$

Let us see on two example, how with a circular set of operations we can go back to the initial lists $[3, p_1, p_2, p_3]$ and $[3, h_1, h_2, h_3]$.

```
a1 : pui2comp(3, [3]);
```

$$[3, p_1, \frac{p_2^2}{2} + \frac{p_1^2 p_3}{2}, \frac{p_3^2}{2} + \frac{p_1 p_2^2}{2} + \frac{p_1^3}{6}]$$

```
a2 : comp2ele(3, a1);
```

$$[3, p_1, \frac{p_1^2}{2} - \frac{p_2^2}{2}, \frac{p_3^2}{2} - \frac{p_1 p_2^2}{2} + \frac{p_1^3}{6}]$$

```
a3 : ele2pui(3, a2);
```

$$[3, p_1, p_2, p_3]$$

```
a4 : comp2pui(3, []);
```

$$[3, h_1, 2 h_2 - h_1^2, 3 h_3 - 3 h_1 h_2 + h_1^3]$$

```
a5 : pui2ele(3, a4);
```

$$[3, h_1, h_1^2 - h_2, h_3 - 2 h_1 h_2 + h_1^3]$$

```
a6 : ele2comp(3, a5);
```

$$[3, h_1, h_2, h_3]$$

In the next example we show how to express a Schur function through the monomial forms (see the label c48), and then through the complete functions (c50), the elementary symmetric functions (c51) and the power functions (en c52).

```

(c48) mon2schur([1,2]);
      2
(d48) 2 x1 x2 x3 + x1 x2

(c49) comp2ele(3,[]);
      2      3
(d49) [3, h1, h1 - h2, h3 - 2 h1 h2 + h1 ]

(c50) elem(d49,d48,[x1,x2,x3]);

(d50) h1 h2 - h3

(c51) elem([],d48,[x1,x2,x3]);

(d51) e1 e2 - e3

(c52) pui([],d48,[x1,x2,x3]);
      3
      p1 p3
(d52) --- - --
      3 3

(c53) schur2comp(h1*h2-h3,[h1,h2,h3]);

(d53) s
      1, 2

(c54) schur2comp(a*h3,[h3]);

(d54) s a
      3

```

In the last instruction we have obtained the polynomials $h_1 h_2 - h_3$ and h_3 on the basis of the Schur functions.

2.6 Direct images

In this section, we apply the previous functions to the transformations of polynomial equations.

The direct image intruces in $[G,L,V]$ or in $[V2]$ can represente the The resultant (function `resultsym`), the resolvents, such as Galois or Lagrange's (see $[V2]$ chapter 9 p.91), or more generally minimal polynomials, can be seen as *direct images* ($[G,L,V]$ and $[V2]$). Suppose that A is a field k . Let f be a function in R_D and let $P_1, P_2, \dots, P_p$, p polynomials of degrees $d_1, d_2, \dots, d_p$, respectively. Associate with each P_j the set $(a_1^{(j)}, \dots, a_{d_j}^{(j)})$ of its d_j roots in an algebraic closure K of k in an arbitrary order ($1 \leq j \leq p$), and choose an evaluation map $E_a : R_D \rightarrow K$ which is an algebra homomorphism which sends the variable $x_i^{(j)}$ to $a_i^{(j)}$. The *direct image* $f_*(P)$ is the univariate polynomial whose roots are the images under the map E_a of the elements of the f -orbite $O_{S_D}(f)$, i.e.:

$$f_*(P)(x) = \prod_{g \in O_{S_D}(f)} (x - E_a(g)).$$

• `resultsym(p,q,x) → resultant(p,q,x)`

Computes the resultant of the two polynomials, p and q , using changes of basis on

symmetric functions. The computation is not symmetric in p and q . The computing time is best when the degree of p is less than the degree of q .

- $\text{direct}([P_1, P_2, \dots, P_p], y, f, [lvar_1, lvar_2, \dots, lvar_p]) \rightarrow f(P_1, P_2, \dots, P_p)(y)$
where the lists $lvar_i$ representing X (see p.3) allow us to find the type of the function f .

We now compute the direct image in two different ways. The first one uses, at the top level, the previous functions in order to obtain the elementary symmetric functions of the roots of the polynomial given by the function `direct` (which is the second way). We can change the flag `direct`. If it is `puissances` (the default value) the function `direct` uses the function `multi_pui`. If it is `elementary`, the function `direct` uses the function `multi_elem` (generally less efficient).

```
l : multi_direct(multi_orbit(a*x+b*y, [[x,y],[a,b]]), [[x,y],[a,b]]);
      2      2
      [a x, 4 a b x y + 4 b x ]

m : multi_elem([[2,e1,e2],[2,f1,f2]], l[1], [[x,y],[a,b]]);
      e1 f1

n : multi_elem([[2,e1,e2],[2,f1,f2]], l[2], [[x,y],[a,b]]);
      2      2      2      2
      8 e2 f2 - 2 e1 f2 - 2 e2 f1 + e1 f1

pui2ele(2, [2,m,n]);
      2      2
      [2, e1 f1, - 4 e2 f2 + e1 f2 + e2 f1 ]

direct([z^2 - e1*z + e2, z^2 - f1*z + f2], z, b*v + a*u, [[u,v],[a,b]]);
      2      2      2
      z - e1 f1 z - 4 e2 f2 + e1 f2 + e2 f1
```

The coefficients of this polynomial in z are equal (up to a sign) to the elementary symmetric functions obtained before.

- $\text{somrac}(l\_ele, k) \rightarrow P(e_1, \dots, e_n)(x)$
gives the polynomial whose roots are the sums k by k of the roots of p . The polynomial p is represented by the elementary symmetric functions of these roots, listed in `l_ele`. Here the list `l_ele` cannot be completed by formal values. If the flag `somrac` is `pui` (default value), the function `somrac` uses the function `pui`, and if it is `elem` then it uses the function `elem`.
- $\text{prodrac}(l\_ele, k)$ is the same function, but here we transform the polynomial using a product instead of the sum.

We remark that these functions are special cases of direct images. For example, take the polynomial $x^4 - x^3 - 25x^2 + 25x$:

```
somrac([1,-25,-25,0], 2);
      6      5      4      3      2
      x - 3 x - 47 x + 99 x + 550 x - 600 x
```

Let k be a field.

```
pui_direct ([b*y + a*x, a*y + b*x],[[x,y],[a,b]]);
           2 2
           [a x, 4 a b x y + 4 x ]
```

[illegible]

References

- [Andrews] 1976, George E. Andrews, *The theory of partitions*, Encyclopedia of Mathematics and its Applications, Vol. 2, Section Number Theory, Addison Wesley.
- [Girard], 1629, *Invention Nouvelle en Algèbre*, Amsterdam.
- [G,L,V] 1988, M. Giusti, D. Lazard, A. Valibouze, *Symmetric polynomials and elimination*, Notes informelles de Calcul Formel IX, Prépublication du Centre de Mathématiques de l'Ecole Polytechnique, M810.0987.
- [G,L,V] 1988, M. Giusti, D. Lazard, A. Valibouze, *Algebraic transformation of polynomial equations, symmetric polynomials and elimination*, Proceedings of ISSAC-88 (Roma, Italy), Springer-Verlag.
- [Lascoux], Alain, 1984-1985. *La résultante de deux polynômes*, Séminaire d'Algèbre M.P. Malliavin.
- [Lascoux, Shützenberger] 1985, A. Lascoux, M.P. Schützenberger, *Formulaire raisonné de fonctions symétriques*, L.I.T.P, U.E.R. MATHS, Paris 7, L.A. 248.
- [Macdonald], I.G., 1979, *Symmetric functions and Hall polynomials*, Clarendon Press, Oxford.
- [V1] 1897, A. Valibouze, *Fonctions symétriques et changements de bases*, Proceedings of the European Conference on Computer Algebra EUROCAL '87 (Leipzig, RDA), Springer-Verlag.
- [V2] 1988, A. Valibouze, *Manipulations de fonctions symétriques*, Thèse de l'Université Paris VI.

SIMULTANEOUS COMPUTATIONS IN FIELDS OF DIFFERENT CHARACTERISTICS

Dominique DUVAL
Faculté des sciences
Laboratoire de théorie des nombres et algorithmique
F-87060 LIMOGES Cedex, FRANCE

Introduction— This paper presents new software for computing simultaneously in fields of different characteristics. Such computations are made possible thanks to:

- The computer algebra system Scratchpad 2 [Je], especially its wide genericity features.
- And the *dynamical evaluation principle* : It generalizes traditional evaluation and was first used to deal with algebraic numbers [D-D-D, D-D-1].

The *prime fields* are the field $\mathbb{Q}$ of rational numbers and the finite fields $\mathbb{F}_p$ with p elements for every prime number p . It is known that every field K contains one and only one prime field. The *characteristic* of K is the integer 0 if K contains $\mathbb{Q}$, and it is p if K contains $\mathbb{F}_p$.

It is possible to build a Scratchpad domain in the *Field* category (in the Scratchpad sense, i.e. essentially with equality test and the four elementary operations) which represents several "usual" fields. For example the *DDPrimeField* domain represents all the prime fields (the "D"'s here and there are the "trademark" for the dynamical evaluation features which are developed in Scratchpad). It means that a result obtained in *DDPrimeField* is valid in $\mathbb{Q}$ as well as in any of the $\mathbb{F}_p$'s.

But the result of a given computation may depend on the characteristic: For example, the boolean value of $12 = 0$ is *true* in characteristic 2 and *false* in every other characteristic. As a consequence, if a domain in the *Field* category represents fields of several characteristics, it must be possible for a function sqfr? at domain to return several values, depending on the characteristic. This paper explains how it is possible, and easy to use.

Prime fields— Let us begin with a simple example. Let K denote some domain in the *Field* category, and *POLY* the domain $UP(X, K)$ of univariate polynomials in X with coefficients in K . Then the function *sqfr?*, from *POLY* to *Boolean*, defined by

$$\text{sqfr?}(\text{poly}) == \text{if } \text{degree}(\text{gcd}(\text{poly}, \text{deriv}(\text{poly}))) = 0 \text{ then true else false}$$

tests whether its argument *poly* is squarefree or not. It can be applied with $K = \mathbb{R}\mathbb{N}$ (the field of rational numbers) or any "usual" field which can be constructed in Scratchpad. It can also be used with $K = \text{DDPrimeField}$ (abbreviated *DDPF*):

```

POLY:= UP(X,DDPF)
sqfr?(poly:POLY):B == if degree(gcd(poly,deriv(poly)))=0 then true
                        else false

poly0:POLY:= X**12+1
allCases(sqfr?,poly0)$DDCP(POLY,B)

[ value is false in case characteristic divides 12,
  value is true in case characteristic divides 11,
  value is true in case characteristic does not divide 12*11 ]

```

Note that we have not called $sqfr?(poly0)$, as in an usual field, but $allCases(sqfr?,poly0)$. The function $allCases$ is taken from the *DDControlPackage* (abbreviated *DDCP*). The arguments of this package respectively are the domain and the codomain of the function $sqfr?$. The result is that the value of $sqfr?(poly0)$ depends on the characteristic of the field. Precisely, it is *false* if the characteristic divides 12 (i.e. if it is 2 or 3), and it is *true* in every other case.

The package *DDCP* has two parameters $D1$ and $D2$, which may be any *Set*. It contains the function $allCases$. This function has two arguments : a function f from $D1$ to $D2$, and an element x of $D1$. The value of $allCases(f,x)$ is a finite list of *results*. Each result is made of a *value* (of type $D2$) and of the *case* where this result is valid. A case here is a set of prime fields, which are described by their characteristics. The list of results is always complete (every prime field appears in one of the cases) and irredundant (a given prime field appears in only one of the cases).

The generic and degenerate cases— What happens if you forget about $allCases$, and ask for $f(x)$? Let's try :

```

sqfr?(poly0)

true

```

You just get the *generic case*. Actually, starting from *DDPF* which represents any prime field, the list of results is always made of one *generic case* and of some *degenerate cases*. Each degenerate case represents a finite number of prime fields, while the generic case represents an infinite number of them. The field $\mathbb{Q}$ always is in the generic case.

You may like to know more precisely what is the generic case for your computation of $f(x)$. Then ask for *currentDcase* :

```

currentDcase()$DDPF

characteristic does not divide 12*11

```


However, you must be aware that the degeneracy may have different reasons : either mathematical reasons (like the case *characteristic divides 12* in the example above), or just opportunity reasons due to the way the computation is performed (like the case *characteristic divides 11*).

Remark— It will become possible, some day, to issue $f(x)$ instead of $allCases(f,x)$. Some “flag” will then allow the computation of the generic case only.

Remark— When computing with univariate polynomials, you may avoid a lot of un-significant degenerate cases by using the *DDUnivariatePolynomial* domain constructor (abbreviated *DDUP*). It is similar to the standard *SparseUnivariatePolynomial* constructor, but when possible in the algorithms the equality tests between coefficients are replaced by “rough” equality tests. Two polynomials $p1$ and $p2$ in *DDUP* are said *eqRough* if they are equal in *every* prime field in the current case (defined below). This is very simple to test and cannot lead to a splitting. For example, in the division of some $p1$ by some $p2$ in *DDUP*, the test $p2 = 1$ which is only used to fasten the division is replaced by $p2$ *eqRough* 1, but the test $p2 = 0$ remains unchanged, since it ensures that one does not divide by 0. Our first example then has only one degenerate case :

```
DPOLY:= DDUP(DDPF)
X:= varPol()$DPOLY
dsqfr?(dpoly:DPOLY):B == if deg(gcd(dpoly,deriv(dpoly)))=0 then true
                        else false
dpoly0:DPOLY:= X**12+1
allCases(dsqfr?,dpoly0)$DDCP(DPOLY,B)

[ value is false in case characteristic divides 12,
  value is true in case characteristic does not divide 12 ]
```

The current case— Let us consider the evaluation of $sqfr?(poly0)$ above. Before the evaluation, *DDPF* represents a prime field of any characteristic. But after the evaluation, it represents a prime field of any characteristic not dividing 11×12 . We say that the *current case* for *DDPF* is first *any characteristic* and then *characteristic does not divide 11×12* . At every moment, the function *currentDcase()* returns the value of the current case. It is also possible to use the function *dchar()*, described below.

What happened is that, during the evaluation, the system was asked whether $12 = 0$ or not. It led to a *splitting* of the case *any characteristic* in two subcases : either *characteristic does not divide 12*, and the answer is *false*, or *characteristic divides 12*, and the answer is *true*. The computations went on in the generic case, until the question $11 = 0$ was met, which led to a second splitting.

The management of the different cases is similar for every application of dynamical evaluation. It is currently performed at Scratchpad level, in the *DDControlPackage*. This package is made of an external function *allCases* and an internal function *oneCase*.

Starting from any current case ca , the call of $oneCase(f, x)$ returns the value y of $f(x)$ in some subcase ca' of ca , together with this subcase ca' , and the list lca of the subcases which have been left aside during the computation (adjoining ca' to lca gives a partition of ca).

The function $allCases(f, x)$ manages the list lct of cases to be treated, as well as the list lr of results yet obtained. To begin with, lct has the current case as unique element, and lr is empty. Until lct is empty, the function $allCases$:

- Sets the current case to the first element ca of lct ;
- Removes this first element ca from the list lct ;
- Calls $oneCase(f, x)$ which gives some y , ca' , and lca , as above;
- Adds lca to lct ;
- And adds the pair (y, ca') to lr .

If the program terminates (but it is easy to imagine tricky examples where it does not), the list lr of results is returned, with each result formatted

value is ... in case ...

as in the examples presented in this paper.

Selection of some characteristics— You may want $DDPF$ to represent only *some* prime fields. For example, assume you want to perform the same computation over the finite fields $GF(p)$ for p prime less than 20, but not over the other prime fields. The product of the primes less than 20 is 9699690. You have two possibilities for your purpose : You may issue either $setDchar(9699690)$ (I want the characteristic to be any prime divisor of 9699690) or $areEqual(9699690, 0)$ (I want the element 9699690 in my $DDPF$ to be equal to 0).

Here is a very simple example of a modular computation of gcd's, proving that the given polynomials are coprime over $\mathbb{Q}$ [D-S-T]. Since the current $allCases$ function requires a *univariate* function as first argument, we must build the domain of pairs of polynomials :

```
poly1:POLY:= X**2+1
poly2:POLY:= X+1
PAIR:= Record(p1:POLY,p2:POLY)
gcdpair(pair:PAIR):POLY == gcd(pair.p1,pair.p2)$POLY
setDchar(9699690)$DDPF
pair0:= [poly1,poly2]
allCases(gcdpair,pair0)$DDCP(PAIR,POLY)
```

```
[ value is X+1 in case characteristic divides 2,
  value is 1 in case characteristic divides 4849845 ]
```

You may, on the contrary, want to avoid some characteristics. For example because they correspond to exceptional cases that you do not want to consider. In books about elliptic curves are assertions like "let K denote a field of characteristic different from 2 or

3". Similarly, you may then use either *setDchar*(-2 * 3) or *areDifferent*(2 * 3, 0).

You may also use *setDchar*(0) if you want to compute over the field of rational numbers, but then the standard *RationalNumber* field will probably be more efficient. This case appears as *characteristic* = 0. It cannot be reached from any other case by computations in *DDPF*.

More interesting is *setDchar*(1) which makes the current case to be *any characteristic*. It is the default choice when you call *DDPF*, but it may get changed after some computation.

The characteristic functions— A *Field* in Scratchpad must be endowed with a *characteristic* function, with no arguments and with value a non-negative integer. In the *DDPF* field, the value of this function is usually an error message. Except if the current case corresponds to exactly one characteristic : You get the value 0 if the current case is *characteristic* = 0, and the value *p* if the current case is *characteristic divides p* for some prime number *p*.

In addition, in *DDPF* is a function called *dchar*. It has no argument, its value is a rational integer, and it never returns an error message. The value of *dchar*() is

- 0 in case *characteristic* = 0,
- 1 in case *any characteristic*,
- *n* (> 1) in case *characteristic divides n*,
- and -*n* (*n* > 1) in case *characteristic does not divide n*.

```
12 =$DDPF 0
    false
dchar()$DDPF
    -12
setDchar(1)$DDPF
    true
dchar()$DDPF
    1
```

Non-prime fields— From the *DDPrimeField* domain, it is possible to build other fields of arbitrary characteristic. For example the field of rational functions of one variable over a prime field is obtained from univariate polynomials with the standard Scratchpad *QuotientField* domain constructor :

```
FRAC:= QF(POLY)
divide(pair:PAIR):FRAC ==
    pair.p2 =$POLY 0 => 0$FRAC
```

```

pair.p1 /$FRAC pair.p2
allCases(divide,pair0)$DDCF(PAIR,FRAC)

[ value is X-1 in case characteristic divides 2,
  value is  $\frac{X^2+1}{X+1}$  in case characteristic does not divide 2 ]

```

This gives access to purely transcendental extensions of *DDPF*. For algebraic extensions, the standard *SimpleAlgebraicExtension* domain constructor is generally difficult to use: This constructor requires an irreducible polynomial as argument, so that you must restrict the characteristic in order to ensure the irreducibility. For example, if you want to build the extension of *DDPF* by a root of $X^2 + 1$, you have to restrict to the characteristics 0 or p with $p \neq 0$ and such that -1 is not a square modulo p , i.e. $p \equiv 3 \pmod{4}$ [Sa]. But there is an infinite number of such primes, and also an infinite number of primes p such that $p \not\equiv 3 \pmod{4}$. As a consequence, it is impossible with the *DDPF* domain to restrict to every possible prime field where $X^2 + 1$ is irreducible.

But actually this does not matter, since Scratchpad will soon be able to construct the extension of a given field by any root of a given polynomial, irreducible or not [D-D-2]. It will then be possible to compute in a large number of fields of arbitrary characteristic.

Conclusion— The general prime field described here is an example of *dynamical evaluation* [D-R]. It consists in handling *cases*, each case corresponding to several domains in some category (and even to an infinite number of such domains). When it is needed during the computation, a case *splits* in two disjoint *subcases*. The initial case is then forgotten, and the two subcases are handled *in parallel*. Of course, they may themselves later split, and so on. This method was initiated by the *D5* system for computations with algebraic numbers in Reduce. It happens that Scratchpad is better suited than the “classical” computer algebra systems for the development of the numerous possible applications of dynamical evaluation, among which are the features described above.

References—

- [D-S-T] J.H.Davenport, Y.Siret, E.Tournier — *Calcul formel*, Masson, 1986
- [D-D-D] J.Della Dora, C.Dicrescenzo, D.Duval — *About a new method for computing in algebraic number fields*, LNCS 204, p.289-290, 1985
- [D-D-1] C.Dicrescenzo, D.Duval — *Computations with algebraic numbers : The D5 method*, Submitted to publication
- [D-D-2] C.Dicrescenzo, D.Duval — *Algebraic extensions and algebraic closure in Scratchpad 2*, Proc.ISSAC'88, to appear, 1988
- [D-R] D.Duval, J.-C.Reynaud — *Esquisses et calcul formel*, In preparation
- [Je] R.D.Jenks — *A primer : 11 keys to new Scratchpad*, LNCS 174, p.123-147, 1984
- [Sa] P.Samuel — *Théorie algébrique des nombres*, Hermann, 1971