ISSAC'99 Poster Abstracts: Session I

Communicated by Eugene Zima
Symbolic Computation Group,
University of Waterloo,
Waterloo, Canada
ezima@daisy.uwaterloo.ca

Here we present the ISSAC 1999 poster abstracts. The abstracts are as distributed at the conference. This is the first of two parts, containing abstracts of the first poster session. Eugene V. Zima, Mohamed O. Rayes.

A Comparison of Symbolic Solution of Radioactive Decay Chains Using MATHEMATICA

Aslam, Rao F. H. Khan and N. Ahmad
Pakistan Institute of Engineering and Applied Sciences,
P.O. Nilore, Islamabad, Pakistan

Different analytical methods available for the solution of radioactive decay chains have been implemented in Mathematica. The implemented algorithms include the method of dagonalization, the Laplace transformation, Matrix Exponential Method, Bateman solution and the E formulation. These methods have been programmed in Mathematica by defining the functions for each method separately. These functions require only the decay constants.There is no limitation on the number of members of decay chains. Each algorithm has been tested for different sets of decay constants. These methods differ only in execution time. Laplace transformation is the slowest method while Bateman and E formulation proves to be the fastest ones. Method of diagonalization and forward sequential method, both use built-in function `DSovle`, take comparable time for execution. Matrix Exponential Method is also less time consuming in comparision to Forward sequential and Method of diagonalization, however it fails in the case of singularities. This failure may be prevented by avoiding the singularity. The methods may be extended to the chains where branching also occurs.

Computing the Galois Group of $y^{(3)} + ay' + by = 0,\ a, b \in \mathcal{C}[x]$

Peter Berman
Department of Mathematics
North Carolina State University
phberman@eos.ncsu.edu

We describe an algorithm to compute the Galois group of a differential equation of the form $y^{(3)} + ay' + by = 0$, $a, b \in \mathcal{C}[x]$, where $\mathcal{C}$ is a computable, algebraically closed constant field of characteristic zero (e.g., the algebraic numbers). This algorithm applies the approach used in [M. van der Put, *Symbolic Analysis of Differential Equations*, Some Tapas of Computer Algebra (Cohen et. al., ed.), Springer Verlag, 1998] to study order-two equations with one or two singular points.

Our algorithm relies on a result of Ramis that says that the group of such an equation must be connected and have *defect zero* (c.f., [C. Mitschi and M. Singer, *Connected Linear Groups as Differential Galois Groups*, Journal of Algebra 184 (1996), 333-361] and [M. van der Put, *Recent Work on Differential Galois Theory*, Seminaire Bourbaki Vol. 1997-98, Asterisque 252, 1998]). Using this result we give a complete list of conjugacy classes of all subgroups of $\mathrm{SL}_3(\mathcal{C})$ that occur as Galois groups of equations of the prescribed form. For each group we give a simple test based on finding rational solutions of Riccati equations of associated equations.

A Java Framework for Massively Distributed Symbolic Computing

Laurent Bernardin
Institute for Scientific Computing
ETH Zurich, Switzerland
bernardin@inf.ethz.ch

1 Introduction

We present a framework for implementing massively distributed applications in symbolic computing. Using this framework, computations with massive resource requirements can be distributed and processed in parallel on a network of workstations [1] or on a large scale network such as the Internet [3]. For each concrete application only minimal code is needed to complement the generic framework in order to enable large-scale distributed processing of the application.

The use of Java[2] is essential in order to allow secure distributed computations on a set of heterogenous clients that do not need to be known a priori.

2 The "Divide and be Conquered" Model

The model we advocate in this paper eliminates the burden of managing a pool of workers. The user advertises his need for computational power along with the data set to compute on. Any machine on the Internet that sees this advertisement can choose to act as a worker for this particular task, compute on the corresponding data and return results to the principal for that computation. One could say that the workers "push" their computing power to the principal or that the workers "shop around" on the Internet, looking for suitable tasks to compute on. We call this model "divide and be conquered", since a large computation is broken up into small subtasks and the initiative for processing these subtasks is coming from the workers.

This approach scales to a large number of hosts involved in a computation, since the principal, which is originating and coordinating a computation, needs no a priori knowledge of the workers which will be involved. In particular, the managing node does not (need to) know which machine is going to act when and for how long as a worker for his task. The hassle of maintaining a dedicated pool of workers disappears.

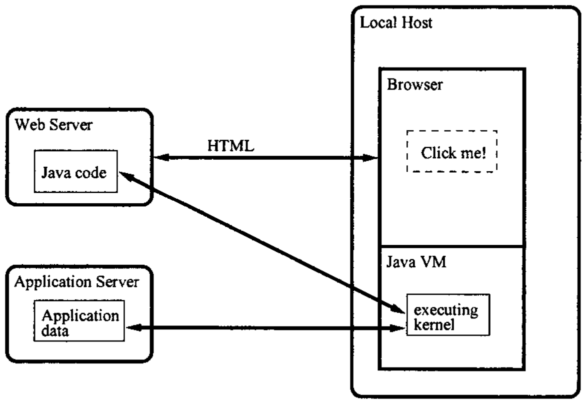

Figure 1: The "divide and be conquered" model of distributed computing

3 Conclusions

We present a new alternative for distributed symbolic computing on the Internet. The use of the Java language provides us with a flexible generic framework using a small amount of code. Certain large scale computations can take advantage of this model to advance the class of tractable problems today.

References

[1] Anderson, T., Culler, D., and Patterson, D. A case for NOW (networks of workstations). *IEEE Micro 15*, 1 (1995), 54–64.

[2] Gosling, J., Joy, B., and Steele, G. *The Java Language Specification.* Addison-Wesley, 1996.

[3] Strumpen, V. Coupling hundreds of workstations for parallel molecular sequence analysis. *Software – Practice and Experience 25*, 3 (1995), 291–304.

CtCoq: an environment for mathematical reasoning

Lemme Group
Yves Bertot, Laurence Rideau, Looc Pottier, and Laurent Thiry
INRIA Sophia Antipolis
`{Yves.Bertot,Laurence.Rideau}@inria.fr`,
`{Loic.Pottier,Laurent.Thery}@inria.fr`

The work we want to present on this poster derives from the experience we have developed on providing user-interfaces for deduction systems. The latest developments concern a better display of mathematical formulae, possibilities for illustrations, and rich interaction capabilities.

1 What is it?

CtCoq is an independent application designed to help perform proofs using the Coq proof assistant. It takes care of recording the commands sent to the Coq proof engine and it provides a better display for the mathematical formulas constructed by the user or produced by the proof engine in response to requests.

Through an analysis of mathematical formulas and mouse gestures, it also provides novel ways to perform mathematical reasoning.

We intend to have a little paragraph explaining this multi-process architecture (and maybe a drawing showing the data flow between the processes).

2 Interaction features

2.1 Displaying Mathematics

We have an algorithm that takes care of the layout of mathematical formulae in two dimensions, thus making the display of square roots, summations, fractions, and other formulae automatic. This algorithm is efficient and incremental, thus making it possible to use it in an interactive tool to perform mathematics. We intend to show a few examples of displayed mathematical formulae.

When data is displayed on the screen, it is sensible to mouse interaction, thus making it possible to trigger operations by simply clicking on some sub-formula. With this, we explored new ways to construct mechanically-checked proofs or to manipulate algebraic formulas, based on direct manipulation.

Work in progress in this area of formula layout concerns extensibility, enabling users to adapt notations to their field of interest. We envision several levels of programmability. The first one follows a model of programming by examples: users show the text they want to see on the screen for some formula patterns, but they have no to limit their choice to a fixed set of graphical combinations. More complicated levels require a more precise description of layout computation. For now, this complicated level requires programming in Java.

The layout mechanism also provides ways to include drawings or bitmaps and enriched text in comments.

2.2 Using the mouse to perform proofs

With the CtCoq experiment, we have developed a strong competence around ways to use the mouse to express logical operations.

A feature provided in CtCoq makes it possible to perform proofs of predicate calculus without equality by simply pointing at the sub-expressions of logical formulas that are subject of the next step. We intend to show a sequence of windows, indicating the operations performed with the mouse and the consequence.

A second feature provided in CtCoq makes it possible to perform proofs based on rewriting by dragging data around algebraic expressions. This tool is very much inspired by the direct manipulation provided in tools such as Theorist (a product distributed in Europe under the name MathPlus). We will also show a little scenario of this kind of interaction.

Work in this progress concerns ways to merge the two interaction techniques into one, using the drag-and-drop concept in new ways, etc.

2.3 Rendering machine checked proofs

We have also worked around the rendering of proof attempts as text in a form close to natural language. This makes it possible to present the formal proofs checked by the computer in a manner that also seems natural to readers that are not computer specialists.

This rendering mechanism can actually be used to display proofs-in-progress. In this case, incomplete proofs appear like proof texts in which unfinished parts appear as conjectures. Using mouse interaction facilities, users can then proceed by filling in the gaps, with the illusion of simply transforming some proof text. If our implementation progresses enough, we hope to include an example of incomplete proof text in the poster, and maybe steps that show how the text construction progresses.

Mutatis mutandi, this work can also be used to write functional programs in a programming style close to ML. In this case, the program and the proof of its correctness are developed in parallel.

3 Application fields

3.1 Education

In its most basic use, the proof engine can be used to force users to spell out the proofs they performed. In this respect, it can be used effectively to teach the basic techniques of mathematical reasoning. Also, because proofs can be performed in advance by teachers and collected in some form of interactive books, with high-quality display of formulas, rich comments, and illustrations, this tool can also be used to provide a reference to mathematical knowledge.

As an interactive book, the tool can be used to enliven mathematical results, as students may try to redo proofs under the supervision of the computer, with the possibility to analyze the proofs given as example by the teachers.

3.2 Studying computer algebra algorithms

The Coq proof system is particularly well adapted to reason on computer algebra algorithms. For instance members of our team have developed a machine-checked implementation of Buchberger's algorithm for Gröbner bases. Such mechanically checked algorithms may be integrated in the proof engine itself to make further proofs more efficient, replacing the need for the user to spell out computation steps by the capability to let the computer compute.

We intend to have a box on the poster containing an enumeration of the various computer algebra algorithms that have been studied using our technology (actually mentioning work done with the previous implementation of our user-interface for mathematics on the computer).

4 Conclusion

The general impression that we want to convey is that the CtCoq tool can be used to support the development of interactive books on mathematics, where the constraints of safe reasoning can be checked by the computer. Apart from powerful notations and illustration capabilities, the tool also provides the possibility to check the validity of logical reasoning steps and computations. For students, this makes it possible to supervise exercises, for working mathematicians, this makes it possible to use the computer as an "intelligent" sheet of scratch paper.

Developing the Soliton Explorer: A Problem Solving Environment for Soliton Surface Investigation

Bruce W. Char
Anthony Harrison
Thomas Hewett
Ron Perline
Muksim Rakhimov
Department of Mathematics and Computer Science
Drexel University
Philadelphia, PA 19104 USA
email: bchar@mcs.drexel.edu

We discuss the continuing development of "Soliton Explorer", a Problem Solving Environment for *Soliton Geometry*.

Soliton equations are a recently investigated class of nonlinear ordinary and partial differential equations, which can be linearized via a non-trivial transformation (the so-called spectral transform); as a consequence there are many explicit solutions to these equations, in terms of exponential, trigonometric, and elliptic functions.

Soliton geometry refers to geometric equations for curves and surfaces which can be explicitly described in terms of soliton equations. While there are many such equations, our project's research focus centers on the following partial differential equation describing curve evolution in three dimensions (and related modifications):

$$\gamma_t = \gamma_x \times \gamma_{xx}.$$

This equation is called the smoke ring equation, or the localized induction equation (LIE), from fluid mechanics theory.

The equation LIE and its relatives are sufficiently complicated that a custom Problem Solving Environment tailored to these equations is required. Our PSE, called Soliton Explorer, is an interaction between analytic tools developed within Maple, a Fortran library which produces data files for surface visualization and analysis, and a Java3D Visualization tool for rendering.

The particular roles of Maple include
(i) generation of explicit solutions to related curvature equations;
(ii) integration of curvature equations exactly to obtain desired curves and surfaces (the Sym differentiation trick);
(iii) when possible, further structural analysis of the explicitly described surface (geometry of parameter curves, for example);
(iv) automatic code generation for numerical evaluation of geometric data formulas.

The purpose of the visualization tool includes the investigation of topological and global questions not readily accessible thru algebraic calculations (connectivity, intersection properties, symmetries, etc.)

Our poster for Soliton Explorer illustrates some of the features available within our visualization tool for analysis of surfaces, including colorization options, culling options, and simultaneous rendering styles, which highlight the structure of the given surfaces. We discuss role of Maple as a code generating software component in this system, as well as some of the mathematical discoveries that have been made with the help of this tool.

We conclude with a discussion of current and planned extensions of Soliton Explorer.

This project is funded in part by the National Science Foundation, as grant number CCR-9527130.

The Search for and Classification of Integrable Abel ODE Classes

Edgardo S. Cheb-Terrab
Centre for Experimental and Constructive Mathematics
Simon Fraser University, Burnaby
British Columbia, Canada
ecterrab@cecm.sfu.ca
Theodore Kolokolnikov
Department of Mathematics
University of British Columbia
Vancouver
British Columbia, Canada
tkolokol@math.ubc.ca
Austin D. Roche
Symbolic Computation Group
Department of Computer Science
University of Waterloo
Ontario, Canada
adroche@daisy.uwaterloo.ca

We present a classification of both known and newly discovered integrable cases of Abel ODEs,

$$y' = f_3(x)\,y^3 + f_2(x)\,y^2 + f_1(x)\,y + f_0(x) \qquad (1)$$

where $y \equiv y(x)$. It is well known that Abel ODEs can be organized into *equivalence classes* under the structure preserving transformation

$$\{x = F(t), \quad y(x) = P(t)\,u(t) + Q(t)\} \qquad (2)$$

The standard theory of Abel ODEs is based on certain *invariants* of the transformation (2) which can be expressed in terms of the coefficients $\{f_3,\, f_2,\, f_1,\, f_0\}$ and their derivatives. The nontrivial and therefore interesting case occurs when the invariants are not constant. Liouville [1] showed that by comparing their respective invariants, it can be determined when a given ODE is equivalent to any one of a set of representative ODEs, each of different class, via a transformation (2). This determination would fix the class of the given equation, so that the transformation itself, along with the solution to the canonical representative of the class, would be enough to build the solution to the given equation.

It is clear then that the value of an integration strategy for nonconstant invariant Abel ODEs relies upon the number of distinct integrable Abel ODE classes for which a representative and its solution are known. However, there are relatively few integrable cases with nonconstant invariant found in the literature, scattered widely among the various sources. Many of them have ended up in Kamke [2], but the presentation there lacks a classification; for example some of the equations listed actually belong to the same equivalence class. Furthermore, we believe that it has recently become possible, with the aid of computer algebra tools, to conduct a more systematic and exhaustive search for new integrable cases than has been attempted previously. Our goal was to collect the known integrable Abel ODE classes and search for new ones, building as complete as possible a set, including the corresponding canonical representatives and solutions. Inserting this data into the ODEtools package of Maple would enable the computer to solve automatically any Abel equations which we know how to solve by hand.

One of the interesting aspects of this search has been the discovery of some new *parameterized* equivalence classes. For each different value of the parameter there exists a different class of Abel ODEs.

Our results can be summarized as follows. By analyzing the Abel classes discussed in the pioneering works by Abel, Liouville and Appell [3, 1, 4] we collected four 1-parameter classes - two of which were actually not presented in those works - plus three classes without parameters, two of them by Liouville, and one by Halphen.

Then we analyzed the collection of Abel equations presented in Kamke (69 ODEs) noticing that 42 have constant invariant - hence presenting no interest - and from the remaining set only one of them, number 235 is a new integrable class - without parameters. All the other examples from Kamke are shown to be members of the classes just mentioned, or their solution is not shown in the book and we were not able to obtain it by other means.

By following different approaches we then succeeded in obtaining three more classes without parameters and one more integrable class depending on four parameters, not previously presented in the literature to the best of our knowledge. By splitting this 4-parameter class into different cases, it is possible to show that the class actually consists of a varied set of two, one and zero parameter classes.

The idea used to obtain this new 4-parameter integrable class also leads to a wider 6-parameter nonconstant invariant Abel class all of whose members can be systematically mapped into Riccati ODEs, as well as a recipe to generate new integrable Abel classes from previous ones when they satisfy certain conditions.

References

[1] R. Liouville, Comptes Rendus, Sep. 12, 1887, p460-463.

[2] E. Kamke, *Differentialgleichungen: Lösungsmethoden und Lösungen.* Chelsea Publishing Co, New York (1959).

[3] N.H. Abel, Oevres Complétes II, S.Lie and L.Sylow, Eds., Christiana, 1881.

[4] P. Appell, Journal de Mathématique **5**, 361-423 (1889).

[5] E.S. Cheb-Terrab, A.D. Roche, *Abel ODEs: Equivalence and New Integrable Cases.* (In progress)

[6] E.S. Cheb-Terrab, T. Kolokolnikov, A.D. Roche, *Abel ODEs: A symmetry approach.* (In progress)

Modules for Maple

David Clark and James McCarron

Waterloo Maple, Inc.
{drclark,jmccarro}@maplesoft.com

This poster presents a design for a module system for the computer algebra system Maple [6], and describes a prototype implementation of it. The term "module" is used here in the computer science sense, not the mathematical sense.

The module system design is similar to those found in implementations of most modern functional programming languages, such as Scheme [2] or ML [1]. Modules are first class expressions in Maple that represent the lexical closure of a particular protected scope of computation that exists transiently during the module's instantiation. It is possible to view a module as the result of invoking a thunk that returns some subset of its local environment for use by client code.

A trivial, but important use of modules is the construction of "packages": collections of related procedures that are to be bundled together. Maple currently uses tables to aggregate routines into coherent packages. The ability to aggregate routines into packages is an important part of the management of system namespaces. Thus, modules contribute to better software engineering in Maple.

One source of difficulty with the extant package system that is addressed by modules is the inability to protect the implementation of an API. There was no way to protect data used by a package from tampering. The module system provides access control to module members. It is possible for the programmer to declare that some data or procedures are "private" to the module, and they are inaccessible from outside the modules definition. Only the data or routines that are supposed to be exposed to the user need be.

A new `bind()` command supports the interactive use of modules as packages. It improves on the `with()` command in that it allows the effects of global rebinding of names to be reversed.

We discuss some of the results of experiments that use the prototype module system to create packages.

For the programmer, a distinct, but complementary feature of the design is a new `use` statement. This statement allows one to establish block local bindings for names. Although syntactically similar to the `let` forms found in most functional languages, it is actually much closer to a macro facility as found in Common Lisp [3], in that bindings are resolved at compile time ("simplification time" in Maple). If you say `use( MyModule )` then the bindings exported by the module `MyModule` are visible within the scope delimited by the `use` statement. This facility is specifically designed to support rebinding of operators. Thus, within the body of a `use` statement, the symbol '`+`' can be rebound to mean, for instance, matrix addition.

Modules allow a programmer to implement "objects" that have mutable local state, and that control access to that state. This allows the programmer to construct clean and robust systems of interacting objects. On the other hand, our design avoids trying to implement a full system for object orientation such as one finds in Common Lisp [3]. It is expected that future revisions of the design will add subtyping via ML-style signatures [5].

Some simple examples of programming objects using modules are given.

An interesting problem that arose early in the design was the ability to save a module once instantiated. Since module instantiation can involve arbitrarily large amount of computation, it is desirable to be able to generate a module and then save it in a "library". The existing facilities for saving Maple expressions were unable to cope with the new structures which essentially amount to collections of escaped local variables (closures). Enhancements in the direction of a persistent store were effected to deal with this problem.

Apart from the direct and obvious benefits to software engineering with Maple, one of the most exciting prospects for the use of modules is in programming domains of computation, as found in Axiom [4] and the Maple package `Domains` [7]. This application is still being studied, but we give some simple examples of how this might eventually be done.

References

[1] Cousineau, G., and Mauny, M. *The Functional Approach to Programming.* Cambridge University Press, Cambridge, 1998.

[2] Dybvig, R. K. *The Scheme Programming Language: ANSI Scheme*, second ed. Prentice Hall, Upper Saddle River, New Jersey, 1996.

[3] Graham, P. *ANSI Common Lisp.* Prentice Hall, Englewood Cliffs, New Jersey, 1996.

[4] Jenks, R., and Sutor, R. *AXIOM: The Scientific Computation System.* Springer Verlag, 1992.

[5] Milner, R., Tofte, M., and Harper, R. *The Definition of Standard ML.* MIT Press, Cambridge, MA, 1989.

[6] Monagan, M., Geddes, K., Heal, K., Labahn, G., and Vorkoetter, S. *Maple V Programming Guide.* Springer-Verlag, New York, 1996.

[7] Monagan, M. B. Gauss: A parameterized domain of computation system with support for signature functions. In *Proceedings of DISCO '93* (1993), vol. 722

of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 81–94.

Automatic Construction of an R Matrix

David De Wit
RIMS, Kyoto University, JAPAN
ddw@kurims.kyoto-u.ac.jp

Within mathematical physics, R matrices are of interest in the study of exactly solvable models (i.e. quantum statistical mechanics), and also in knot theory, in that they are solutions of the Yang–Baxter equation. In principle, to each member of a large class of representations of a large class of algebraic structures, there corresponds a unique (up to unitary transformations) R matrix. However, the explicit construction of that R matrix typically presents both theoretical and practical difficulties, thus there are few known explicit examples of them.

Here (fixing m), we describe the automation of the construction of an R matrix corresponding to the 2^m dimensional $(\dot{0}_m \mid \alpha)$ highest weight representation of the *quantum superalgebra* $U_q[gl(m|1)]$. This algebra contains a free complex variable q, whilst the vector space (i.e. module) $V \equiv V_{(\dot{0}_m \mid \alpha)}$ corresponding to the representation contains a free complex variable α. In this case, there are no theoretical difficulties in the construction. The 'R matrix' is actually a rank 4 tensor with 2^{4m} (mostly zero) components, each of which is an algebraic expression in the two complex variables q and α.

Where V has an orthonormal (weight) basis $\{v_i\}_{i=1}^{2^m}$, the tensor product $V \otimes V$ (also a $U_q[gl(m|1)]$ module) has a natural 2^{2m} dimensional (weight) basis $\{v_i \otimes v_j\}_{i,j=1}^{2^m}$. To build the corresponding R, we require a different, very specific (orthonormal) basis $B = \bigcup_{k=1}^{m+1} B_k$ for $V \otimes V$ corresponding to the (orthogonal) decomposition $V \otimes V = \bigoplus_{k=1}^{m+1} V_k$ of $V \otimes V$ into orthogonal $U_q[gl(m|1)]$ submodules V_k. The basis vectors within each B_k are expressed as linear combinations of the vectors $v_i \otimes v_j$, where the coefficients involved are nontrivial algebraic functions of q and α. From the B_k, we may immediately construct projectors P_k onto the modules V_k. R itself is then obtained as the sum $\sum_{k=1}^{m+1} \lambda_k P_k$, where λ_k is the eigenvalue that the second-order $U_q[gl(m|1)]$ Casimir invariant takes on V_k. Both the weights of the V_k and the eigenvalues λ_k are well-known.

Although the dimensions are quite small, manual calculation is demanding (for the case $m = 2$, see [2]). To improve on this situation, we have written a suite of MATHEMATICA functions to automate the entire process for general m. The automation involves the prior writing of the $U_q[gl(m|1)]$ generators and relations in a form that facilitates the normal ordering of strings of generators [4]. This code is currently feasible for the cases $m = 1, 2, 3$ and 4.

From a symbolic programming perspective, the interesting part of the process is the construction of the B_k.

- Initially, we declare a weight basis for V. Knowledge of the intended action of the algebra generators on these basis elements allows us to deduce the matrix elements of the representation, and this information facilitates the construction of the B_k.
- For each module V_k, we must first construct a highest weight vector v_k^+; and this involves the establishment and solution of a system of algebraic equations (known to have a unique solution). The weights of the V_k are included as data within the MATHEMATICA functions, although in principle this could be automatically deduced. From this v_k^+, we construct a spanning set for V_k by the repeated action of (the coproduct of) the $U_q[gl(m|1)]$ lowering generators. Applying a Gram–Schmidt process to this spanning set yields the basis B_k, which typically contains huge expressions.
- In contrast with the vectors in the B_k, the components of R are quite wholesome, although we must typically apply some simplification efforts to see this.

The R matrices obtained may be used in the evaluation of (new, two-variable) polynomial link invariants. A link invariant associated with the $m = 2$ case was introduced in [3], and first evaluated in [2]. The latter work constructed the R matrix manually, and this construction has been used to confirm the correctness of our current algorithm and code. This evaluation also involves automatic symbolic computation, but the code is comparatively pedestrian. The results are exciting for knot theory in that these invariants (for $m \geqslant 2$) are more powerful than their confederates, the well-known HOMFLY and Kauffman polynomials [1].

References

[1] David De Wit. Automatic Evaluation of the Links–Gould Invariant for all Prime Knots of up to 10 Crossings. Under consideration as at May 1999.

[2] David De Wit, Louis H Kauffman, and Jon R Links. On the Links–Gould Invariant of Links. To appear in the *Journal of Knot Theory and its Ramifications.* `math.GN/9811128`.

[3] Jon R Links and Mark D Gould. Two Variable Link Polynomials from Quantum Supergroups. *Letters in Mathematical Physics*, 26(3):187–198, November 1992.

[4] Rui Bin Zhang. Finite Dimensional Irreducible Representations of the Quantum Supergroup $U_q(gl(m|n))$. *Journal of Mathematical Physics*, 34(3):1236–1254, March 1993.

Symbolic Execution and NaNs: Diagnostic Tools for Tracking Scientific Computation

Richard J. Fateman
Computer Science Division, EECS Dept.
University of California at Berkeley
Berkeley, CA 94720-1776, USA
`fateman@cs.berkeley.edu`
`http://www.cs.berkeley.edu/ fateman`

When unexpected results appear as output from your computation, there may be bugs in the program, the input data, or your expectation. In grappling with difficult diagnostic tasks one direction is to seek tools to analyze the symbolic mapping from the input to the output. We are generally ahead in this game if we can understand complex system behavior without necessarily studying detailed program source code. This is true especially if the source is in an unfamiliar language, difficult to understand, or simply unavailable. We explain two tools: time-honored "symbolic execution" which requires some kind of computer algebra system, and a novel modification, NaN-tracking. This is a simplified version of symbolic execution that is more easily implemented in a conventional language like Fortran or C. The principal requirement of this second approach is a competent implementation of a compiler and run-time system. In particular, the language system must provide access to features of the IEEE-754 binary floating-point arithmetic standard [1]. While our own experiments are based in part on an implementation in Lisp, the mechanisms we use should be accessible from languages in nearly every C-based UNIX workstation used for scientific computing.

References

[1] IEEE Computer Society Microprocessor Standards Committee Task P754, a standard for binary floating-point arithmetic, (see, for example, draft 8.0, *Computer 14*, 3 Mar. 1981, 52-63)

Parsing TEX into Mathematics

Richard J. Fateman
Computer Science Division, EECS Dept.
University of California at Berkeley
Berkeley, CA 94720-1776, USA
fateman@cs.berkeley.edu
http://www.cs.berkeley.edu/ fateman
Eylon Caspi
Computer Science Division, EECS Dept.
University of California at Berkeley
Berkeley, CA 94720-1776, USA
eylon@cs.berkeley.edu

Communication, storage, transmission, and searching of complex material has become increasingly important. Mathematical computing in a distributed environment is also becoming more plausible as libraries and computing facilities are connected with each other and with user facilities. TEX is a well-known mathematical typesetting language, and from the display perspective it might seem that it could be used for communication between computer systems as well as an intermediate form for the results of OCR (optical character recognition) of mathematical expressions. There are flaws in this reasoning, since exchanging mathematical information requires a system to parse and semantically "understand" the TEX, even if it is "ambiguous" notationally. A program we developed can handle 43% of 10,740 TEX formulas in a well-known table of integrals. We expect that a higher success rate can be achieved easily.

Gauss-Legendre Quadrature

Greg Fee
Dept. of Mathematics and Statistics,
Simon Fraser Univ.,
Burnaby, B.C., V5A 1S6,
gjfee@cecm.sfu.ca

It is known that gaussian quadrature is one the best numerical methods for computing definite integrals of analytic functions. For example, consider using 101 evaluation points to find π from $\int_0^1 4\,(1+x^2)^{-1}\,dx$. If we use a closed type Newton-Cotes formula, which just integrates the interpolating polynomial which agrees with the integrand at the 101 equally spaced nodes $[0.00, 0.01, 0.02, ..., 1.00]$ then we can compute the integral to 54 significant digits. If we use the 101 point gaussian quadrature formula, then we can compute the integral to 134 significant digits. The evaluation points for an n point Gauss-Legendre quadrature formula are the n roots of the n^{th} Legendre polynomial. One of the drawbacks to using gaussian quadrature, is the problem of computing these roots. We will compare several methods for computing the nodes and weights for an n point Gauss-Legendre quadrature formula at various precisions. As an application , consider the problem of numerically computing a truncated fourier series of a non-periodic analytic function. In this case gaussian quadrature can be faster than the fast fourier transform based algorithm, if we only want a few trigonometric coefficients to high precision. If we compute the fourier series for $\exp(-x)$ on the interval $[0,1]$ up to the terms $\cos(2\pi 2^{10}x)$ and $\sin(2\pi 2^{10}x)$ then the 2048 point gaussian quadrature formula gives the coefficients to 32 digits to the right of the decimal point. Now compare this with an FFT based algorithm. The error in a 2^m point FFT algorithm is $O(2^{-2m})$, so we would need about 2^{27} evaluation points for only double precision (16 digit) accuracy and about 2^{54} evaluation points for quadruple precision (32 digit) accuracy.

Computing Greatest Common Divisors of Polynomial Matrices

Cassidy Gentle
Symbolic Computation Group
University of Waterloo
cgentle@daisy.uwaterloo.ca

Abstract

In control theory the problem of computing greatest common divisors of polynomial matrices arises when trying to compute coprime matrix factorizations of a given transfer function (see Kailath [2]). Much work has been done on this topic in the numerical case, where the coefficients of the polynomials are floating point numbers. We are investigating the extension of these algorithms to the symbolic case.

1 Matrix Greatest Common Divisors

A matrix $R(s)$ is a *Greatest Common Right Divisor (GCRD)* of two matrices $\{N(s), M(s)\}$ if it satisfies the following properties:

1. there exist polynomial matrices $N_1(s)$ and $M_1(s)$ such that
$$N(s) = N_1(s)R(s) \text{ and } M(s) = M_1(s)R(s)$$
2. If $R_1(s)$ is any other right divisor of $N(s)$ and $M(s)$, then there exists a polynomial matrix $W(s)$ such that
$$R(s) = W(s)R_1(s)$$

Matrices $N(s)$ and $M(s)$ are *right coprime* if all their GCRD's are *unimodular*, having determinants that are not a function of s.

2 Computing GCRDs using Gaussian Elimination

Bitmead et. al. [1] provide a method for computing Matrix GCDs which obtains the GCD by performing Gaussian Elimination on an associated generalized Sylvester matrix.

Consider two matrices $N(s)$ and $M(s)$ with polynomial entries, each containing p columns.

We can write $N(s)$ and $M(s)$ as polynomials with matrix coefficients

$$\begin{aligned} N(s) &= N_0 s^m + N_1 s^{m-1} + \ldots + N_{m-1}s + N_m \\ M(s) &= M_0 s^m + M_1 s^{m-1} + \ldots + M_{m-1}s + M_m \end{aligned}$$

where m is the highest degree of all the polynomial entries of $N(s)$ and $M(s)$.

The *generalized Sylvester matrix* of $N(s)$ and $M(s)$ of order k is defined as

$$S_k = \left[\begin{array}{ccccccc} N_0 & N_1 & \ldots & N_m & 0 & \ldots & 0 \\ M_0 & M_1 & \ldots & M_m & 0 & \ldots & 0 \\ 0 & N_0 & N_1 & \ldots & N_m & & 0 \\ 0 & M_0 & M_1 & \ldots & M_m & & 0 \\ \vdots & & & & & & \\ 0 & \ldots & 0 & N_0 & N_1 & \ldots & N_m \\ 0 & \ldots & 0 & M_0 & M_1 & \ldots & M_m \end{array}\right] 2k \text{ block rows}$$

Let k^* be the smallest integer satisfying $\operatorname{rank} S_{k^*} - \operatorname{rank} S_{k^*-1} = p$.

Then the coefficient matrices of a GCRD of N and M can be extracted from certain rows of E_{k^*}, the row-echelon form of S_{k^*}.

However, since k^* is not known in advance, we start with S_1 and add and eliminate 2 block rows at a time until the desired number of zero rows is achieved.

Bitmead et. al. [1] speed up the computation by at each step copying the results from the previous 2 block rows as the rows to be added.

This procedure provides a goood routine for computing matrix GCDs in the case where the coefficient matrices contain floating point numbers; however, numerical stability cannot be guaranteed, since complete pivoting cannot be used. More elaborate methods rely on singular value decomposition.

3 Fraction-Free Methods

When working in an exact arithmetic or symbolic environment, we desire to use fraction-free Gaussian elimination in order to avoid coefficient growth.

When trying to extend the numerical GCD algorithm described in the previous section, we run into the following difficulties:

1. At each step, when we add 2 block rows to the Sylvester matrix, careful bookkeeping is required to keep track of row divisors.

2. The first problem is compounded when trying to take advantage of the structure of the Sylvester matrix, as was done by Bitmead et. al. [1] in the numerical case. The problem here arises from the fact that at some columns of the Sylvester matrix, two pivot elements have been used in the elimination.

3. Even with careful bookkeeping, this method will not remove the entire content of the rows of the Sylvester matrix.

In this poster, we will demonstrate these difficulties with specific examples, and then provide a fraction-free matrix GCD algorithm. We also compare an implementation of this method with an implementation of the fraction-free matrix GCD algorithm described by Beckermann and Labahn [3].

References

[1] R. R. Bitmead, S.-Y. Kung, B. D. O. Anderson and T. Kailath. Greatest common divisors via generalized sylvester and bezout matrices. *IEEE Transactions on Automatic Control*, AC-23(6):1043–1047, December 1978.

[2] Thomas Kailath. *Linear Systems.* Prentice-Hall Information and System Sciences Series. Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1980.

[3] B. Beckermann & G. Labahn Fraction-free Computation of Matrix GCD's and Rational Interpolants. CS Tech Report, University of Waterloo Submitted to *SIAM J. Matrix Anal. Appl.* (1997).